MATTHIAS KRAFT

AUSTAUSCH JURISTISCH RELEVANTER FAKTEN ZWISCHEN HETEROGENEN ANWENDUNGEN

COMPUTER IM RECHT

Herausgegeben von

Prof. Dr. Dr. Jörg Berkemann, Berlin

Prof. Dr. Maximilian Herberger, Saarbrücken

Prof. Dr. Dieter Meurer, Marburg

N.G. ELWERT VERLAG MARBURG

in Zusammenarbeit mit

MEDICONSULT Wiesbaden

Schriftenreihe

Band 8

AUSTAUSCH JURISTISCH RELEVANTER FAKTEN ZWISCHEN HETEROGENEN ANWENDUNGEN

von

MATTHIAS KRAFT

N.G. ELWERT VERLAG MARBURG
1999

CIP-Titelaufnahme der Deutschen Bibliothek

Austausch juristisch relevanter Fakten zwischen heterogenen Anwendungen [Medienkombination] / von Matthias Kraft. - Marburg : Elwert 1999
(Computer im Recht : Bd. 8)
Zugl.: Saarbrücken, Univ., Diss., 1997
ISBN 3-7708-1118-6

Druck und Bindearbeiten: C. H. Beck, München
Printed in Germany

Für Priska

Vorwort

Das Ganze ist mehr als die Summe seiner Teile.

So trivial uns dieser Satz erscheinen mag, so inflationär er zitiert wird, er beschreibt exakt den Schlüssel für einen effektiven Einsatz des Computers am juristischen Arbeitsplatz (und sicherlich auch an vielen anderen Arbeitsplätzen). Die richtigen Informationen und die benötigten Werkzeuge sollten zu jeder Zeit nur einen Fingertick entfernt sein. Jegliche Abfrage von redundanten Informationen, also solchen, die der Computer bereits kennen muß, verbietet sich grundsätzlich unabhängig von technischen Zwängen. Eingegebene Daten sollten genauso jederzeit verfügbar sein, wie die benötigten Fachinformationen. Im Vordergrund der Tätigkeit des (juristischen) Anwenders steht nicht das Werkzeug, sondern die Information.

Der Computer wird also erst dann optimal eingesetzt, wenn man ihn morgens als erstes einschaltet und ihn abends als letztes ausschaltet. In der Zwischenzeit soll er nicht punktuell einzelne Arbeitsschritte unterstützen sondern den gesamten Arbeitsprozeß integral unterstützen.

Die hier vorgelegte Arbeit beschäftigt sich mit einem Detailproblem eines integrierten juristischen Arbeitsplatzes: dem Austausch von Daten zwischen einzelnen Programmen. Dabei wird die nicht ganz triviale Thematik auseinandergesetzt, es wird ein Lösungskonzept erarbeitet und es wird ein Software vorgestellt, die eine prototypische Implementierung des Konzeptes darstellt. Das Ansinnen, in einer Arbeit eine theoretische Auseinandersetzung mit dem Problem des Datenaustausches bis hin zu rechtsmethodischen Fragen zu bieten und gleichzeitig eine softwaretechnischen Lösung entwickeln war ehrgeizig. Es entsprang einer Zeit, in der Kooperation der Disziplinen, insbesondere also Informatik und Jurisprudenz nicht besonders ausgeprägt war. Der programmierende Jurist ist ein Kind dieser Zeit. Es ist zu hoffen, daß eine Besinnung auf Kernkompetenzen und einhergehend eine gute Kommunikation die Zukunft bestimmt. Diese Kommunikation zu ermöglichen kann die Aufgabe von Spezialisten mit intimen Kenntnissen beider Bereiche sein.

Die Arbeit hat trotz eines langen Werdeganges nichts an Aktualität eingebüßt. Die technische Realisierung hat diverse Updates des zugrundeliegenden Betriebssystems unbeschadet überstanden. Die Brisanz des Themas ist aufgrund der steigenden Anzahl von Insellösungen noch gestiegen. Die besondere Eignung des Familienrechts für integrierte Informationsbeschaffung und –weitergabe zeigt sich zuletzt an dem weitgehend formalisierten Unterhaltsverfahren. Die entsprechenden Formulare sind heute bereits elektronisch verfügbar und ausfüllbar. Diese Formulare konnte leider in der Arbeit nicht mehr erwähnt werden. Sie sind aber zur Information auf der zugehörigen CD-ROM abgelegt.

Einem anderen Baustein auf dem Weg zur globalen Integration juristischer Anwendungen widme ich mich in meiner aktuellen Tätigkeit. Hierbei werden unterschiedliche Fachinformationsquellen möglichst einfach verknüpft. Die grundsätzlich positive Resonanz auf den Ansatz, der bereits in der breiten Praxis im Einsatz ist, bestätigt mich in dem weiteren Bemühen, das umrissene Ziel zu erreichen.

Eine ehrgeizige Arbeit, wie die vorliegende, kann nicht ohne intensive Unterstützung vieler Mithelfer zu einem erfolgreichen Abschluß gebracht werden. Besonders gedankt sei Prof. Herberger und Prof. Rüßmann, die mir für zahllose interessante Gespräche zur Verfügung standen. Auch die angenehme, konstruktive Atmosphäre an der Universität des Saarlandes und die anregenden Gespräche mit vielen Mitarbeitern der betroffenen Lehrstühle hatten großen Anteil am Gelingen der Vorhabens. Besonders erwähnen möchte ich Herrn Prof. Philipps, der mein Interesse für

die Materie der Rechtsinformatik wecken konnte, und dem viel zu früh verstorbenen Prof. Suhr. In vielen intensiven und anregenden Gesprächen mit ihm wurde manche für die Arbeit grundlegende Idee geformt.

Nicht zuletzt möchte ich all denen in Familie und Freundeskreis danken, die mich teils durch geduldiges Ertragen, teils durch beharrliches Antreiben bis zum Abschluß der Arbeit gebracht haben.

München, den 10. Oktober 1998

Matthias Kraft

Inhalt

Gliederung

Teil I. Einleitung und Methodik

A. Einleitung

1. Systematische Stellung der Arbeit

Die Rechtsinformatik, der die folgende Arbeit zuzuordnen ist, gehört zu den typisch interdisziplinären Wissenschaften. Sie behandelt nach der wohl vorherrschenden Ansicht die Möglichkeiten der Anwendung moderner EDV für die Lösung von Problemen aus den Gebieten des Rechts[1]. Hierzu gehören sowohl Fragen wissenschaftlicher bzw. rechtsmethodischer Art als auch Probleme aus der alltäglichen juristischen Praxis. Eine juristische Dissertation aus dem Bereich der Rechtsinformatik wird sich angesichts dieser Beschreibung oftmals der Kritik ausgesetzt sehen, primär technische und mithin nicht juristische Probleme zu behandeln.

Das Thema dieser Dissertation unterliegt sicherlich dem ersten Anschein nach in besonderem Maße diesem Vorwurf. Behandelt wird der *Austausch juristisch relevanter Fakten zwischen einzelnen Anwendungen*, etwas technischer formuliert also der Datenaustausch zwischen - juristisch relevanten - Computerprogrammen. Dieser Datenaustausch ist bezogen auf den Computer sicherlich ein technischer Vorgang, der wesentlich von den Möglichkeiten einer technischen Realisierung beeinflußt wird. Betrachtet man jedoch den schnellen Fluß der Entwicklung gerade im Computerbereich, so sind technische Grenzen vernünftigerweise nur zweitrangig in die Erörterungen einzubeziehen. Realisationsprobleme, die heute einer juristisch wünschenswerten Lösung im Wege stehen, können bereits morgen durch die Entwicklung überholt sein. Der Wert einer Arbeit, die lediglich auf das technisch Machbare abstellt, ist deshalb in unüberschaubarem Maße von sekundären Faktoren abhängig.

Diese Arbeit wird zunächst eine *ideale* Methode des Datenaustauschs abgehoben von Fragen der Realisierbarkeit entwickeln. Hieraus ergibt sich jedoch nicht, daß in den theoretischen Betrachtungen technische Fragen gänzlich außen vor gehalten werden. Von entwicklungsabhängigen technischen Problemen sind nämlich diejenigen Schranken zu unterscheiden, die auf unbestimmte Dauer einer Übertragung von juristischen Aufgaben auf den Computer immanent sind. Derartige Grenzen ergeben sich vor allem aus der notwendigen Präzision, die die Maschine von einer derartigen Portierung verlangt. Besteht die Kunst juristischer Tätigkeit nicht selten auch in der - sprachlichen - Verschleierung von Problemen und in der bewußten Aufrechterhaltung von Unklarheiten, so wird sich die Maschine mit derartigen Sprachgebilden auf absehbare Zeit nicht zufrieden geben. Die von ihr erwartete Genauigkeit und Eindeutigkeit des Ausdrucks ist der eigentlich neue Aspekt, den die EDV in die juristischen Überlegungen einbringt.

2. Das Problem des Datenaustauschs aus der Sicht praktischer Rechtsanwendung

Auf dem aktuellen Markt juristischer Computersoftware ist eine Vielzahl von Programmen zu beobachten, die auf die Lösung sehr spezieller juristischer Aufgaben zurechtgeschnitten sind. Als Beispiele mögen etwa Programme zur Durchführung

---

[1] Vgl. *Bund*, S. 11. Eine weitergehende Ansicht möchte der Rechtsinformatik, wie ich meine zu Unrecht, auch die rechtliche Beurteilung von EDV-Sachverhalten zuordnen, so etwa *Fiedler*/*Traunmüller*, **Methodik**, S. 2.

familienrechtlicher[2] oder steuerrechtlicher[3] Berechnungen dienen oder noch wesentlich kleinere Anwendungen etwa zur Berechnung von gesetzlich festgelegten Formeln wie in § 1592 BGB[4]. Auch in diesen Bereich fallen kleinere sogenannte Expertensysteme[5] oder Konsultationsprogramme[6,7]. Ausgangspunkt der Überlegungen in dieser Arbeit ist die Tatsache, daß derartige Programme oftmals vom Benutzer die Eingabe von Daten erwarten und eben solche Daten als Ergebnisse produzieren, ohne sie später anderen Programmen zur systematischen Weiterverarbeitung verfügbar zu machen. Fragt ein Programm **A** beispielsweise nach dem Geburtsdatum einer Person und benötigt ein später aufgerufenes Programm **B** ebenfalls dieses Geburtsdatum, so wird es den Benutzer wiederum danach fragen. Eine Übernahme des bereits einmal erfaßten Datums von **B** aus **A** ist in der Regel bei unterschiedlichen Anwendungen nicht vorgesehen. Bezieht sich ein Schriftsatz, der in der Textverarbeitung **C** erfaßt werden soll, wiederum auf dieses Datum, so muß meist auch hier die Eingabe von Hand wiederholt werden. Diese mehrfachen Eingaben ein und desselben Wertes sind nicht nur für den Benutzer unkomfortabel, sie bergen auch das Risiko eines Eingabefehlers und der damit verbundenen Inkonsistenz der Eingaben in sich. Deshalb ist das Ideal einer integrierten Arbeitsumgebung, daß jede Information an einem Arbeitsplatz lediglich genau einmal vorgehalten wird[8]. Kernproblem ist, daß derzeit keine Normen für Austauschvorgänge existieren, die sich auf die spezifischen Probleme des Austauschs beliebiger juristischer relevanter Fakten spezialisieren. Das Ziel dieser Arbeit ist es, eine entsprechende Norm zu entwickeln, die solche Austauschvorgänge unabhängig von den einzelnen Applikationen ermöglicht.

Der Leser mag an dieser Stelle bereits einen Bruch mit den vorgenannten Prinzipien erkennen. So kann man argumentieren, daß das Problem des Datenaustauschs zwischen inhomogenen Anwendungen einzig deshalb auftaucht, weil die Schaffung eines allumfassenden juristischen Programms derzeit **technisch** nicht möglich aber für die Zukunft erstrebenswert sei. Benötigt das *ideale* Programm letztendlich keine

---

[2] So etwa *Gutdeutsch*, **Familienrechtliche Berechnungen** oder *Hofmmann*, **Unterhalt**. Vgl. *Gutdeutsch, W.*: Informationstechnik..., S. 25 ff.

[3] Die Anzahl der Steuerberechnungsprogramme ist gerade in jüngster Zeit unüberschaubar geworden. Der juristische Gehalt variiert dabei erheblich von einfachen Berechnungsprogrammen bis hin zu komplexen Beratern..

[4] So etwa *Dimbeck*, **JuRech** oder *Hoffmann*, **PC-Praxis für Juristen**.

[5] Im praktischen Einsatz echte Expertensysteme nicht zu beobachten (so jedenfalls bis 1990 *Gordon*, jur-pc 90, 605, 607). Nicht immer besitzen die Programme eine Inferenzmaschine oder eines der anderen Kriterien eines Expertensystems (Vgl. *Haft* u.). Meist handelt es sich tatsächlich um Autorensysteme. Für einen ergebnisorientierteren Begriff des Expertensystems spricht sich *Oechsler* (jur-pc 91, 1205, 1208) aus.

Hier auch nur Beispiele zu nennen, würde dem regen Treiben in diesem Bereich kaum gerecht werden. **Terminus** von *Chung* sei hier als pars pro toto genannt, da es in breitem Maße am Markt verfügbar ist.

Eine ausführliche Übersicht über Expertensysteme und die Anforderungen an solche findet sich bei *Haft*, **Untersuchung der Möglichkeiten des Einsatzes von Expertensystemen zur Unterstütznng von Richtern, Staatsanwälten und Rechtspflegern**. (Anlage 1 zum *Juristar*—Bericht)

[6] Ein beachtenswertes und sehr ausgereiftes Produkt am Markt ist **Arbis**. Vgl. hierzu auch *Ebeling*, **Arbis, jur-pc 91, S. 1237**.

[7] Einige Programmansätze vereinigen Expertensysteme und Berechnungskomponenten wie z.B. JUREX, *Bönniger/Bönniger*, jur-pc 90, 683 ff.

[8] So auch die Beurteilung von *Birkigt* und *Waltl* in ihren Rezensionen von Anwaltssoftware z.B. **JUPITER** in CoR 5/91, S. 19.

anwendungsunabhängige Norm für den Datenaustausch, so wäre eine *ideale* Normierung per se unsinnig. Dem ist entgegenzuhalten, daß die Schaffung eines komplexen homogenen juristischen Programms sowohl an der Inhomogenität des zugrundeliegenden Rechts als auch an der Vielfalt juristischer Tätigkeitsbereiche scheitern muß. Es ist nicht nur unrealistisch, sondern eben auch nicht erstrebenswert alle denkbaren Aufgaben in einem einheitlichen Programm zusammenzufassen. Der modulare Aufbau einer individuellen Arbeitsumgebung paßt sich dagegen besser an die Arbeitsweise der individuellen Benutzer an. Sollte dennoch ein Anbieter ein globales System anstreben, so wird er dies zweckmäßigerweise ebenfalls modular organisieren[9]. In diesem Fall mag das vorgeschlagene System als interne Schnittstelle[10] für die einzelnen Module[11] dienen.

3. Prototypenentwurf

Neben einer theoretischen Konstruktion eines Austauschmechanismus für juristische Fakten enthält diese Arbeit ein praktisches Beispiel für die Realisierung eines solchen Mechanismus. Dieser praktische Ansatz bietet teilweise eine Untermenge an Funktionalität gegenüber dem theoretischen Ansatz, teilweise werden Optionen zugefügt, die in der Theorie unter Zugrundelegung idealer technischer Bedingungen nicht benötigt werden. Mit dem Prototyp soll die Möglichkeit eröffnet werden, die Funktionsweise des erstrebten Austauschmechanismus auszutesten und die sich hieraus ergebenden neuen Anwendungsbereiche zu erkennen.

Es liegt in der Natur eines Prototyps, daß er in unterschiedlicher Weise fehlerbehaftet ist. Zum einen liegen Fehler technischer Art vor, die einen sicheren Dauerbetrieb ausschließen. Zum anderen werden dem Benutzer Begrenzungen auferlegt, die sich aus der mangelnden technischen Realisierbarkeit mit Hilfe der verfügbaren Mittel ergeben. Da diese Fehler jedoch den prinzipiellen Möglichkeiten des Prototyps zur Veranschaulichung einer praktischen Umsetzung der entwickelten Theorie nicht schaden, halte ich sie im Rahmen einer derartigen Arbeit für vertretbar.

4. Zusammenfassung

Die Arbeit beschäftigt sich im wesentlichen mit den Besonderheiten des Austauschs juristisch relevanter Fakten zwischen einzelnen Programmen. Dieser Austauschvorgang wird zunächst aus juristischer Sicht analysiert und für eine *ideale* technische Umgebung beschrieben. In einem zweiten Teil wird ein in einer konkreten Umgebung realisierter Prototyp beschrieben.

B. Präzisierung des Arbeitsziels

An dieser Stelle soll nochmals ausführlicher das Arbeitsziel umrissen werden. Insbesondere sollen die Grenzen des geplanten Systems möglichst eindeutig abgesteckt werden.

1. Beschreibung des Bedarfs

Die Bearbeitung eines juristischen Falls beispielsweise durch einen Rechtsanwalt setzt sich aus einer Kollektion von mehreren Einzelvorgängen zusammen, die sich über unterschiedliche Zeiträume erstrecken und die verschiedensten Tätigkeiten umfassen. Sie reichen von der ersten Aufnahme der Daten des Mandanten durch das

---

[9] Ein interessantes Beispiel hierfür ist das Anwaltssystem **PHANTASY** mit dem Datenaustauschkonzept **AIDA**. Vgl. CoR 4/93, S. 3.

[10] Zur Definition s.S. 11.

[11] Zur Definition s.S. 11.

Büropersonal, die Aufnahme des Sachverhalts durch den Anwalt, das Entwerfen und Schreiben von Schriftsätzen nach erfolgter Würdigung des Sachverhaltes und der späteren Ereignisse bis hin zum Auftreten im Prozeß. Beendet wird die Sache einerseits durch die bürotechnische Abwicklung des Mandats und andererseits durch die Durchsetzung der erstrittenen Entscheidung[12].

Jeder Schritt kann nur unter Einbeziehung von Informationen geschehen. Diese werden entweder abgerufen oder neu erfaßt oder aus vorhandenen Informationen abgeleitet. Ein Teil der Informationen wird bereits bei der ersten Aufnahme des Falls ermittelt. Hierbei handelt es sich meist um die grundlegenden Personaldaten (Adresse, Geburtsdatum persönliche Verhältnisse etc.) und eine grobe Kategorisierung des Sachverhalts. Diese Daten werden in nahezu jedem juristischen Fall meist aus rein praktischen Gründen erhoben. Sie werden oft als *Stammdaten* bezeichnet.

Neben einer kontinuierlichen Veränderung dieser Daten im Verlauf eines Mandats kommen immer neue Informationen aus den einzelnen Vorgängen hinzu. In der Regel benötigen nachfolgende Vorgänge die vorher angesammelten Informationen. Oftmals ist es eine Konstellation von Daten, die einen bestimmten Vorgang erst auslöst. Beispielsweise wird Prozeßkostenhilfe nur dann beantragt, wenn es zum Prozeß kommt und die persönlichen Verhältnisse des Mandanten den Antrag erfolgreich erscheinen lassen. Dem Vorgang *Antrag auf Prozeßkostenhilfe* gehen also die Vorgänge *Prüfung der Erfolgsaussichten der Partei* und *Prüfung der Erfolgsaussichten eines Antrags auf Prozeßkostenhilfe* voraus. Bei letzterem werden bestimmte Informationen zur Subsumtion benötigt. Im Antrag selbst erscheinen dieselben Informationen wiederum und zwar in der Antragsbegründung. Zudem enthält der Antrag das Formblatt *Erklärung über die persönlichen und wirtschaftlichen Verhältnisse*[13], das zum Teil dieselben Informationen benötigt, die etwa bei der Aufnahme des Sachverhaltes abgefragt wurden[14].

Bei der klassischen, nicht EDV-gestützten Arbeitsweise werden diese Informationen je nach dem Zeitraum, der zwischen zwei Vorgängen liegt, durch die Niederschrift in Akten, das Diktat im Diktiergerät oder einfach durch Merken transportiert. Für die korrekte Einordnung einmal erhobener Informationen gibt es vielfältige Möglichkeiten. Neben oft ungeordneten Notizen treten nicht selten Formblätter und andere Kategorisierungsverfahren. Oftmals schlägt die Einordnung jedoch auch fehl. Die Folge ist, daß der Vorgangsbearbeiter bei anderen, nicht zuletzt dem Mandanten nachfragen muß. Dieser wird selten dafür Verständnis aufbringen, daß er über denselben Sachverhalt mehrfach befragt wird. Jedenfalls liegt es auch im Interesse des Anwalts, seine Aktenhaltung so zu organisieren, daß er sein eigenes Haftungsrisiko möglichst klein hält.[15]

Beim Einsatz von EDV ändert sich typischerweise nicht viel an der Zerlegung einer komplexen oft zeitlich und örtlich verteilten Aufgabe in einzelne Vorgänge. Die einzelnen Bearbeitungseinheiten werden intern durch Module organisiert, die den Vorgang unterstützen[16]. Module sind technisch abgegrenzte Programmeinheiten bzw. Teilprogramme eines großen Programms, die für die Erledigung einer spezifi-

---

[12] Sehr praxisorientiert zur gesamten Abwicklung eines Mandats *Commichau*, **Anwaltliche Praxis**.

[13] S.u. S. 246.

[14] Vgl. hierzu *Aufnahmebogen Familiensachen* (S.u. S. 242).

[15] Vgl. zur konventionellen Aktenhaltung *Abel*, **Rechtsanwaltshandbuch 93/94**, K I RdNr. 6 ff.

[16] S. hierzu ausführlich *Wolf*, **Rechtsanwaltshandbuch 93/94**, K III, RdNr. 33 ff..

schen Aufgabe entwickelt sind[17]. Einzelne Module werden über Schnittstellen miteinander verbunden. Schnittstellen sind die Verbindungsstellen zwischen mehreren Systemen[18]. Die Ausgestaltung der Schnittstelle ist ausschlaggebend dafür, wie unabhängig die Module voneinander sind. Je abstrakter Schnittstellen sind, respektive je weniger sie auf die konkrete Ausgestaltung der Module zugeschnitten sind, desto unabhängiger sind diese voneinander. Gehen beispielsweise zwei Module beim Austausch ihrer Daten zwingend davon aus, daß eine bestimmte Anzahl von Informationen in einer definierten Reihenfolge übertragen werden (z.B. *name; straße; plz; ort*) so ist es ausgeschlossen, daß ein später weiterentwickeltes Modul dann auch mehr Informationen weitergeben kann, wenn ein anderes Modul diese Informationen noch nicht verarbeiten kann. Eine flexiblere Schnittstelle erlaubt den Austausch beliebiger Informationen. Das kann etwa dadurch erfolgen, daß eine Information durch einen eindeutigen Namen bezeichnet wird (z.B. *Name=name Straße=straße PLZ=plz Ort=ort*). Die Reihenfolge ist nun ebenso wie die Anzahl der Informationen beliebig. Das empfangende Modul wertet nur diejenigen Informationen aus, die es selbst verwenden kann. Das zweite Verfahren ermöglicht wesentlich einfachere und schnellere Entwicklungszyklen, da im Fall einer Weiterentwicklung eines Moduls nicht gleichzeitig alle anderen ausgetauscht werden müssen, um die Integration zu erhalten.

Wird zunächst gar kein Zusammenhang zweier durch Programme gestützte Vorgänge gesehen, so fungiert der Mensch mit seiner sogenannten *Mensch-Maschine-Schnittstelle* und seiner Fähigkeit zur intellektuellen Umsetzung der Informationen von einem Programm auf das andere als Schnittstelle. Existiert beispielsweise keine technische Schnittstelle zwischen einem Modul zur Unterhaltsberechnung und der Textverarbeitung, auf der ein entsprechender Antrag formuliert wird, so müssen die Ergebnisse vom Computer weg abdiktiert werden. Wird die Schnittstelle so entwikkelt, daß das Berechnungsprogramm einen fertigen Antrag an die Textverarbeitung als ausformulierten Text übermittelt, so werden bei den später folgenden Schriftsätzen die notwendigen einzelnen Informationen aus den Akten oder den verschlungenen Pfaden der Festplatte zu entlocken sein. (Hier liegt also eine zu unflexible technische Schnittstelle vor.) Diese Variante des Datenaustauschs ist zwar sehr flexibel, jedoch entsprechend mühsam und in der gleichen Weise fehleranfällig wie eine konventionelle Verfahrensweise. Sie stellt einen klassischen Medienbruch dar, der für den Benutzer auf Dauer nicht hinnehmbar ist.

2. Eingrenzung des Themas

An dieser Stelle folgt ein Abriß über die wünschenswerten Ziele und eine Einschränkung auf die realisierbaren Möglichkeiten.

a) Verwaltung juristisch relevanter Informationen

In den bisherigen Ausführungen wurden als Beispiele für austauschbare Informationen regelmäßig die Fakten eines Falls verwendet. Dies geschah bereits im Vorgriff auf die Einschränkungen, denen diese Arbeit unterworfen ist. Dennoch sei zunächst aufgezeigt, daß die Einschränkung auf Fakten nicht selbstverständlich ist.

Die Lösung einer juristischen Aufgabe besteht aus der Anwendung von Regeln auf einen konkreten Sachverhalt. Hierzu müssen folgende Schritte durchgeführt werden:
– Ermittlung des Ziels des Vorgehens
– Ermittlung der anwendbaren Regeln

---

[17] S. dazu eine etwas technischere Definition im dtv Computer–Lexikon S. 583 sowie unten S. 11.

[18] Vgl. dtv Computer–Lexikon, S. 779. S.a. S. 11.

– Ermittlung der Fakten eines Sachverhalts
– Anwendung der Regeln auf die Fakten

Der Vorgang wiederholt sich mehrmals ganz oder teilweise. Aufgrund gefundener zielführender Regeln sind weitere Fakten zu ermitteln. Bestimmte Fakten können als Zwischenziele definiert und durch neue Regeln bestimmt werden. Regeln in diesem Sinne sind nicht nur als Regeln formulierte klassische Rechtsquellen nämlich Gesetze, Verordnungen und Richtlinien. Vielmehr wird in der Praxis aus Gerichtsentscheidungen oder aus einschlägiger Literatur der Sinngehalt als Regel extrahiert und auf weitere Sachverhalte angewendet.

Neben der Anwendung von abstrakten Regeln auf konkrete Fakten besteht ein wesentlicher Teil der praktischen juristischen Tätigkeit entsprechend im Auffinden der anwendbaren Regeln und aus der Ermittlung der relevanten Fakten[19]. Dieser Teil ist mit einem nicht unerheblichen Aufwand verbunden.

– Die Ermittlung des Sachverhaltes erfolgt in zeitintensiven Gesprächen sowie aufgrund meist langwieriger Schriftwechsel.
– Das Auffinden von einschlägigen Regeln, insbesondere von entsprechenden Gerichtsentscheidungen oder Rechtsmeinungen, bedingt teilweise aufwendige Recherchen.

In beiden Fällen mag es von Interesse sein, die Ergebnisse des Arbeitsaufwandes optimal für den Ablauf eines Mandats bzw. einer Fallbearbeitung zu perpetuieren. Sowohl die Fakten als auch die bisher gefundenen Regeln sollten im Idealfall im automatischen Zugriff des Anwenders stehen. Der Rechtsanwender sollte sich weder genötigt sehen, einmal ermittelte Fakten lange zu suchen oder erneut zu ermitteln, noch einmal aufgefundene Regeln ein weiteres mal zu recherchieren. Für die Fakten des Sachverhaltes wurde dies bereits ausführlich besprochen. Die Auswirkungen einer Ablage von Regeln soll hier noch an einem Beispiel dargestellt werden.

Angenommen, in einem Fall verlangt die Mutter Unterhalt für ein Kind. Aufgrund bestimmter Umstände in der Familie ergibt sich die Anwendbarkeit einer spezifischen Berechnungsregel. Diese Regel ermittelt der betraute Anwalt nach aufwendigen Recherchen und berechnet hiernach den einzufordernden Unterhalt. Nun fordert die Mutter auch für das zweite Kind Unterhalt. Selbstverständlich kann derselbe Anwalt aufgrund seines eigenen Intellekts die gefundene Regel auch auf das zweite Kind anwenden. Ein eventueller Vertreter müßte möglicherweise nochmals die Regel suchen. Soll dieser Vorgang jedoch optimal durch den Computer unterstützt werden, so muß dieser die einmal gefundene Regel automatisch wieder anwenden. Wird beispielsweise ein neuer Schriftsatz verfaßt, in dem die Höhe des eingeforderten Unterhalts für das zweite Kind anzugeben ist, so müßte diese im Idealfall ohne das Zutun eines Menschen korrekt eingetragen werden. Hierzu muß zunächst die tatsächliche Konstellation der Fakten bezüglich des zweiten Kindes mit dem in der Regel vorausgesetzten Szenario abgeglichen werden, um die Anwendbarkeit der Regel zu erkennen. Weiterhin muß die Regel selbst verfügbar sein und zuletzt auch angewendet werden.

b) Beschränkung auf Fakten

Die vorliegende Arbeit verzichtet auf eine Behandlung der zuletzt angesprochenen Integration von beliebigen Regeln und beschränkt sich auf den Austausch relevanter Fakten.

---

[19] Vgl. *Commichau*, **Anwaltliche Praxis**, RdNr. 2 ff., insbesondere RdNr. 7.

Im Folgenden soll diese Abgrenzung und mit ihr der Titel der Arbeit spezifiziert werden. Wenngleich es notwendig scheint, das Arbeitsfeld auf ein praktikables Maß zu begrenzen, ist dabei jedenfalls als oberste Priorität die Schaffung eines noch praxisrelevanten Systems zu beachten. Einschränkungen sollten also lediglich soweit vorgenommen werden, als ihnen nicht der Vorwurf einer willkürlichen und nicht mehr brauchbaren Vereinfachung entgegenstehen kann.

Bei der Begrenzung der Arbeit auf den Austausch von Fakten ist nicht zwingend auf eine stehende Definition des Begriffs **Fakten** zurückzugreifen. Eine typische Definition dieses Begriffs für den juristischen Bereich ist auch nicht einfach auszumachen[20], wenngleich der Begriff selbst vielerorts Verwendung findet[21]. Es scheint deshalb sinnvoll, den Begriff für eine brauchbare Eingrenzung des hier angestrebten Arbeitsfeldes zunächst eigens zu definieren.

Die nominelle Begrenzung des Arbeitsthemas auf Fakten dient zunächst der Ausgrenzung der Behandlung der bereits erwähnten Regeln. Dabei legt eine Regel eine Vorgehensweise für einen festgelegten Anfangszustand (Input) fest. Diese verfolgt meist das Ziel, einen neuen Endzustand (Output) zu erreichen. Die angeordnete Vorgehensweise selbst ist beliebig. Auch das Gebot einer willkürlichen oder zufälligen Methode ist eine Regel in diesem Sinne[22].

Juristisch betrachtet ist zunächst jede Norm eine Regel: Sie ermöglicht es aus definierten Sachverhaltsvoraussetzungen einen Schluß zu ziehen. Der Schluß führt zu einem Zwischenergebnis (so bei Legaldefinitionen), einer Handlungsanweisung oder es wird der Rechtszustand unmittelbar verändert. Aus der Feststellung beispielsweise, daß zwei Personen einen Vertrag geschlossen haben, ermöglicht die Norm den Schluß auf die Pflichten der einzelnen Vertragspartner. Weiterhin ist jede legislative und exekutive Entscheidung oder Anordnung dann eine Regel, wenn sich hieraus Anweisungen für ein entsprechendes Vorgehen auch in anderen als dem konkret geregelten Einzelfall ergeben. Dies ist jedenfalls dann der Fall, wenn die Entscheidung andere als die Verfahrensbeteiligten bindet[23]. Darüber hinaus sind auch alle anderen Entscheidungen von Gerichten, Erlasse von Behörden oder auch einfache Veröffentlichungen zumindest dann de facto Regeln, wenn sich die Praxis an die dort hinterlegten Handlungsanweisungen tatsächlich hält. Eine faktische Regel liegt also schon dann vor, wenn sich das Handeln eines anderen in einer Situation prognostizieren läßt und man das eigene Handeln tunlichst an dieser Prognose ausrichtet. So ist etwa eine Verwaltungsanweisung dann jedenfalls praktisch eine Regel, wenn sich hieraus ergibt, welche Situation von einer Behörde wie behandelt wird.

Die Art der Regelung einer Norm, eines Vertrages, einer Gerichtsentscheidung oder einer anderen rechtlich oder praktisch relevanten Regel kann erheblich variieren. Klassische Normen ermöglichen oft einen booleschen[24] Schluß vom Vorliegen be-

---

[20] Man mag geneigt sein, den Begriff *Faktum* mit *Tatsache* gleichzusetzen. Hierbei handelt es sich um einen *sinnlich wahrnehmbaren oder feststellbaren Zustand* (Creifelds S. 1216) bzw. um unstreitige Rechtsverhältnisse.

[21] Nach dem Grundsatz *da mihi factum, dabo tibi ius* beispielsweise werden Fakten offensichtlich als **Gegensatz zum Recht**, besser gesagt zur Rechtsanwendung verwendet. Er bleibt aber auch hier undefiniert.

[22] Eine solche Regel wird oft als *Black Box* bezeichnet, wenn die Verfahrensweise unbekannt ist.

[23] So insbesondere im Fall Normkraft von Entscheidungen gem. Art. 94 II GG, § 31 II BVerfGG oder auch bei der Bindungswirkung von Entscheidungen der Revision (§ 565 II ZPO) bzw. der Bindung von Entscheidungen der BGH-Senate untereinander (§ 136 GVG).

[24] Die **boolesche** Logik behandelt lediglich die Werte *wahr* und *falsch* (vgl. S. 38).

stimmter Sachverhaltsvoraussetzungen auf das Normziel[25]. Betrachtet man hingegen § 472 BGB, so beschreibt er einen dezimalen (also nicht booleschen) Dreisatz. Viele weitere Regeln lassen einen Ermessensspielraum oder schreiben den Ermessensgebrauch ausdrücklich vor. Solche Regeln lassen sich mit streng algebraischen Techniken oft gar nicht mehr beschreiben[26].

Die Vielzahl von Möglichkeiten, Regeln zu definieren, findet Ausdruck in einer entsprechenden Vielzahl von künstlichen Regelbeschreibungsmöglichkeiten und - technisch betrachtet - in einer unüberschaubaren Anzahl von Programmiersprachen. Ihnen allen ist gemeinsam, daß sie eine Möglichkeit bieten, auf einem mehr oder weniger abstrakten Niveau Daten zu manipulieren und somit die dargelegte Zustandsveränderung vom Input zum Output zu beschreiben. Allgemeine Programmiersprachen[27] lassen dem Anwender die Freiheit, fast jeden denkbaren Algorithmus mit oft allerdings erheblichem Aufwand zu beschreiben. Spezialisiertere Sprachen[28] sind oft auf einen begrenzten Anwendungsbereich festgelegt, in dem der Aufwand, eine Regel abzubilden, jedoch sehr gering ist. Selbst wenn **die** typisch juristische Programmiersprache noch nicht entwickelt wurde - logische Programmiersprachen wie LISP oder Prolog kommen der juristischen Denkweise in vielen Bereichen wohl sehr nahe - scheint es wenig sinnvoll, sich in den Reigen der Compiler- oder Interpreterentwickler einzureihen. Vielmehr muß man davon ausgehen, daß die Möglichkeiten, Regeln in einer maschinell verständlichen Sprache zu formulieren, sowohl der juristischen als auch der technischen Entwicklung ausgesetzt sind und keine abschließende Behandlung ermöglichen.

Ein Hauptmerkmal einer Regel ist also die Möglichkeit, einen Zustand in einen anderen zu überführen. Die Feststellung dieses Zustandes erfolgt durch die Beschreibung meist mehrerer kombinierter Elemente, die hier als Fakten bezeichnet werden sollen. Fakten sind somit die kleinsten vernünftigen Elemente zur Beschreibung eines realen Zustandes. *Vernünftig* meint, daß hier keine Zerlegung der Wirklichkeit in das physikalisch Mögliche, sondern in das aufgrund der verfügbaren juristischen Regeln Notwendige erfolgen soll. Da die Beschreibung des Anfangs- und des Endzustandes notwendiger Bestandteil der Beschreibung einer Regel ist, ist für die abstrakte Definition von Regeln zunächst auch eine abstrakte Definition der Zustände mithin der zugrundeliegenden Fakten notwendig, wenn auch längst nicht hinreichend.

Diese Definition von Fakten mag den Juristen an die klassische Aufteilung der prozessualen Aufgaben nach dem Grundsatz *da mihi factum, dabo tibi ius* erinnern. Durch sie wird der Rechtssuchende auf die Beschreibung des Sachverhaltes verwiesen, während das Gericht zuständig ist für die Anwendung des juristischen Regelwerks auf den dargelegten Sachverhalt. Der hierbei verwendete Faktenbegriff ist jedoch weitaus enger, als der in dieser Arbeit angewendete. Er schließt nämlich

---

[25] Vgl. zum logischen Schluß z.B. *Koch*/*Rüßmann* S. 14 ff. m.w.N.; *Herberger*/*Simon* S. 54 ff. Vgl.

[26] Dieses Thema ist viel diskutiert. Sicherlich ist zu unterscheiden zwischen Normen mit unklaren Tatbestandsvoraussetzungen (vgl. *Koch*/*Rüßmann*, § 9) und einer nicht eindeutigen Rechtsfolge (vgl. hierzu *Koch*/*Rüßmann*, § 10).

[27] Typische Beispiele sind Pascal, BASIC oder C. Ein neuer beachtenswerter Ansatz ist die Sprache JAVA, die speziell für die Übermittlung von Algorithmen im Internet und ihre Ausführung auf verschiedensten Plattformen entwickelt wurde.

[28] Hierzu gehören z.B. Datenbanksprachen wie SQL und dBase, Sprachen der künstlichen Intelligenz wie PROLOG und LISP oder auch Sprachen zur Gestaltung von multimedialer Information wie OpenScript.

bereits alle Ergebnisse einer Subsumtion aus. Angenommen, ein Kind verlangt von einem Elternteil Unterhalt in einer gewissen Höhe, so ergibt sich die Höhe aus § 1610 BGB. Dabei reicht es prozessual nicht aus, daß das Kind den eingeforderten Betrag benennt. Vielmehr muß es die Umstände beschreiben, die den Richter in die Lage versetzen, den Betrag zu errechnen. Die Höhe des Unterhaltes ist also nicht *factum* im Sinne der genannten Maxime. Wohl aber ist sie ein Ergebnis der Anwendung einer Regel und zumindest Zwischenergebnis eines komplexeren Rechtsanwendungsprozesses. Sie wird in dieser Arbeit - einmal mit Hilfe der Regel ermittelt - als Faktum behandelt.

Ein besonderes Problem ergibt sich dann, wenn eine Regel selbst zum Gegenstand einer weiteren (Metha-)Regel wird. Dies ist der Fall, wenn eine Regel die grundsätzliche Anwendbarkeit anderer Regeln zum Gegenstand hat. Das Grundgesetz enthält teilweise derartige Metharegeln, da es auch einfache Gesetze zum Gegenstand juristischer Kontrolle macht. In diesem Fall werden bestimmte Eigenschaften einer Regel, etwa eine Ungleichbehandlung verschiedener Bevölkerungsgruppen zu Fakten. Diese Arbeit wird das Gebiet der Metharegeln allerdings nicht vertiefen. Vielmehr werden im folgenden lediglich Fakten einer konkreten und individuellen Sachverhaltskonstellation behandelt. Es ist jedoch nicht ausgeschlossen, daß die gewonnenen Ergebnisse letztlich auf Metharegeln ebenso Anwendung finden können.[29, 30]

Die Begrenzung auf den Austausch von Fakten zwischen einzelnen Regeln bietet einen realistischen Rahmen für diese Arbeit. Sie ist deshalb erfolgversprechender als eine Einbeziehung aller vom Juristen für einen Vorgang benötigten Informationen, mithin die entsprechenden Entscheidungsregeln selbst. Dabei definieren sich Fakten im Sinne dieser Arbeit in Abgrenzung zur Regel als Ein– und auch Ausgabedaten für juristische Regeln.

c) Juristische Relevanz

Angesichts der vorangegangenen Ausführungen zur Begrenzung des Arbeitsfeldes auf den Austausch von Fakten, bedarf es nun einer Erläuterung dessen, was unter **juristischer Relevanz** verstanden wird. Ein Blick in den Bereich der EDV zeigt, daß diese sich bei der Abbildung von Daten meist auf unterschiedliche Variationen dreier Datentypen beschränkt: digitale Daten (boole), analoge Daten (Zahlen unterschiedlicher Ausprägung[31]) und einen beliebigen Typ (Text), der diverse Informationen aufnehmen kann. Beschränkt man die Schnittstelle auf eine weitere Normierung auf dieser Ebene, so wäre die Arbeit zu Ende, bevor sie noch richtig begonnen hat.

Thema nun ist die Beschreibung von Fakten in einer Weise, daß sie in die praktische juristische Arbeit einfließen können. Die einfache Aussage, es gibt eine Variable **Gehalt**, die jeweils eine Zahl darstellt, reicht hierfür beispielsweise nicht aus. Die Zahl läßt sich nicht in den Sachverhalt einordnen und mithin nicht weiter verwerten.

---

[29] Zu unterscheiden ist die Regel, die eine andere Regel zum Gegenstand hat, von derjenigen, die lediglich die Anwendung einer anderen Regel beeinflußt. Das Grundgesetz nimmt dabei beide Aufgaben wahr. Seine *Prinzipien* (vgl. *Koch/Rüßmann*, § 10, 3 und § 21) beeinflussen als allgemeingültige Regeln die Anwendung anderer Regeln auf einen konkreten Sachverhalt. Gleichzeitig kann aber eine Regel selbst etwa aufgrund von Art. 1 III, 100 GG zum Gegenstand der Betrachtung, also zum eigentlichen Sachverhalt werden.

[30] Vgl. hierzu auch die Frage der Prädikatenlogik zweiter Ordnung, S. 43.

[31] Vielfach gibt es noch einen eigenen Datentyp für Datum/Zeit–Angaben. Hierbei handelt es sich jedoch lediglich um eine eigene Ausprägung des Zahlenformats.

Die Frage ist, wessen Gehalt? Gehalt in welchem Sinne, brutto oder netto? Wer gibt den Wert vor? Wie verläßlich ist die Quelle? etc.

Juristisch relevante Fakten in diesem Sinne sind also zum einen so zu beschreiben, daß sie ohne weitere Interpretation in einen juristischen Verarbeitungsprozeß zu übernehmen sind. Zum anderen ist wenigstens denkbar, daß die beschriebenen Fakten auch in einen juristischen Verarbeitungsprozeß einfließen werden. Ausgehend von dem juristischen Regelwerk sind Fakten dann juristisch relevant, wenn sie für wenigstens eine juristische Regel eindeutig einen Teil des Anfangs- oder Endzustandes dieser Regel beschreiben. Wenn beispielsweise die Farben der Blumen auf einer Wiese in keiner denkbaren juristischen Regel Bestandteil des Tatbestandes oder der Rechtsfolge sind, so sind sie keine juristisch relevante Information. Dasselbe gilt für unvollständige Informationen wie das bereits bemühte Gehalt. Solange nicht wenigstens feststeht, wessen Gehalt mit einer Zahl gemeint ist, gibt es keine Regel, die dieses Faktum verwerten kann.

Juristische Relevanz wird hier also klar danach definiert, ob ein Faktum Input oder Output für eine juristische Regel darstellen kann.

d) Beschreibung der Domäne Familienrecht

Die Arbeit ist zumindest bezüglich der Wahl der Beispiele auf das Familienrecht fokussiert. Das hat vor allem praktische Gründe:

- Im Familienrecht wird der Grundsatz *iudex non calculat* auf vielfältige Weise durchbrochen. So wurde dieses Rechtsgebiet zu einem Vorreiter bei der Einführung von Computerprogrammen zur Unterstützung abgegrenzter Rechtsanwendungsvorgänge[32]. Weiterhin fließen in diesen Bereich in praktisch erheblichem Umfang allgemeine Berechnungen wie etwa die der Prozeßkostenhilfe ein.
- Das Familienrecht arbeitet mit sehr begrenzten tatsächlichen Szenarien. Dies macht es immer wieder recht einfach, Sachverhalte aufzuzeigen und auch mit EDV-Unterstützung darzustellen. Ziel dieser Arbeit ist es letztlich nicht, die beliebige Komplexität etwa des Gesellschaftsrechts abzubilden, sondern sich auf exemplarische Darstellung zu beschränken.
- Aufgrund der Beschränktheit der typischerweise auftretenden Szenarien sind auch die Verfahren in wesentlichen Teilen sehr stark schematisiert. Diese Vorleistung, die sich aus der klassischen Verfahrensweise ergibt, kann sich ein EDV-unterstütztes Verfahren zunutze machen.

3. Weitere Definitionen

Die folgenden Begriffe nehmen eine zentrale Stellung in der Arbeit ein. Sie sind jedoch bereits so vielfach an anderen Orten definiert worden, daß hier lediglich ein zusammenfassender Überblick erfolgen soll, der für das Verständnis der folgenden Ausführungen geboten erscheint.

a) Daten

Nach DIN 44300 sind **Daten** die kleinste Informationseinheit. Eine Legaldefinition des Begriffs gibt es weder im StGB noch im BDSG. Sofern der Begriff Daten im rechtlichen Bereich auftaucht, wird auch in der Sekundärliteratur entweder auf eine Definition vollends verzichtet oder auf die Definition der DIN[33] verwiesen[34]. Die

---

[32] Vgl. Fn. 2 sowie *Viefhues*, CoR 2/92 S. 21 ff. und ders., jur-pc 93, S. 2179.

[33] DIN 44300, 2.1.13: *Gebilde aus Zeichen oder kontinuierlichen Funktionen, die aufgrund bekannter oder unterstellbarer Abmachungen Informationen darstellen, vorrangig zum Zweck der Verarbeitung oder als deren Ergebnis.*

Definition ist nicht besonders präzise, zumal es an einem eindeutigen Informationsbegriff mangelt. In dieser Arbeit werden die Begriffe Daten und Fakten nahezu synonym verwendet. Von Daten ist eher dann die Rede, wenn Fakten in einer technischen Form auftreten. Fakten beschreiben hingegen die juristische Variante der Informationseinheit.

b) Information

Eine allgemeingültige Definition für **Information** kann nicht gegeben werden[35]. Vielmehr kennt nahezu jede Wissenschaft einen eigenen Informationsbegriff. Der juristische Informationsbegriff ist geprägt durch das Institut der informationellen Selbstbestimmung gem. Art. 2 I i.V.m. Art. 1 I GG einerseits und der Informationsfreiheit gem. Art. 5 I GG andererseits. Im einen Fall werden als Informationen die persönlichen Daten bezeichnet, im anderen Fall geht es um den Vorgang der Beschaffung von Daten. Dies charakterisiert den Zwiespalt, dem die gesuchte Definition ausgesetzt ist. In dieser Arbeit werden als Informationen Daten bzw. Fakten bezeichnet, die zwischen zwei Vorgängen ausgetauscht werden.

c) Vorgang

Als **Vorgang** wird die kleinste sinnvolle Abfolge von Aktionen bezeichnet, die bei Vorliegen eines bestimmten Ausgangszustandes selbständig durchgeführt werden kann, um einen definierten Endzustand herbeizuführen.

d) Modul

Module im Sinne dieser Arbeit sind austauschbare abgeschlossene Einheiten einer Software, die die Bearbeitung eines Vorgangs am Computer unterstützen. Module können selbständig ablaufen (Programme) oder unselbständig als Teil eines größeren Programms implementiert sein (Programmbibliotheken)[36]. Wesentlich ist, daß ein Modul über definierte Schnittstellen verfügt, über die es in eine komplexe Anwendung integriert werden kann.

e) Schnittstelle

Als **Schnittstelle**[37] wird eine Verbindung zwischen zwei an sich selbständigen Einheiten bezeichnet[38]. Die Schnittstelle dient der Kommunikation der Einheiten miteinander. Grundelement einer brauchbaren Schnittstelle ist deren Normierung. Schnittstellen sind deshalb in der Regel so dauerhaft und allgemein anzulegen, daß sie von der Entwicklung der Module unabhängig sind, die auf die Schnittstelle zugreifen.

---

[34] Diskutiert wird vielfach nur die Frage, ob auch Computerprogramme als Daten (i.S. v. § 202a StGB) zu bezeichnen sind. Dies wird (m.E. zu unrecht) von der herrschenden Meinung bejaht (*Lencker/Winkelbauer*, CR 1986, 483ff; *Schönke/Schröder* § 202a StGB, RN 1 ff). Im Zusammenhang dieser Betrachtung führt eine Einbeziehung von Programmen zu einer übermäßigen Dehnung des Datenbegriffs.

[35] DIN 44300, 1.10: *Nachricht; Auskunft; Belehrung* bzw...*physikalische Signale.., die beim Empfänger ein bestimmtes Verhalten bewirken.*

[36] Duden: *austauschbares, komplexes Teil eines Gerätes oder einer Maschine, das eine geschlossene Funktionseinheit bildet. In der EDV wird als Modul ein selbständiger Programmteil erachtet.* Zu technisch ist die Definition im dtv Computer–Lexikon S. 583, das auf die Ebene der Programmierung abstellt.

[37] Im EDV–Bereich wird oft das engl. Wort **Interface** verwendet.

[38] DIN 44300, 1.10: *Gedachter oder tatsächlicher Übergang an der Grenze zwischen zwei gleichartigen Einheiten, wie Funktionseinheiten, Baueinheiten oder Programmbausteinen, mit den vereinbarten Regeln für die Übergabe von Daten und Signalen.*

Die **Mensch–Maschine–Schnittstelle** dient der Kommunikation des Menschen insbesondere mit dem Computer. Elemente dieser Schnittstelle sind beispielsweise der Bildschirm, die Tastatur und die Maus. Die **Maschine–Maschine–Schnittstelle** verbindet meist mehrere Maschinen wie Computer und Drucker miteinander. Diese Arbeit beschäftigt sich fast ausschließlich mit Schnittstellen von Softwaremodulen. Hierbei handelt es sich um die Ein– und Ausgabeelemente eines Moduls, die so gestaltet sind, daß ein anderes Modul die entsprechenden Daten übernehmen kann[39].

Die Beschränkung der Arbeit auf Fakten dient vor allem dazu, eine überschaubare Aufgabe weitestgehend abschließend zu behandeln. Dabei ist der hier verwendete Faktenbegriff weiter gefaßt, als er gemeinhin zivilprozessual gesehen wird. Juristische Relevanz ergibt sich aus der Möglichkeit, ein Fakt in juristischen Regeln weiterzuverarbeiten oder aus ihnen zu gewinnen. Die Fokussierung der Arbeit auf den Bereich des Familienrechts beruht im Wesentlichen auf der existierenden EDV-Unterstützung und der relativ begrenzten Möglichkeit von Fallkonstellationen in diesem Bereich.

C. Methodik

Im folgenden Teil soll eine Methodik entwickelt werden, die einen Weg zur Entwicklung des geplanten Systems weist. Dabei wird aufzuzeigen sein, inwieweit sich eine Abstraktion des theoretischen Idealzustandes von der technischen Verwirklichung vornehmen läßt. Hierzu wird zunächst auf grundsätzliche Arbeitstechniken eingegangen. Anschließend wird die Arbeitsmethode in einzelnen Teilbereichen näher beschrieben.

Sowohl bei der theoretischen Aufarbeitung des Stoffs als auch im Bereich der praktischen Implementierung bietet sich ein dreistufiges Vorgehen an. In der ersten Stufe werden die Einflüsse aus den berührten Teildisziplinen der Informatik und des Rechts analysiert. Ziel dieser Analysen ist das Auffinden von

- Bedürfnissen der späteren Nutzer,
- Schranken insbesondere juristischer Art,
- vorhandenen Lösungsansätzen.

In der folgenden zweiten Stufe wird aufgrund der gewonnenen Erkenntnisse ein eigener Lösungsansatz formuliert bzw. im praktischen Teil unmittelbar realisiert. Schließlich wird in der dritten Stufe die Realisierung anhand der ermittelten Probleme auf ihre Gültigkeit hin überprüft.

1. Deskriptive Darstellung von Ansätzen einzelner Fachgebiete

Beiden Arbeitsabschnitten vorangestellt ist eine überblicksartige deskriptive Darstellung des Forschungsstandes der relevanten Disziplinen. Während diese Beschreibungen bei der theoretischen Behandlung des Themas naturgemäß einen großen Raum einnehmen werden, können sie sich bei der praktischen Implementierung auf wenige Untersuchungen beschränken.

Diese Arbeit setzt sich aufgrund ihrer interdisziplinären Themenstellung an unterschiedlichen Punkten zwangsläufig mit einer Vielzahl wissenschaftlicher Spezialgebiete aus zwei völlig divergierenden Disziplinen auseinander. Dabei behandeln diese Disziplinen, von wenigen Ausnahmen abgesehen, das hier behandelte Thema nicht unmittelbar. Vielmehr ergeben sich lediglich mittelbar aus den grundsätzlich behandelten Fragen Auswirkungen auf die für die angestrebte Entwicklung zu wählende Vorgehensweise. So reicht die Spannweite der berührten Gebiete von Ansätzen der juristischen Methodenlehre über das Prozeßrecht sowie Probleme des gewählten juristischen Fachgebietes, des Familienrechts bis hin zu unterschiedlichsten

[39] Recht ausführlich und verständlich dtv Computer–Lexikon S. 779.

Spezialfragen der Informatik wie etwa der Datenbanktechnik oder der künstlichen Intelligenz. Es würde über den Rahmen einer solchen Arbeit hinausgehen, bei der analytischen Abhandlung der Fachgebiete den jeweiligen Forschungs- und Meinungsstand detailliert und in allen Einzelheiten darzulegen. Diese Einschränkung ist um so stärker notwendig, als wenigstens ein Teilziel der Arbeit die Erstellung eines lauffähigen Prototypen ist.

Die gewählte Vorgehensweise entspricht diesem pragmatischen Hauptziel der Arbeit: Es werden die im Kern berührten Bereiche also gründlicher abgehandelt, als eher mittelbar betroffene oder sehr theoretische Disziplinen. So wird die Logik, die von jeher ein Bindeglied zwischen Informatik und Recht bildet[40], noch einmal unter dem hier behandelten Gesichtspunkt beleuchtet. Weiterhin wird die praktische Vorgehensweise der angesprochenen juristischen Zielgruppe behandelt. Im Bereich der Informatik wird ein Schwerpunkt bei den klassischen Ansätzen des Datenaustauschs und der Datenhaltung gesetzt.

Auf eine präzise Darstellung der Methodenlehre wird hingegen verzichtet. Die Arbeit setzt sich damit zwar dem Vorwurf aus, womöglich keine solide theoretische Basis für dieses weitreichende Problem der Behandlung von Fakten geschaffen zu haben. Andererseits werden durch die Abhandlung der Logik auch andere rechtsmethodische Bereiche gestreift. Dabei bewegt sich die Logik auf einem weitaus konkreteren und für die hier angestrebte Lösung relevanteren Niveau als dies andere abstrakte rechtstheoretische Diskussionen vermögen. Die Betrachtungen der juristischen Logik und der Informatik erfolgen hier aus einem recht ähnlichen Blickwinkel.

Bei der Behandlung der Bereiche der Informatik wird von einer detaillierten Untersuchung der einzelnen Ansätze zur künstlichen Intelligenz abgesehen. Zwar kann über Sinn und Zukunft der künstlichen Intelligenz trefflich gestritten werden[41], die Zielsetzung im juristischen Bereich ist, sofern sie überhaupt einsetzbar ist, leicht zu prognostizieren. Es handelt sich letztlich in den meisten Fällen um Formen der Gewinnung und Darstellung von Regeln. Geht man davon aus, daß auch diese Regeln einen juristisch fundierten Input und Output benötigen, so bedarf es keiner detaillierten Analyse ihres Innenlebens. Ein recht nützlicher Effekt dieses Verzichts ist zudem, daß dem juristischen Leser dieser Arbeit der wohl komplexeste Teil des fachfremden Bereichs erspart bleibt.

2. Vorgehensweise bei der juristischen Analyse

a) Empirische oder normative Analyse

Intention dieser Arbeit ist es, ein an den spezifisch juristischen Bedürfnissen gemessene Systematik des Datenflusses zwischen unterschiedlichen Vorgängen zu erkennen und hieraus Folgerungen für eine technische Implementation zu ziehen. Für eine Analyse des vorhandenen juristischen Materials lassen sich dabei folgende Unterziele festlegen:

- Verifizierung der Notwendigkeit eines Austauschs von Fakten zwischen einzelnen Vorgängen.
- Beschreibung dieses Vorgangs mit Blick auf die Fakten und ihre Aufgaben.

---

[40] Vgl. *Weinberger*, S. VII; *Bund*, S. 11 m.w.N.

[41] Jedenfalls spätestens in den 90er Jahren eine spürbare Ernüchterung bezüglich der Erwartungen an juristische Expertensysteme und KI eingetreten. Vgl. etwa *Haft*, *Juristar*-Bericht, Anhang 1, S. 77 ff; van *Raden*, **JURISTAR**, jur-pc 91, 989 ff. Vgl. auch Wolf, CoR 3/94, S. 174 ff.

– Beschreibung von beachtenswerten juristischen Umgebungsparametern.

Methodisch kann eine derartige Untersuchung jeweils **empirisch** oder **normativ** erfolgen[42]. Eine **empirische** Untersuchung orientiert sich im wesentlichen an der tatsächlichen Arbeitsweise der Zielgruppe, hier also der Rechtsanwälte. Zu diesem Zweck ist es notwendig, die einzelnen Vorgänge bei der Tätigkeit eines Rechtsanwalts sowie den Fluß von Fakten zwischen diesen Vorgängen zu erfassen und auszuwerten. Hieraus lassen sich dann Konsequenzen für die Entwicklung des neuen Systems herleiten. Eine **normative** Analyse geht von dem zugrundeliegenden Normapparat aus. Bei diesem Vorgehen sind hier Vorschriften daraufhin zu analysieren, inwieweit sie zu einer Verteilung der Arbeit auf Einzelvorgänge und einem Austausch der Fakten untereinander zwingen. In diesem Zusammenhang sind sowohl die Grundprinzipien dieser Austauschvorgänge als auch hiermit assoziierte Aspekte herauszuarbeiten und Thesen zur Gestaltung der zu entwickelnden Technik zu formulieren.

Für ein empirisches Vorgehen spricht der starke Praxisbezug der hier bearbeiteten Aufgabe. Letztendlich ist es das Ziel, ein System zu schaffen, welches den Anwalt bei seiner Tätigkeit unterstützt. Die Zielvorstellung der Arbeit ist also die Steigerung der Leistungsfähigkeit des Anwalts durch Optimierung des Datenflusses. Deshalb wäre anzunehmen, daß eine Beobachtung des praktizierten Austauschs von Fakten und eine EDV-technische Unterstützung gerade dieser Praxis zu einer Verwirklichung des Ziels erheblich beiträgt. Aus diesen Gründen neigen auch die Anbieter derzeitig eingesetzter Software zu einer starken Abbildung von praktischen Arbeitsweisen in ihren Programmen. Ebenso orientieren sich die Forderungen aus der Praxis wie etwa die von *Becker*[43] nicht selten an diesen herkömmlichen Arbeitsweisen und den bereits bestehenden EDV-technischen Unterstützungen.

Gegen dieses Verfahren sprechen zumindest dann methodische Bedenken, wenn es das einzig angewendete ist. Schließlich ist es nicht der Sinn der Einführung neuer Techniken, alte Vorgehensweisen kritiklos zu übernehmen, in EDV–Module zu implementieren und somit im wesentlichen die Arbeitsgeschwindigkeit des Anwenders zu steigern. Geht man davon aus, daß der Einsatz der EDV primär leistungssteigernd wirken soll, so genügt ein einfacher Blick auf die Leistungsformel[44] für die Erkenntnis, daß neben einer Verkürzung der für einen Vorgang aufgewendeten Zeit auch die Steigerung der Arbeit, mithin die Optimierung des Arbeitsergebnisses[45] zu einer Leistungssteigerung des Anwenders führt. Die Qualitätsverbesserung ist also ein mindestens ebenso erstrebenswertes Ziel der EDV-Unterstützung und somit des hier angestrebten Arbeitsergebnisses[46] wie die Geschwindigkeitssteigerung. Selbstverständlich wird auch die Qualität zunächst mit einer Umsetzung alter Verfahren gesteigert. So läßt die schnellere Verfügbarkeit und die einer EDV-Organisation immanente stringentere Datenhaltung an sich schon eine Verringerung der Fehlerquote erwarten. Eine nachhaltigere Optimierung des Arbeitsergebnisses ist jedoch

---

[42] Ähnlich *Möller*, S. 3 unter Verweis auf *Hunziker*, S. 72 ff.

[43] Vgl. *Becker*, a.a.O. S. 85 ff.*; Bartsch*, a.a.O.; *Germ*, CoR 2/90, S. 28 ff.

[44] $Leistung = \frac{Arbeit}{Zeit}$.

[45] Definiert man Arbeit frei nach der Formel $W = \delta s \times f$ als $(Endzustand - Ausgangszustand) \times (entgegenwirkende)\ Kraft$, so führt die Verbesserung des Endzustandes, der einzigen vom Rechtsanwender beeinflußbaren Größe auch zu einer Steigerung der Arbeit.

[46] *Berkemann*, **Informationstechnik..**, S. 34.

erst dann zu erzielen, wenn die gewohnten Abläufe anhand einer erneuten juristisch–methodischen Überprüfung neu überdacht und mit Hilfe des Einsatzes des Computers optimiert, respektive dem juristisch Gewünschten angenähert würden.[47].

Eine Unterstützung juristischer Tätigkeit durch EDV sollte also nicht zwingend eine zeitliche Optimierung gewohnter Arbeitsvorgänge erreichen, sondern mindestens ebenso eine Optimierung der geleisteten Arbeit durch Steigerung des Arbeitserfolges sowie Begradigung des zurückzulegenden Weges ins Auge fassen. Die derzeitigen Arbeitsmethoden von Praktikern stellen zwangsläufig lediglich einen Kompromiß zwischen dem juristisch Wünschenswerten und dem mit derzeitigen Mitteln praktisch realisierbaren Verfahren dar. Grenzen für den Spielraum dieses Kompromisses werden durch das Gesetz direkt, durch das anwaltliche Standesrecht sowie die allgemeinen Haftungsregeln vorgegeben. Ebenso ist die Gesetzgebung selbst in vielen Fällen durch Gesichtspunkte der praktischen Umsetzung geprägt. So beruhen Fristen auf Vorgaben des klassischen Postweges, offene Rechtsbegriffe beruhen auf z.T. mangelnder Objektivierbarkeit mit klassischen Methoden[48]. Das Ergebnis einer empirischen Analyse kann aber lediglich in der Festschreibung dieses Zustandes, nicht jedoch in der qualitativen Weiterentwicklung liegen.

Die hierfür notwendige Neustrukturierung, die nicht notwendiger Weise auf eine tatsächliche Veränderung hinauslaufen muß, erfordert sicherlich eine erneute Analyse der normativen Grundlagen. Sie kann deshalb nicht mit Hinweis auf empirische Analysen völlig verworfen werden. Intention einer solchen Untersuchung ist die Sammlung der rechtlich erforderlichen bzw. idealiter wünschenswerten Gesichtspunkte. Sie sollen als Basis für die Entwicklung eines von technischen Fragen möglichst unabhängigen Konzeptes dienen. Dabei müßte selbst bei der Betrachtung von Normen untersucht werden, inwieweit diese durch Berücksichtigung technischer Mankos von dem angestrebten Idealzustand abweichen.

Die normative Methode ermöglicht es vornehmlich, die Vorgaben der gesetzlichen Grundlage aufzudecken. Hieraus ergibt sich zwar der gesetzliche Idealzustand, dieser muß jedoch nicht zwingend auch die praktisch optimale Vorgehensweise darstellen. Insoweit ergibt sich ein weiterer bisher noch nicht berücksichtigter Aspekt der empirischen Analyse gerade daraus, daß die Erfahrung von Generationen von Rechtsanwendern Wege offen legt, wie die gesetzlichen Vorgaben **praktisch** umzusetzen sind. Sie stellt aufgrund ihres Kompromißcharakters nicht nur, wie oben beschrieben, ein Minus gegenüber dem gesetzlich Gewollten dar, sondern eröffnet wegen ihres hohen Erfahrungsschatzes auch ein **Mehr** an Informationen gegenüber einer rein theoretischen Normanalyse. So werden sich normativ kaum Angaben darüber finden lassen, zu welchem Zeitpunkt der Rechtsanwalt am **zweckmäßigsten** welche Fakten von seinem Mandanten erfragt, sondern lediglich, wann sie spätestens benötigt werden. Es ist offensichtlich, daß die Forderung etwa eines Gesetzes[49], ein bestimmtes Faktum in einem bestimmten Verfahrensschritt

---

[47] So wohl auch *Wolf*, CoR 94/3, S. 174; Vgl. zum Ansatz *Rüßmann*, jur-pc 90, 661 ff., der hier nachweist, daß bestimmte Anforderungen des Gesetzes ohne Computer praktisch nicht zu erfüllen sind (ähnlich ders. jur-pc 94, 2828 ff.). Ebenfalls Ansätze für eine Neugestaltung der juristischen Informationsverarbeitung finden sich bei *Becker*, S. 90 - er verweist allerdings auf Standardprogramme - oder auch bei *Nack*, S. 47, die die Unterstützung der Sachverhaltserfassung als eine wesentliche Aufgabe der EDV ansehen. Zu weit gehen jedoch viele Ansätze aus dem Bereich der Expertensysteme, wenn sie mehr auf das technisch Machbare als auf das juristisch oder praktisch Wünschenswerte abzielen.

[48] Vgl. *Rüßmann*, **Denkgesetze.**

[49] Beispielsweise § 130 ZPO.

vorzutragen, für Fragen des praktischen Vorgehens nicht sehr hilfreich ist. Der Anwender von Gesetzen wird vielmehr gut daran tun, sich die notwendigen Fakten rechtzeitig und en bloc zu beschaffen. Ansonsten wäre es notwendig, sich mit dem Mandanten wegen jeder auftauchenden Frage erneut in Verbindung zu setzen[50]. An diesen Überlegungen wird deutlich, daß gerade die Diskrepanz zwischen dem normativ Gewünschten und der praktischen Umsetzung dessen zu einem erheblichen Bedarf an Austauschvorgängen von Informationen, mithin Fakten führt. So muß das zum Zeitpunkt **A** zweckmäßigerweise ermittelte Faktum in einem anderen Vorgang zum Zeitpunkt **B** juristisch, d.h. normativ verarbeitet werden.

Letztlich kann also in diesem Rahmen nicht auf eines der Verfahren - empirisch oder normativ - verzichtet werden. Vielmehr sollten beide Methoden in einer Form angewendet werden, die auch übergreifende Analysen bezüglich der sich aus den unterschiedlichen Ergebnissen resultierenden Folgen ermöglicht. Hierzu ist zunächst eine Vorstellung der verfügbaren Instrumentarien und Materialien notwendig.

b) Empirische Analyse

Für eine empirische Analyse stehen folgende Techniken zur Verfügung:
- Unmittelbare **Beobachtung** der Tätigkeit der potentiellen Anwender vor Ort.
- **Auswertung** von ähnlichen bereits durchgeführten Analysen.
- Analyse des verfügbaren klassischen **Arbeitsmaterials** der Zielgruppe.
- Auswertung von **Lehrbüchern**, deren Zielgruppe der Zielgruppe dieser Untersuchung entspricht.

aa) Beobachtung

Eine direkte statistische Untersuchung der Vorgänge bei der praktischen Tätigkeit ist ein sehr interessanter methodischer Ansatz, um Ergebnisse über die praktische Vorgehensweise des Rechtsanwenders zu erzielen. Es ist ein Versuchsaufbau zu entwerfen, der speziell auf die oben herausgearbeiteten Ziele abgestimmt ist. Mit einer statistisch abgesicherten Zahl von Probanden sind die notwendigen Beobachtungen oder Befragungen durchzuführen und anschließend anhand der korrekten statistischen Testverfahren auszuwerten. Somit ist ein sehr präzises Bild von der **tatsächlichen** juristischen Arbeitsweise zu erwarten. Hingegen erfordert dieses Vorgehen bereits aufgrund der mathematisch notwendigen Versuchsgrößen einen erheblichen Aufwand, der innerhalb dieser Arbeit nicht geleistet werden kann. Als besondere Problemfaktoren sind dabei die unterschiedlichen Größen der Kanzleien sowie die diversen Spezialisierungsmöglichkeiten zu erwarten. Zudem scheint das zu erwartende Ergebnis bezüglich der hier behandelten Probleme auch auf anderem Wege einfacher und korrekter zu erreichen[51]. Eine allgemeine Analyse anwaltlicher Tätigkeit wäre dennoch ein interessantes Vorhaben, allerdings für eine eigenständige Untersuchung[52].

bb) Auswertung von Untersuchungen

Ein einfacherer Weg ist die Auswertung vorangegangener Analysen der oben beschriebenen Art. Problematisch ist hierbei jedoch, daß bisher keine nennenswerten Analysen dieser Art durchgeführt wurden bzw. dem Autor bekannt sind[53]. Zu nen-

---

[50] *Commichau*, RdNr. 2.

[51] S. die folgenden Punkte.

[52] So wurde etwa der Arbeitsablauf bei Gericht im Jahr 1990 durch eine aufwendige Studie der *Kienbaum Unternehmensberatung GmbH* untersucht. Vgl. zu Ansätzen hieraus *Herberger*, jur-pc 91, S. 1259.

[53] Vgl. zu einigen Studien *Endrös, A.*: jur-pc 90, 466 ff.

nen sind hier lediglich im Bereich der Justiz das Projekt JURISTAR sowie das hiermit assoziierte Gutachten der Firma *Kienbaum* über die Organisation der Gerichte, insbesondere der Amtsgerichte (Kienbaum-Gutachten)[54]. Beide Analysen sind auf den hier behandelten Bereich nicht ohne weiteres übertragbar. So unterscheiden sich Arbeitstechnik und Stil von Rechtsanwendern in der Justiz erheblich von denen in der anwaltlichen Praxis. Dies ergibt sich sowohl aus den unterschiedlichen Arbeitszielen als auch aus der unterschiedlich differenzierten gesetzlichen Regelung des Arbeitsablaufs. Letztendlich haben auch die Größenunterschiede und die unterschiedliche organisatorische Struktur der institutionellen Einheiten einen erheblichen Einfluß auf die praktische Tätigkeit.

Gerade dem Thema *(Büro-)Organisation* widmet sich das Kienbaum-Gutachten hauptsächlich, geht dabei allerdings nur unwesentlich auf juristische Probleme ein. Demgegenüber behandelt die JURISTAR–Analyse vornehmlich Fragen der EDV–unterstützten Rechtsanwendung. Die Ausführungen sind jedoch nicht ausreichend tiefgehend in der Form, daß sich für das weitere Vorgehen im Rahmen dieser Arbeit befriedigende Schlüsse ziehen ließen. Beide Berichte können lediglich im Bereich der Bedarfsanalyse für Datenaustauschvorgänge als fundierte Grundlage herangezogen werden. Hingegen kann eine Auswertung dieser Berichte allein einem empirischen Vorgehen nicht genügen.

cc) Auswertung von Arbeitsmaterial

Ein weiterer Ansatzpunkt ist die Analyse allgemein zugänglichen Arbeitsmaterials, dessen sich der Kreis der Zielpersonen bedient. Gemeint ist hiermit neben den Gesetzestexten[55] sekundäres Material wie Formulare und andere Vordrucke, Mustersammlungen oder etwa Praxishandbücher. In einem abgegrenzten juristischen Teilgebiet ist eine derartige Analyse sicherlich mit einem vertretbaren Aufwand zu bewältigen. Sie läßt auch bezogen auf das anvisierte Ziel einen nennenswerten praktischen Ertrag erwarten. Einerseits verfolgen diese Hilfsmittel für den Anwender denselben Zweck wie die an dem angestrebten Austauschvorgängen partizipierenden EDV-Anwendungen. Sie sollen die juristische Leistung des Rechtsanwenders steigern, indem sie innerhalb einer gewissen Zeit mehr (oder bessere) Arbeit ermöglichen. Andererseits bündeln sie sehr häufig in kompakter Weise ein erhebliches Erfahrungswissen speziell über praktisches Vorgehen bei der Rechtsanwendung. So sammelt ein Formular in der Regel auf engem Raum Fakten, die für die Anwendung unterschiedlichster materieller und formeller Vorschriften benötigt werden, und sorgt für ihre rechtzeitige und vollständige Erfassung. Die durch die Formulare etwa bedingte Typisierung der Fälle und Fallbearbeitungsschritte sowie die Zusammenstellung der für diese Falltypen generell anwendbaren Normen und somit der notwendigen Ausgangsfakten stellt eine erhebliche juristische Leistung dar. Sie spiegelt insbesondere die vorgangsbetonte Arbeitsweise wider, von der in dieser Abhandlung ausgegangen wird.

Entsprechend wird den oben aufgezählten Arbeitsmitteln hier eine eigene Untersuchung gewidmet. Dabei wird sich die Analyse des vorhandenen Materials neben der generellen Bedarfsanalyse auch auf Detailfragen zur Realisierung erstrecken. Schwerpunktmäßig werden familienrechtliche Formulare der Hans-Soldan-Stiftung, Schriftsatzentwürfe des Beck'schen Formularhandbuchs sowie der Textbausteinsammlung von *Vespermann*[56] ausgewertet.

---

[54] S. FN. 52.

[55] Eine Analyse der Gesetzestexte entspräche letztlich der normativen Methode.

[56] a.a.O.

Neben klassischen gedruckten Arbeitsmaterialien erfüllen auch einige EDV-Anwendungen, so etwa familienrechtliche Berechnungsprogramme, ganz ähnliche Aufgaben bezüglich Arbeitserleichterung und Leistungssteigerung für den praktischen Rechtsanwender. Oftmals sind diese Produkte auch zumindest optisch an klassische Arbeitshilfen angelehnt. Die EDV sorgt dann im Rahmen des Programms dafür, daß keine unnötigen Rechnungen durchgeführt werden müssen und prüft die Konsistenz und Vollständigkeit der Eingaben. Neben einer technischen Untersuchung[57] werden diese Programme wie die gedruckten Arbeitshilfen als praxisorientierte Aufbereitungen eines juristischen Stoffes ausgewertet.

Weitere verfügbare Instrumente des Anwalts sind sogenannte Anwaltspakete[58]. Sie dienen primär der anwaltlichen Büroorganisation. Eine Unterstützung der für den Rechtsanwender spezifischen Tätigkeit ist eher zweitrangig. In Anbetracht dieser Tatsache erschien eine vollständige Analyse der mehr als zwanzig am Markt befindlichen Produkte wenig erfolgversprechend und mithin zu aufwendig. Hingegen wurden die fachspezifischen Periodika NJW-CoR, CR sowie jur-pc auch mit Blick auf derartige Software systematisch ausgewertet. Diese Auswertung verfolgte im wesentlichen das Ziel, Erfahrungen über praktische Anwenderbedürfnisse und deren Verwirklichung innerhalb der Produkte zu gewinnen. Die Arbeitsergebnisse sind zum Teil für die juristische z.T. aber auch für die spätere EDV-technische Auswertung von Interesse.

dd) Auswertung von praxisorientierten Lehrbüchern

Während juristische Studienbücher sich vornehmlich mit dem Bereich der Rechtsanwendung auf einen gegebenen Sachverhalt beschränken, sind von praxisorientierten Lehrbüchern auch Antworten auf organisatorische Fragen, zeitliche Abläufe einer Mandatsbearbeitung und Hinweise auf Arbeitsmittel zu erwarten. Aus den Arbeitsmitteln selbst, die oben beschrieben wurden, erschließt sich der Zeitpunkt des Einsatzes ebensowenig wie aus einer reinen Analyse der zugrundeliegenden Vorschriften. Lehrbücher dieser Art insbesondere für die anwaltliche Tätigkeit sind äußerst rar. In diesem Rahmen sind die in Zusammenarbeit mit der Rechtsanwaltskammer erschienen Werke von *Comicheau*[59] zu erwähnen, dessen familienrechtlicher Spezialband jedoch bis zum Abschluß der Arbeit nicht verfügbar war. Ebenso übernehmen Rechtsanwaltshandbücher teilweise die beschriebenen Aufgaben. Es bedarf an dieser Stelle keiner eigenen Analyse dieser Werke. Vielmehr werden Erkenntnisse hieraus bei der Beschreibung der Arbeitsmittel selbst eingebracht.

c) Normative Analyse

Eine Aufarbeitung der normativen Grundlagen für den Austausch von Fakten zwischen Vorgängen, insbesondere zwischen Computeranwendungen erfolgt in dieser Arbeit in drei Gebieten:

- Aspekte der Methodenlehre, insbesondere der logischen Grundlagen
- Vorgaben der prozeßrechtlichen Grundlagen
- Analyse des materiellen (Familien-)Rechts

Hingegen wird auf eine gesonderte Untersuchung der rechtlichen Rahmenbedingungen für den elektronischen Datenaustausch etwa in Hinblick auf Aspekte des Datenschutzes verzichtet. Diese Aspekte werden im Regelfall lediglich Einfluß auf Details

[57] S. S. 88.

[58] S. S. 86.

[59] a.a.O.

der technischen Realisierung haben, die jedoch für einen reinen Prototypenentwurf noch ohne Belang sind.

aa) Methodenlehre

Diese Arbeit beschränkt sich im Bereich der Methodenlehre auf die Beschreibung der *juristischen Logik*. Sie hat naturgemäß eine erhebliche Ausstrahlung auf die Rechtsinformatik augesübt. Aufgrund der sehr starken Einflüsse der modernen Logik sowohl auf die Rechtswissenschaft als auch auf die Informatik stellt sie bereits formell eine Art Bindeglied beider Disziplinen dar. Infolgedessen wurden Materialien aus beiden Bereichen ausgewertet[60].

bb) Formelles Recht

Eine Analyse der prozessualen Vorschriften erfolgt mit dem Ziel, das formal notwendige Minimum an Fakten aufzuzeigen, das in unterschiedlichen Vorgängen der Rechtsfindung und -durchsetzung vorgeschrieben ist. Hierzu wurden primär die Gesetze selbst sowie die gängigen Sekundärmaterialien, d.h. Kommentare und Lehrbücher ausgewertet.

cc) Materielles Recht

Als Domäne insbesondere auch der Prototypenbildung wurde das Familienrecht mit Schwerpunkt auf dem Unterhaltsrecht ausgewählt. Diese Auswahl hat zunächst pragmatische Gründe, da im Familienrecht ein erheblicher Bestand an auf dem Markt verfügbarer Software zu verzeichnen ist. Dies ermöglicht eine praxisorientierte Beschreibung von EDV-gestützten Vorgängen. Einer Rechtfertigung für den Einsatz von EDV im Bereich des Unterhaltsrechts bedarf es nicht[62]. Des weiteren stellt das Familienrecht ein sehr stark formalisiertes Verfahren im Bereich des Zivilrechts dar. Dies zeigt sich auch in den für diesen Bereich verfügbaren Formularen und anderen praktischen Arbeitsmitteln. Letztendlich sind die Fallkonstellationen im Bereich Unterhaltsrecht zwar nicht trivial, aber von den Beziehungen der beteiligten Personen her leicht überschaubar. Dies macht dieses Gebiet, wie später noch auszuführen sein wird, zu einer sehr anschaulichen Domäne für die Demonstration EDV-gestützter Arbeitsabläufe.

Im Rahmen der Verwendung des Gesetzes als Beispielsgebiet wird zweistufig vorgegangen. Sofern eine Analyse mit dem Ziel der Darstellung des Ist-Zustandes vorgenommen wird, wird von dem aktuellen Gesetzestext ausgegangen. Dies ist insbesondere bei der Analyse praktischer Arbeitsmaterialien wie Formularen oder auch Programmen notwendig. Da sich jedoch diese Arbeit nicht mit der Schaffung eigener familienrechtlicher Anwendungen zu beschäftigen hat, wird für die Darstellung fachfremder Ansätze von einem vereinfachten, künstlichen Normapparat ausgegangen. Dieser wird zunächst mit Blick auf die Originalnormen entwickelt. An den entsprechenden Stellen wird dann hierauf verwiesen. Das angewendete Verfahren

---

[60] Die mannigfaltige Literatur im Bereich[61] der juristischen Logik zwingt bei der Zitierung zur Mäßigung. Insbesondere ist es nicht das Ziel der Ausführungen, sich mit den wenigen Unterschieden der Darstellungen auseinanderzusetzen. Soweit möglich, werden die sehr komprimierten und verständlichen Arbeiten von *Herberger*/*Simon* **Wissenschaftstheorie** und *Koch*/*Rüßmann* **Begründungslehre** herangezogen. Dabei folgt das erste Werk eher einem praktisch–mathematischen Ansatz. Die Begründungslehre geht stärker von einer juristischen Zielvorgabe aus.

[61] *Herberger*/*Simon*; *Koch*/*Rüßmann*; *Weinberger* (Rechtslogik, Berlin 1989; Normenlogik); *Schneider*, Logik für Juristen; *Engisch*, **Logische Studien zur Gesetzesanwendung**; *v. Kutschera*, **Logik der Normen, Werte und Entscheidungen** etc.

[62] Vgl. z.B. *Mertl*, CoR 4/91, S. 17 ff.

ermöglicht es, beispielhaft Probleme und deren Lösungen zu demonstrieren, ohne daß zu tief auf die Fachfragen des Familienrechts eingegangen werden müßte.

3. Analyse aktueller EDV-Ansätze

Es ist nicht Sinn und Zweck einer derartigen Arbeit, Lösungen für mutmaßlich originär juristische Probleme zu schaffen, die sich bei genauerer Betrachtung als wesentlich allgemeinere Probleme entpuppen und womöglich von einer anderen Fachwissenschaft bereits gelöst sind. Daher erfolgt neben der Beschreibung der juristischen Grundlagen auch eine Beschreibung aktueller Tendenzen der Informatik. Diese Beschreibung wird bereits mit Rücksicht auf den vornehmlich juristischen Charakter der Arbeit beispielhaft und mitnichten vollständig sein, im Sinne einer detaillierten Darstellung aller vertretenen Ansätze.

Primäres Untersuchungsziel ist die Darstellung von z.T. auch praktisch eingesetzter Entwicklungen zum Austausch von Daten. Dabei handelt es sich im wesentlichen um datenbankorientierte Abfragesprachen. Das Gewicht wurde hierbei wiederum auf Ansätze gelegt, die sich mit dem Auffinden von Fakten beschäftigen. Insoweit wurden die wesentlichen Fachberichte und Lehrbücher ausgewertet.

Ein weiteres Ziel ist die Untersuchung der Anwendungen, die an einem Datenaustausch möglicherweise beteiligt sein können. Das sind vor allem Programme bzw. Konzepte zur EDV-gestützten Lösung von Problemen unterschiedlichster Art. Insoweit ist an dieser Stelle lediglich von Interesse, in welcher Form derartige Anwendungen Fakten verarbeiten bzw. Ergebnisse erzeugen. Weniger interessant ist die Frage, wie sie dies tun und ob es überhaupt sinnvoll und wünschenswert ist, derartige Anwendungen im juristischen Umfeld einzusetzen. Die Darstellung beschränkt sich auf die Grundprinzipien unter Berücksichtigung gerade des Aspektes des Datenaustauschs. Ausgewertet wurden auch insoweit die wesentlichen Fachberichte und die grundlegenden Lehrbücher.

4. Implementierung

Die Implementierung eines Programms erfolgt bis zu einem lauffähigen Prototypen. Ziel ist der Nachweis der Funktionalität des beschriebenen Systems. Dabei sind folgende Gesichtspunkte zu beachten:

- Das Programm muß auf einer breiten Basis von Computern lauffähig sein.
- Der Funktionsumfang sollte weitgehend der theoretischen Beschreibung entsprechen.
- Es müssen Beispiele vorgelegt werden, die den Nutzen des Systems wenigstens vermuten lassen.
- Die Programmierung darf nicht den Hauptteil der Arbeitszeit in Anspruch nehmen.

Obwohl bei der Implementierung vielfach ein sehr pragmatischer Ansatz verfolgt wurde, stellte sich die letzte Prämisse als unrealistisch heraus. So wurden vielfach Abstriche bei der Praxistauglichkeit der Komponenten gemacht. Für die Programmierung der Beispiele wurde auf eine einfachere Programmiersprache gewechselt. Zudem wurde der Funktionsumfang des Hauptprogramms und der Beispiele zum Teil stark reduziert.

Das ursprüngliche Anliegen, den Source-code zum Gegenstand der Arbeit zu machen, wurde ebenfalls aufgegeben. Statt dessen werden die Codebeispiele nun anhand von fiktiven Passagen in BASIC abgedruckt, was allerdings der Lesbarkeit für den Computerlaien zugute kommt. Die originalen Codes werden dennoch in der Anlage abgedruckt.

Insgesamt ist das Vorgehen bei der Implementierung von Pragmatismus geprägt. So erfolgte die Auswahl der Arbeits- und Betriebssystemumgebung eher anhand der Verfügbarkeit der entsprechenden Werkzeuge als aufgrund wissenschaftlicher oder didaktischer Überlegungen. Das Vorgehen ist jedoch insofern zu rechtfertigen, als nun fast ausschließlich Standardkomponenten Verwendung finden, die auch für den Leser unproblematisch verfügbar sind. Ebenso ist die notwendige ergänzende oder weiterführende Literatur einfach zu beschaffen. Gerade im Hinblick auf den juristischen Leser wird wo irgend möglich auf handelsübliche und eher laienverständliche Literatur verwiesen.

Teil II. Konzeptentwicklung

A. Der Austausch von Fakten aus der Sicht der Rechtsanwendung

Im ersten Teil der Konzeptentwicklung werden die juristischen Grundlagen der Entwicklung aufgezeigt. Dazu wird überprüft, inwieweit ein praktischer Bedarf für das System besteht und welche Anhaltspunkte für die Entwicklung aus juristischen Vorgaben zu ziehen sind.

1. Bedarfsanalyse

a) Der Austausch von Fakten bei herkömmlicher Rechtsanwendung

An dieser Stelle soll aufgezeigt werden, daß überhaupt ein juristisch begründbarer Bedarf an Austauschmechanismen für Fakten besteht. Dieser Bedarf für den Austausch von Fakten zwischen zwei Vorgängen ist immer dann gegeben, wenn dasselbe Faktum in beiden Vorgängen benötigt wird. Dabei ist die Frage, wann zwei Fakten wirklich identisch bzw. austauschbar sind, später noch zu relativieren.

Die Aufteilung des Rechtsanwendungsprozesses in einzelne, wenn auch sehr grob strukturierte Vorgänge ergibt sich zumindest für das Zivilrecht aus den allgemeinen Verfahrensvorschriften der ZPO. Hier läßt sich zwar keine Regelung zur Mandatsaufnahme, d.h. zum eigentlich ersten Schritt des betrachteten Verfahrens entnehmen. Die §§ 53 und 276 ZPO zeigen jedoch, daß zumindest die Schritte **Klageerhebung** und **schriftliches Vorverfahren** als Abschnitte des Verfahrens vorgesehen sind, bis es überhaupt zu einer mündlichen Verhandlung kommt. Aufgrund von § 93 ZPO wird oftmals noch vor der Klageerhebung ein Schriftwechsel notwendig sein, will der Anwalt oder Mandant nicht das Risiko einer Kostenlast trotz Obsiegens auf sich nehmen[63]. Daraus ergeben sich in der ersten Instanz verfahrenstechnisch folgende Schritte (ein früher erster Termin bleibt unberücksichtigt):

1. Mandatsübernahme
2. Schriftwechsel vor Klageerhebung
3. Klageerhebung (§ 253 ZPO; Antragsschrift in Scheidungssachen, § 622 ZPO)
4. Schriftliches Vorverfahren (§§ 276 ff, 129 ZPO)
5. Hauptverhandlung (§§ 278 ff ZPO) mit anschließendem Urteil (§§ 300 ff ZPO).
6. Durchsetzung (§§ 704 ff ZPO)
7. Beendigung des Mandats

Jeder dieser Schritte ist ein Vorgang, der sich freilich in weitere Einzelvorgänge aufspalten läßt. Die Schritte 2 bis 4 erfolgen über die Formulierung und Auswertung von Schriftsätzen. Die inhaltlichen Erfordernisse für Schriftsätze der Schritte 3 und 4 ergeben sich aus §§ 130 sowie 53 ZPO. Es sind zum einen Formalien wie Parteibezeichnung, Vertreter, Wohnort etc. zum anderen auch Angaben zum individuellen Sachverhalt wie die **tatsächlichen Verhältnisse** (aus der Sicht des Autors des Schriftsatzes) sowie die Behauptungen des Gegners. In allen Fällen handelt es sich um Fakten bzw. präziser um die Behauptung, daß etwas ein Faktum sei[64]. Zumindest die Parteibezeichnung sowie ihr Wohnort werden in jedem Schriftsatz erneut benötigt. Ebenso werden auch Teile des Sachverhaltes wiederholt in Schriftsätzen anzugeben sein. Findet zweckmäßigerweise vor Klageerhebung ein Schriftwechsel statt, so wird dieser Angaben über den geltend gemachten Anspruch und

---

[63] Vgl. *Thomas–Putzo*, § 93 Nr. 3.a).

[64] Vgl. zur Wahrheitspflicht gem. § 138 I ZPO *H. Michel*, S. 11 f.

den zugrundeliegenden Sachverhalt enthalten[65]. Dieselben Angaben müssen bereits wegen § 282 ZPO auch in den vorbereitenden Schriftsätzen enthalten sein.

Diese Betrachtungen zeigen auf sehr grober Ebene ein Erfordernis für den Austausch (und die Ablage) von Fakten vom Aufsetzen eines Schriftsatzes zum nächsten. Ein weiteres Beispiel soll dies auf einer differenzierteren Ebene vertiefen. Betrachtet man die Ermittlung eines Unterhaltsanspruchs nach § 1601 ff. BGB, so ergibt sich bereits aus den gesetzlichen Vorschriften eine Trennung zwischen der Frage nach dem Vorliegen eines Anspruchs (§§ 1601 ff. BGB) und dessen Höhe (§§ 1610 ff. BGB). Die Behandlung der Fragen erfolgt praktisch auch in zwei unterschiedlichen Vorgängen, indem sie sich unterschiedlicher Mittel bedient. Die Frage nach dem Anspruchsgrund ist primär durch eine logische Deduktion, die Frage nach der Höhe durch Abwägung bzw. Berechnung zu lösen. Da jedoch eine Berechnung der Unterhaltshöhe erst dann notwendig ist, wenn überhaupt ein Anspruch dem Grunde nach vorliegt, muß dieses Faktum, also das Ergebnis des ersten Vorgangs in den zweiten Vorgang übernommen werden. Ähnliche Beispiele lassen sich unzählige insbesondere dann finden, wenn man jede Anwendung eines einzelnen Normsatzes als eigenständigen Vorgang beschreibt. An dieser Stelle kann jedenfalls festgehalten werden, daß sich sowohl bei einer Betrachtung der Verfahrensschritte als auch bei einer Betrachtung der Normanwendung ein Bedürfnis für den Austausch von Fakten zwischen den einzelnen Vorgängen aufzeigen läßt.

b) Der Bedarf des Faktenaustausches aus Sicht des juristischen EDV-Anwenders

Unabhängig von der oben abgehandelten normativen Darstellung des generellen Bedarfs eines Faktenaustausches wird auch von Anwendern oft eine Forderung nach Integration erhoben[66]. Die Formulierungen derartiger Bedürfnisse aus Sicht des Endanwenders sind dabei in vielen Fällen an den tatsächlich verfügbaren Werkzeugen orientiert. Sie wirken aus Sicht der Wissenschaft, in der das Wort **Integration** zu einem Modewort geworden ist, oftmals trivial. Dennoch hat weder die Praxis noch die Theorie bisher praktikable und befriedigende Lösungsvorschläge gebracht.

So wird im Bereich von Tests insbesondere der Anwaltssoftware regelmäßig die Forderung formuliert, daß alle Daten in jedem Anwendungsmodul eines **integrierten** Paketes verfügbar sind[67]. Letztendlich beschränkt sich diese Integration auf die klassischen **Stammdaten**, d.h. den Austausch von Adreß-, Akten– und Buchungsdaten. Eine weitergehende Integration der juristisch relevanten Daten wird hingegen auch von den Testern nicht angedacht.

Betrachtet man die wenigen Beiträge von Praktikern speziell zum Thema **Arbeitsplatz des Anwalts**, so beschreiben diese häufig lediglich den Ist-Stand der Softwaretechnik mithin den eigenen Arbeitsplatz und empfehlen oft nur marginale Verbesserungen[68]. Vorträge und Veröffentlichungen aus wissenschaftlicher Sicht, allen voran der JURISTAR Bericht, betrachten regelmäßig einen **integrierten Arbeitsplatz** als erstrebenswertes Ziel[69]. Hierbei wird die Integration jedoch meist auf das

---

[65] Vgl. *Commichau*, RdNr. 25 sowie RdNr. 37 ff.

[66] *Birkigt, Waltl*, CoR 5/91, S. 19; Bzgl. der Geschäftsprozesse bei Gericht: *Viefhues*, **Informationstechnik..**, S. 70.

[67] So regelmäßig Softwaretests in CoR zuerst zu *RA-MICRO* (2/89 S. 5, 10) oder später z.B. zu *ReNoFlex* (1/90, S. 10, 13), *NoRA II* (6/90, S. 9, 14), *pharao* (2/91 S. 15, 20), etc.

[68] *Bartsch* a.a.O.; *Germ* a.a.O.; *Morgenstern* a.a.O.; recht informativ *Becker/Neske* CoR 1/88, S. 24 ff., CoR 2/88 S. 23 ff.

[69] Im Abschlußbericht Nr. 7.5 (S. 249) wird ein besonderer Wert auf die Prozeßintegration gelegt.

jeweilige Werkzeug bezogen gesehen. Eine Datenintegration wird dabei entweder als selbstverständlich erachtet oder gar nicht angedacht. Der juristische Arbeitsplatz wird also als eine Sammlung von EDV-Werkzeugen unterschiedlicher Art, die der Unterstützung der **juristischen** Tätigkeit dienen, betrachtet. Die Werkzeugpalette reicht von klassischer Standardsoftware (Textverarbeitung und Tabellenkalkulation) bis hin zu quasi intelligenten oder nicht linear operierenden Entscheidungssystemen. In den ausgewerteten Veröffentlichungen wurden dagegen keine Wege aufgezeigt, wie die Integration abgesehen von Fragen des Zugriffs auf die Anwendungen zu erfolgen habe. Vielmehr wird die Datenhaltung unter dem Schlagwort **elektronische Akte** oft als individueller Vorgang zur Ablage von Volltextdaten angesehen, nicht jedoch als Instrument der Integration.

Für die Übernahme von Berechnungsergebnissen juristischer Berechnungsprogramme in den späteren Schriftsatz werden aus der Praxis konkrete Forderungen formuliert[70]. Dabei wird allerdings oft erwartet, daß diese Ergebnisse natürlichsprachlich formuliert als Textpassagen in einen Schriftsatz übernommen werden können. Die Qualität dieser Art des Datenaustauschs ergibt sich aus der Transparenz und Genauigkeit der generierten Texte. Die Forderung nach der Herstellung eines Klartextes durch Berechnungsprogramme ergibt sich zum einen daraus, daß dies die einfachste und direkteste Art der Datenausgabe ist und keiner weiteren Normierung bedarf. Deshalb wird dieser Export von den meisten Programmen in allerdings variierender Qualität auch angeboten. Zum anderen kann das Berechnungsprogramm selbst auch am besten den beschrittenen Weg darstellen und ist deshalb prädestiniert für die Zusammenstellung der das Rechenergebnis begründenden Textpassagen. Hierdurch entsteht eine Architektur, in der Berechnung und Textgenerierung in einem einheitlichen Vorgang erfolgen. Eine derartige Architektur ist zwar praktisch, jedoch nicht zwingend notwendig. Da die Texterstellung allerdings ein (wenn nicht sogar **das**) wesentliche Anliegen der Praktiker ist, wird dieser Aspekt in den weiteren Erörterungen besondere Beachtung finden[71].

Sowohl eine Analyse des Verfahrensgangs als auch die Beobachtung der Praxis bestätigt, daß die Weitergabe von Daten zwischen unterschiedlichen Vorgängen eine Selbstverständlichkeit ist. Dabei geht es nicht nur um klassische Stammdaten wie Adressen, sondern auch insbesondere um Einzelheiten des Sachverhalts sowie um Ergebnisse juristischer Auswertungen.

2. Aspekte der Rechtsanwendung

Soll ein System zum Austausch von Fakten zwischen juristischen Vorgängen geschaffen werden, das unabhängig sowohl von technischen als auch juristischen Entwicklungen ist, so empfiehlt sich eine Analyse der methodischen Grundlagen des Rechts. Sie verfolgt in dieser Arbeit das Ziel, die Atomisierung des Rechtsanwendungsprozesses in Einzelvorgänge sowie die Kommunikation dieser Vorgänge darzustellen.

a) Rechtsfindung mit Hilfe modularer Regeln

aa) Modularisierung im deutschen Recht

Die Rechtsanwendung jedenfalls im deutschen Rechtskreis beruht hauptsächlich auf der Anwendung von legislativen Normen. Eine Norm beschreibt regelmäßig einen generell abstrakten Tatbestand mit einer daraus resultierenden Rechtsfolge. Wird der Tatbestand durch einen konkreten, realen Lebenssachverhalt gedeckt, so ordnet die Norm diese Rechtsfolge an. Die Umsetzung des generell abstrakten Tatbestands

---

[70] Vgl. etwa *Viefhues*/*Viefhues*, CoR 1/92, 21, 23.

[71] S. S. 65 ff.

auf den konkreten Sachverhalt wird als **Subsumtion** bezeichnet. Neben Normen, die einen grundsätzlich regelnden Charakter besitzen, also als Rechtsfolge eine Anordnung enthalten oder unmittelbar einen anderen Rechtszustand bewirken, gibt es auch solche, die einen Begriff definieren, Ausnahmen beschreiben oder in anderer Weise zur Präzisierung der regelnden Ausgangsnormen beitragen. Die Anwendung einer einzelnen Norm erfolgt dadurch, daß festgestellt wird, ob ein realer Sachverhalt durch die abstrakte Beschreibung der Norm gedeckt wird. Ist dies der Fall, so kann auf die in der Norm formulierte Rechtsfolge **geschlossen** werden.

Ein derartiger **syllogisitscher** Schluß wird folgendermaßen dargestellt[72]:

Obersatz
Untersatz

Schlußsatz

Der Obersatz ist dabei der generell abstrakte Tatbestand der Norm, der Untersatz ist der konkrete Sachverhalt. Schlußsatz ist die aus der Norm für den konkreten Sachverhalt resultierende Rechtsfolge. Betrachtet werden soll der syllogistische Schluß anhand von § 1601 BGB:

Verwandte in gerader Linie sind verpflichtet, einander Unterhalt zu gewähren.

Der Tatbestand der Norm erschöpft sich hier zunächst in dem Ausdruck *Verwandte in gerader Linie*. Rechtsfolge ist *sind verpflichtet einander Unterhalt zu gewähren*. Betrachtet man nun einen tatsächlichen Sachverhalt, bei dem etwa BERTA geradlinig Verwandte zu ARNO ist, so würde eine Subsumtion wie folgt aussehen:

Obersatz:	Verwandte in gerader Linie sind verpflichtet, einander Unterhalt zu gewähren.
Untersatz:	ARNO und BERTA sind verwandt in gerader Linie
Schlußsatz:	ARNO und BERTA sind verpflichtet, einander Unterhalt zu gewähren

Der hier benannte tatsächliche Lebenssachverhalt füllt also bereits nach seiner Beschreibung den Tatbestand der Norm aus. Das ist zweifellos etwas weltfremd. Die Aussage *jemand ist verwandt in gerader Linie mit jemand anderem* wird üblicherweise nicht originär zu treffen sein. Vielmehr ist regelmäßig lediglich die konkrete Verwandtschaftsbeziehung (Vater, Großvater etc.) bekannt.

Der Sachverhalt ist also entsprechend einer realistischeren Gestaltung zu formulieren: *Berta ist die Mutter von Arno*. Die geforderte Übereinstimmung der hiermit formulierten Tatsache mit dem Tatbestand der Norm läßt sich nur durch eine weitere Prämisse erzielen. Eine derartige Prämisse muß vom Typ *Wenn X dann Y und Z sind Verwandte in gerader Linie* sein, wobei *X* der Tatbestand der zu suchenden Norm ist. Eine derartige Norm findet sich in § 1589 Satz 1 BGB.

Personen, deren eine von der anderen abstammt, sind in gerader Linie verwandt.

Einziges Tatbestandsmerkmal dieser Norm ist die **Abstammung**. Zwei Personen sind dann geradlinig verwandt, wenn eine der beiden von der anderen abstammt. Die Norm besitzt keine Rechtsfolge im Sinne einer Anordnung. Sie stellt fest, wann geradlinige Verwandtschaft im Sinne der im Gesetz folgenden Normen (also insbesondere auch im Sinne von § 1601 BGB) vorliegt. Der Ausgangssachverhalt ist auch hier nicht ohne weiteres auf den Tatbestand abzubilden, da nur bekannt ist, daß BERTA die **Mutter** von ARNO ist. Dies bedeutet nicht automatisch, daß einer vom

[72] Ausführliche Betrachtungen zum Syllogismus finden sich bei *Koch/Rüßmann* insbesondere §§ 3 und 4 sowie bei *Herberger*/*Simon* S. 23 ff.

anderen abstammt[73]. (Spätestens bei der Übertragung einer derartigen Entscheidung auf den Computer fallen solche Probleme stark ins Gewicht.) Im Prinzip wird mindestens eine weitere Regel benötigt, um die Kette der Schlüsse zu vervollständigen: *Wenn X die Mutter von Y ist, so stammt Y von X ab*. Das Gesetz enthält eine solche Regel nicht. Sie entspricht vielmehr dem allgemeinen Wissen und wird als selbstverständlich erachtet. Dennoch wird auch diese Regel zumindest en passant vom Rechtsanwender angewendet. Das Zusammenspiel der einzelnen Regeln läßt sich wie folgt in einer Graphik darstellen:

Verwandte in gerader Linie sind verpflichtet, einander Unterhalt zu gewähren.

Personen, deren eine von der anderen abstammt, sind in gerader Linie verwandt.

Wenn X die Mutter von Y ist, so stammt Y von X ab.
BERTA ist die Mutter von ARNO.

ARNO stammt von BERTA ab.

ARNO stammt von BERTA ab. ⇙

ARNO und BERTA sind verwandt in gerader Linie.

ARNO und BERTA sind verwandt in gerader Linie. ⇙

ARNO und BERTA sind verpflichtet, einander Unterhalt zu gewähren.

Wie *Möller*[74] zutreffend feststellt, kommt diese rekursive[75] Behandlung von Deduktionen sehr der Architektur von EDV-Systemen entgegen.

Für das Problem des Datentransports soll hier jede Anwendung einer Regel als einzelner Vorgang betrachtet werden[76]. Die Vorgänge bauen derart aufeinander auf, daß der Schlußsatz eines Vorgangs gleichzeitig den Untersatz eines nachgeschalteten Vorgangs bildet. Im Sinne dieser Analyse stellt die Verwendung des Schlußsatzes einer Regel als Untersatz einer anderen Regel den Austausch eines Faktums[77] zwischen dem übergeordneten und dem untergeordneten Vorgang dar[78,79]. Abbildung 56 auf der folgenden Seite veranschaulicht die Subsumtion nach diesem Schema in Form eines Flußdiagramms[80]. Die **Unterfunktion** *Subsumtion* wird hier solange in einer weiteren Ebene vertieft (4), bis entweder keine Regel mehr gefunden werden kann (2), die das gewünschte Regelungsziel aufweist, oder ein Abgleich der Merkmale mit dem Sachverhalt gelingt (3).

---

[73] Vgl. auch das Beispiel bei *Koch/Rüßmann*, S. 14 ff., insbesondere hierzu S. 16.

[74] A.a.O. S. 29 f.

[75] *Möller* spricht von **Iteration**. Tatsächlich werden die Deduktionen jedoch nicht aneinandergereiht, sondern jede Deduktion aktiviert eine neue Deduktion, sofern ein Abgleich des Sachverhalts mit der Prämisse nicht möglich ist. Hieraus ergibt sich eine rekursive Subsumtion, wie sie auch im Beispiel erkennbar ist.

[76] *Möller*, S. 17 ff. stellt das beschriebene Vorgehen in einer *Handlungsanleitung zur analytischen Begründungslehre* dar. Er geht davon aus, daß die Subsumtion durch *Zwischensätze* zu atomisieren ist. Sein *iteratives* Vorgehen entspricht etwa der hier dargestellten hierarchischen Subsumtionsstruktur. Für die Differenzierung zum Subsumtionsschema nach *Larenz* sei auf die dortigen Ausführungen verwiesen.

[77] Beachte: als Fakten wurden auch **Zwischenergebnisse** von Vorgängen bezeichnet (vgl. S. 6 f).

[78] Der Austauschvorgang ist in der Graphik mit „⇙“ gekennzeichnet.

[79] *Möller* S. 29 f. beschränkt diesen Austausch von Fakten auf die Bildung von booleschen Wahrheitswerten. Dies entspricht dem aussagenlogischen Ansatz seines Handlungsschemas.

[80] *Möller*, S. 9 u. S. 20 erstellt hier eine „Handlungsanweisung“, die die einzelnen Arbeitsschritte vorgibt.

Für sich betrachtet erscheint die beschriebene modulare Deduktion ein interessanter und schlüssiger Ansatz zur Subsumtion insbesondere unter dem Aspekt der computertechnischen Umsetzung. Nun sollen zur Vervollständigung noch zwei Alternativkonzepte beschrieben werden, die teilweise die beschriebenen Vorteile in Zweifel ziehen.

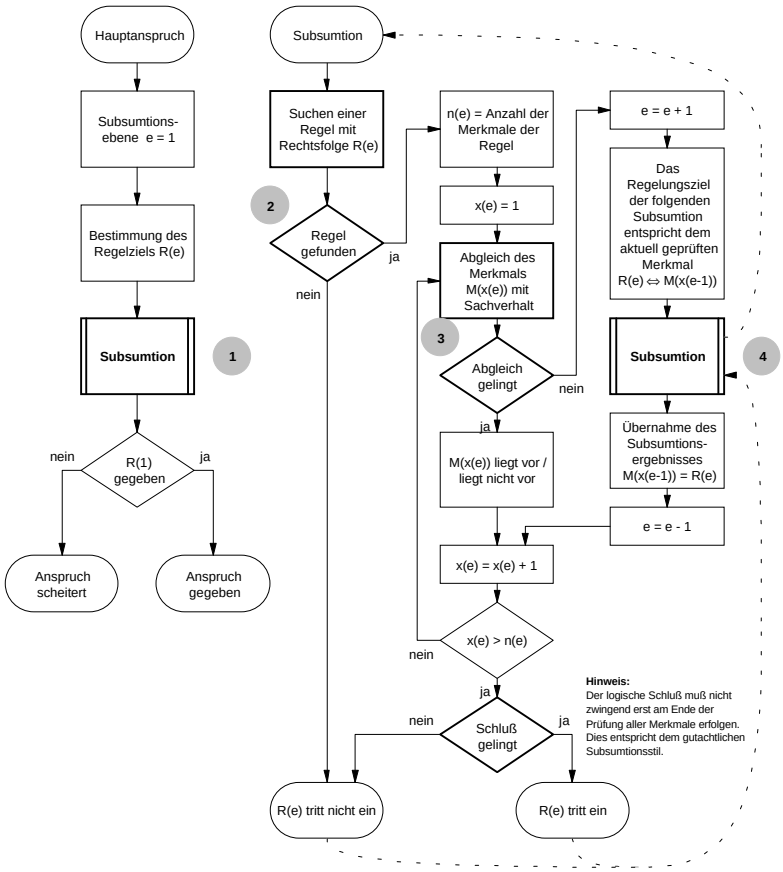


Abbildung 1: Flußdiagramm zum Handlungsschema einer Subsumtion mit rekursiver Regelanwendung. Die Rekursion ergibt sich aus dem Aufruf des Unterprogramms *Subsumtion* aus sich selbst heraus (gestrichelte Linien). Die aktuelle Rekursionsebene wird durch *e* angegeben. Rechtsfolgen werden im Feld $R(e)$, Merkmale der Regeln im zweidimensionalen Feld $M(x(e))$ verwaltet.

bb) Alternative Architektur der Normen

Diese stark atomisierte Architektur der Normen, von der hier ausgegangen wird, ist nicht zwingend. Sie ist aber dem deutschen Recht immanent. Ihre stärkste Ausprägung erhält sie im allgemeinen Teil des BGB. Sie ermöglicht es je nach Lage des Falls mit unterschiedlichen Fakten an einem beliebigen Punkt die Subsumtion zu beginnen. So war es im ersten Beispiel möglich, das Faktum *Arno ist geradlinig verwandt mit Berta* anzunehmen und so die Zwischenschritte des zweiten Beispiels zu sparen. Denkbar wäre anstelle des beschriebenen komplexen Regelapparates auch eine einfache Norm die bestimmt:

Alternativnorm:
Eine Mutter und ihr Kind sind einander zum Unterhalt verpflichtet.

Hierunter wäre der zweite Sachverhalt leicht zu subsumieren:

Eine Mutter und ihr Kind sind einander zum Unterhalt verpflichtet.
BERTA ist die Mutter von ARNO

ARNO und BERTA sind verpflichtet, einander Unterhalt zu gewähren

Probleme ergäben sich jedoch dann, wenn ARNO behauptete, sein Vater CLEMENS sei ihm zum Unterhalt verpflichtet. Nimmt man zunächst an, es stünde fest, daß ARNO das Kind von CLEMENS ist. Ausgehend von der Regelung des BGB bedürfte es nun einer weiteren Regel:

Wenn X der Vater von Y ist, so stammt Y von X ab.

Eine explizite Formulierung dieser Regel kann aufgrund ihrer Selbstverständlichkeit unterbleiben. Der Fall ist also mit dem vorgegebenen Normapparat lösbar. Anders hingegen verhält es sich im Alternativgesetz. Dieses muß explizit erweitert werden.

Alternativnorm:
Eine Mutter und ihr Kind sind einander zum Unterhalt verpflichtet. Ein Vater und sein Kind sind einander zum Unterhalt verpflichtet.

Verkompliziert wird das Problem, wenn nicht feststeht, ob CLEMENS der Vater von ARNO ist. Das BGB stellt mit den §§ 1591 bis 1600o Normen zur Verfügung, die einen Schluß auf die Vaterschaft des CLEMENS zulassen. Sie stützen sich auf wesentlich komplexere als die bisher behandelten Tatbestände. Systematisch können sie jedoch sehr einfach in das Subsumtionsschema eingeklinkt werden.

Verwandte in gerader Linie sind verpflichtet, einander Unterhalt zu gewähren.

Personen, deren eine von der anderen abstammt, sind in gerader Linie verwandt.

Ein eheliches Kind stammt von seinem Vater ab.

Ein Kind, das nach der Eheschließung geboren wird, ist ehelich, wenn die Frau es vor oder während der Ehe empfangen hat und der Mann innerhalb der Empfängniszeit der Frau beigewohnt hat.
BERTA und CLEMENS haben am 1.4.90 geheiratet.
ARNO ist am 5.9.92 geboren.[81]

Als Empfängniszeit gilt die Zeit von dem einhunderteinundachtzigsten bis zu dem dreihundertzweiten Tage vor dem Tage der Geburt des Kindes, mit Einschluß sowohl des einhunderteinundachtzigsten als auch des dreihundertzweiten Tages.
ARNO ist am 5.9.92 geboren.[81]

Die Empfängniszeit reicht vom 8.11.1991 bis zum 8.3.1992

Die Empfängniszeit reicht vom 8.11.1991 bis zum 8.3.1992
CLEMENS hat der BERTA in der Zeit vom 1.4.90 bis zum 1.1.93 beigewohnt

---

ARNO ist das eheliche Kind von BERTA und CLEMENS.

[81]Hier wird als Besonderheit dasselbe Faktum in zwei unterschiedlichen Regeln benötigt.

ARNO ist das eheliche Kind von BERTA und CLEMENS.
ARNO stammt von CLEMENS ab.
ARNO stammt von CLEMENS ab.
ARNO und CLEMENS sind verwandt in gerader Linie.
ARNO und CLEMENS sind verwandt in gerader Linie.
ARNO und CLEMENS sind verpflichtet, einander Unterhalt zu gewähren.

Wollte man dem Alternativbeispiel folgen, so wäre eine erneute Erweiterung der Regel erforderlich. Die Formulierung der Regel müßte alle oben bezeichneten Aspekte beinhalten, insbesondere die Frage des Heirats- und Geburtszeitpunktes sowie des Empfängniszeitraums.

Der Vorteil in der Modularisierung des Regelapparates ergibt sich aus folgendem:
- In unterschiedlich problematischen Sachlagen kann an unterschiedlichen Stellen des Regelapparates eingestiegen werden.
- Aus Sicht der Legislative bedarf es genau einer Regel, um einen identischen Sachverhalt zu behandeln und genau einer Änderung dieser Regel um die Wirkung an allen benötigten Stellen im Regelapparat herbeizuführen.
- Aus Sicht des Anwenders können die Ergebnisse, die sich aus der Anwendung einer Regel ergeben, in einer Vielzahl von weiteren Vorgängen Verwendung finden.

cc) Alternativer Subsumtionsansatz

Ein anderer Ansatz für den Subsumtionsvorgang ergibt sich nämlich, wenn anstelle des modularen Aufbaus, wie er oben beschrieben wurde, die hierarchisch untergeordneten Prämissen in die übergeordnete Prämisse integriert werden. Auf diese Weise wird zunächst eine sehr komplexe Regel mit einer bestimmten oft sehr großen Anzahl von Merkmalen M_1 bis M_n geschaffen, unter die der Sachverhalt letztlich subsumiert wird[82]. Ein Transport von Zwischenergebnissen zwischen modularen Regeln findet dann nicht statt. Dennoch ist es, selbst wenn man dieser Ansicht folgt, notwendig, quasi in einzelnen Vorgängen die Sachverhaltsmerkmale den Merkmalen des Tatbestands zuzuordnen[83]. Die logische Auswertung der Ergebnisse dieser Zuordnung zu einem Gesamtergebnis, also zur Erkenntnis der Rechtsfolge stellt für sich einen eigenen Vorgang dar. Zwischen dem Prozeß der Zuordnung und des Schlusses findet dann ein Datenaustausch statt.

Im folgenden wird davon ausgegangen, daß sich das Rechtssystem aus einer Vielzahl von modularen Regeln zusammensetzt. Dabei definieren eine Vielzahl von Regeln lediglich Sachverhaltsmerkmale der eigentlichen Anspruchsgrundlagen oder sogar weiterer Definitionen. Bei der Subsumtion werden solange rekursiv Definitionen zu Merkmalen einer Regel gesucht, bis sich die Prämisse einer Regel mit dem Tatbestand abgleichen läßt. Die Überführung eines Schlußsatzes von einem Schluß in den Untersatz des übergeordneten Schlusses wird hier als ein Austausch eines Faktums zwischen den Vorgängen gewertet.

---

[82] So wohl die Ansicht von *Larenz*, S. 275, der eine Subsumtion nur dann für möglich hält, wenn der Tatbestand T durch Merkmale M_1 bis M_n definiert wird. Dabei geht er von einer äquivalenten Beziehung der definierenden Merkmale und des Definiendums

$\sum_{x=1}^{n} Mx \leftrightarrow T$ aus (vgl. S. 216).

[83] So wohl auch *Larenz*, S. 273, der die „Aussage“ über einen Sachverhalt - unter dem Gesichtspunkt der Merkmale - von dem eigentlichen Lebenssachverhalt unterscheidet.

b) Der Begriff als *Datenträger*

Geht man davon aus, daß im Subsumtionsprozeß Fakten zwischen einzelnen Regelanwendungen ausgetauscht werden, so stellt sich die Frage nach der entsprechenden Schnittstelle. Dieser soll nun nachgegangen werden.

aa) Modularisierung und Informationsaustausch

Die Modularisierung des Regelwerkes hat eine Vernetzung der Regeln zur Folge. Zwischen mehreren Regeln bestehen Verbindungen, auf denen quasi das Ergebnis der Anwendung einer Regel zu einer anderen transportiert wird. Liegt zwischen der Anwendung der Regeln eine zeitliche Differenz, müssen die gewonnenen Zwischenergebnisse abgelegt und erneut verfügbar gemacht werden. Hat man beispielsweise die Vaterschaft einer Person gegenüber einer anderen festgestellt, so ergeben sich hieraus eine Vielzahl von Ansprüchen, die womöglich erst sukzessive geltend gemacht werden. Eventuell treten auch erst im Laufe der Zeit weitere anspruchsbegründende Merkmale hinzu. Diese Vernetzung von Regeln ähnelt sehr dem Netzwerk von EDV-gestützte Vorgängen, welches hier zu entwickeln ist. Deshalb ist vor allem interessant, wie die Verbindungen der Regeln im juristischen Bereich realisiert werden.

Eine Verbindung zweier Regeln besteht nach den bisherigen Erörterungen dadurch, daß die Folgerung *R* einer Regel einem Merkmal M_x der Prämisse *S* einer anderen Regel entspricht. Abbildung 56 (Seite 27) zeigt diesen Zusammenhang bei Punkt 4 anhand der Gleichungen $R(e) \Leftrightarrow M(x(e))$ und $M(x(e)) = R(e)$. Mit dem ersten Ausdruck wird bestimmt, daß die Rechtsfolge der untergeordneten Regel inhaltsgleich mit dem Merkmal der übergeordneten Regel sein soll. Der zweite Ausdruck formuliert den Austausch des durch die Subsumtion gewonnenen Ergebnisses.

Zwei identische Inhalte definieren sich über einen gemeinsamen **Begriff**. Ein **Begriff** in diesem Sinne stellt eine abstrakte inhaltliche Bedeutung (**Intension**) ungeachtet des sprachlich verwendeten Ausdrucks oder seiner Ausprägung etwa in Beteiligten oder Gegenständen (**Extension**) dar[84]. *Larenz* geht von einem Begriff jedenfalls im *engeren Sinne* nur dort aus, *wo es möglich ist, ihn durch die vollständige Angabe der ihn kennzeichnenden Merkmale eindeutig zu definieren*[85]**.** Ansonsten spricht er vom *Typus*. *Koch*/*Rüßmann* sprechen hingegen von *vagen* Begriffen[86] und *Typusbegriffen*[87], schließen also implizit in eine Umschreibung von *Begriff* auch nicht logisch definierbare Inhalte ein. Die Diskussion soll an dieser Stelle nicht vertieft werden. Entgegen *Larenz* soll hier nicht davon ausgegangen werden, daß ein Begriff nur dann vorliegt, wenn er sich aufgrund einer **äquivalenten** Beziehung mit mehreren **konjugierten** Definitionsmerkmalen definieren läßt[88,89]. Eine solch enge Definition von *Begriff* erscheint jedenfalls für die folgenden Erörterungen nicht brauchbar.

Im folgenden wird mit Begriff immer eine Zusammenfassung mehrerer abstrakter Merkmale zu einem inhaltlichen Ganzen bezeichnet. Dabei kommt es gerade nicht zwingend darauf an, daß die Merkmale kumulativ gegeben sind. Vielmehr kann ein Begriff im Extremfall auch so definiert sein, daß mindestens eines von mehreren Merkmalen disjunktiv vorliegt. Ein Begriff kann seinerseits ein Merkmal eines an-

---

[84] Zur Definition von *Begriff* s. *Herberger*/*Simon* S. 244 ff.

[85] *Larenz*, a.a.O. S. 216.

[86] *Koch/Rüßmann*, a.a.O. § 9 1. (S. 67 ff.).

[87] *Koch/Rüßmann*, a.a.O. § 9 2. (S. 73 ff.).

[88] Vgl. FN. 82.

[89] Zum selben Ergebnis kommen *Koch*/*Rüßmann* a.a.O. S. 77, auf deren ausführliche Auseinandersetzung mit dem Thema hier verwiesen wird.

deren Begriffs sein. Begriffe, die aufgrund ihrer offensichtlichen Bedeutung keinerlei tieferer Definition zugänglich sind, werden als **Grundbegriffe** bezeichnet[90].

Geht man bei der Behandlung eines Falls davon aus, daß ein tatsächlicher Sachverhalt mit einem Begriff erfaßt oder nicht erfaßt wird, entspricht das der Aussage, daß ein Teil eines realen Sachverhalts ein juristisch relevantes Tatbestandsmerkmal erfüllt. Mithin wird hiermit ein juristisch relevantes Faktum eines realen Sachverhalts beschrieben. Gelingt es, von allen Bestandteilen einer realen Situation, eine Aussage über die Deckung mit allen denkbaren juristischen Begriffen herbeizuführen, so ist ein Sachverhalt mit seiner gesamten juristischen Relevanz erfaßt. Diese kombinatorisch kaum zu bewerkstelligende Arbeit wird dadurch verringert, daß man lediglich eine Abdeckung mit Begriffen herbeiführt, die zielführend sind. Das Regelnetz wird dann nur aus der Sicht der anspruchsbegründenden Regel gesehen. Die Betrachtung endet zudem an dem Punkt, an dem über alle hierfür relevanten Begriffe eine Aussage vorliegt[91]. Zudem wird mit Hilfe einer Ausfallregel davon ausgegangen, daß sich ein Begriff und ein Sachverhaltselement grundsätzlich nicht dekken. Es muß also nur eine positive Übereinstimmung festgestellt werden[92].

Der Begriff stellt per se einen Anknüpfungspunkt für die Beschreibung von juristisch relevanten Fakten, mithin deren Austausch zwischen mehreren Vorgängen dar.

bb) Begriffsbildung und -beschreibung

Ein Begriff ist also zunächst ein abstraktes Gebilde und unabhängig von seiner sprachlichen Umschreibung. So kann ein und dasselbe Wort einer Sprache gleichzeitig mehrere Begriffe beschreiben. Umgekehrt kann derselbe Begriff mit unterschiedlichen Wörtern dargestellt werden. In unterschiedlichen Sprachen können diese Phänomene unterschiedlich auftreten[93]. Das Wort *Sache* verkörpert zum Beispiel im Bereich des Zivilrechts[94] einen völlig anderen Begriff als dasselbe Wort im Bereich des Strafrechts[95]. Das Wort *Gehalt* kann *Nettogehalt* und *Bruttogehalt* meinen. *Netto* wiederum kann in unterschiedlichen Zusammenhängen unterschiedliche Abzüge beinhalten. Sprachlich wird dies oft mit Zusätzen wie *im Sinne von...* ausgedrückt, wenn die Beschreibung außerhalb eines ohnehin eindeutigen Zusammenhangs verwendet wird[96]. Das Problem wird um so eklatanter, je höher der Interpretationsbedarf eines Begriffs ist. Während identische Worte möglicherweise an unterschiedlichen Stellen des Regelapparates im Kern auch dieselbe Bedeutung haben[97], unterscheiden sie sich möglicherweise aber in ihren Randzonen erheblich[98]. In so einem Fall handelt es sich letztlich um unterschiedliche Begriffe.

---

[90] So *Koch*/*Rüßmann* a.a.O. S. 25 m.w.N.

[91] Vgl. zur Differenz von tatsächlichem Geschehen und (endgültigem) Sachverhalt *Larenz*, a.a.O. S. 279.

[92] Diese Aussage wird im Hinblick auf Beweislastregeln noch zu differenzieren sein.

[93] So differenziert die englische Sprache beispielsweise mit dem Wort *put* nicht zwischen *setzen*, *stellen* und *legen*. Ein Wort mit dieser mehrschichtigen Bedeutung kennt die deutsche Sprache nicht.

[94] Vgl. *Heinrichs*, Palandt, vor § 90.

[95] Vgl. *Dreher*/*Tröndle* § 242 RdNr. 2, § 303 RdNr. 1c ff.

[96] *Herberger*/*Simon* S. 262 ff.

[97] Es ist wohl kaum ein Fall denkbar, in dem ein Auto nicht den Begriff des mit *Sache* beschriebenen Begriffs abdeckt.

[98] Die Methodenlehre spricht von dem **Begriffshof**. Vgl. hierzu mit einem alternativen Drei–Bereiche–Modell und weiteren Nachweisen *Koch*/*Rüßmann* a.a.O. S. 199 ff.

Der Mangel an sprachlicher Eindeutigkeit stellt insbesondere bei der Übertragung von Begriffen auf die Präzision gewohnte EDV ein Problem dar. Diese erwartet regelmäßig, daß identische Bezeichner auch inhaltsgleich sind. Wird also einer Kette von Zeichen eine bestimmte Bedeutung zugewiesen, so behält sie diese Bedeutung in einem definierten Umfeld bei. Umgekehrt geht ein Programm ohne zusätzliche Regel immer davon aus, daß unterschiedliche Bezeichner auch eine unterschiedliche Bedeutung haben. Will man also einen juristischen Begriff als Träger eines Faktums in der EDV einsetzen, so bedarf es zumindest zweier Schritte:

1. Der Begriff muß in seiner Intension eindeutig beschrieben werden.
2. Der Begriff muß einen eindeutigen Bezeichner erhalten.

Die juristische Methodik bietet hier wenig Hilfe. Regeln für die Begriffsbildung und -beschreibung existieren de facto nicht[99]. Resultat hieraus ist eine Fülle von Interpretationsregeln[100]. Sie behandeln zum einen die Begrifflichkeit einer Formulierung in einer legislativen, exekutiven oder judikativen Regel[101]. Dies betrifft insbesondere das Problem, ob eine gewählte Formulierung in unterschiedlichen Regeln denselben oder einen anderen Begriff meint. Praktisch hängt hiervon die Existenz einer Verbindungsachse im Regelnetzwerk ab. Zum anderen wird die Anwendung des Begriffs auf den Wirklichkeitstatbestand beschrieben[102]. Dies betrifft die Anwendung der vernetzten Regeln in der Wirklichkeit: Beschreibt ein Begriff den konkreten Tatbestand oder nicht. Derartige Probleme der Anwendung von Begriffen führen zu einem komplexen Werk von juristischen **Metharegeln**, die sich mit der Regelung der Anwendung von Regeln beschäftigen. Dieses Werk von Metharegeln ist jedoch nicht mehr Gegenstand der Arbeit[103], da sie sich nicht mit Fakten eines Individualfalls beschäftigen, sondern die konkrete Regel selbst zum Regelungsgegenstand und somit zur *Tatsache* machen.

Die Metharegeln sind bei der Anwendung auch eines EDV-gestützten Netzes von Vorgängen äußerst wichtig. Sie versetzen den Ersteller der technischen Umsetzung erst in die Lage, ein entsprechendes Netz aufzubauen. Dem Anwender geben sie Regeln an die Hand, seinen Sachverhalt korrekt einzuordnen. Für die weitere Arbeit an dieser Stelle sind sie jedoch von geringem Nutzen.

cc) Begriffsformen

Bisher wurde nicht weiter darauf eingegangen, in welcher Form Fakten auftreten können. Vielmehr impliziert eine Subsumtion nach logischen Kriterien zunächst, daß ein Faktum dadurch beschrieben wird, daß ein abstraktes Tatbestandsmerkmal vom Sachverhalt abgedeckt wird oder nicht. Das Faktum entspricht dann einem booleschen Wahrheitswert. Diese digitale Ansicht von Wirklichkeit ist jedoch nicht ausreichend, wie im folgenden darzulegen ist[104].

aaa) Objektive Quantifizierung

In einer Vielzahl von Fällen sind bestimmte Merkmale anhand einer vorgegebenen Maßeinheit quantifizierbar. So besteht eine Forderung nicht einfach, sondern sie hat eine bestimmte meist in einer Währung meßbare Höhe. Eine Sache hat einen Wert,

---

[99] Vgl. *Horn*, a.a.O. S. 22.

[100] Zu diesem Dilemma vgl. *Horn*, a.a.O. S. 22 .

[101] *Larenz*, a.a.O. Kapitel 1; *Koch*/*Rüßmann* a.a.O. 2. Teil.

[102] *Larenz*, a.a.O. Kapitel 3; *Koch*/*Rüßmann* a.a.O. 3. Teil.

[103] Vgl. S. 6.

[104] Vgl. zur **Begriffsformenlehre** *Koch*/*Rüßmann* a.a.O. S. 76 f. mit Verweis auf *Stegmüller* a.a.O. Bd. 2 S. 15 ff.

ein Bremsweg eine Länge, ein Mensch hat ein Alter etc. Die Begriffsformenlehre bezeichnet diese Begriffe auch als **quantitative Begriffe**.

Die Quantifizierung wird hier als **objektiv** bezeichnet, da der zugrundeliegende Maßstab objektiv ist. Oftmals wird auch die Interpretation des tatsächlichen Sachverhaltes nach objektiven oder zumindest streng normierten Gesichtspunkten erfolgen können. Bestimmte Größen wie Temperatur, Gewicht oder Ausdehnung lassen sich messen. Andere Werte wie etwa der Preis einer Sache sind de facto festgelegt. Die Ermittlung eines Sachwertes hingegen wird nicht ohne eine subjektive Beurteilung zu leisten sein. In diesen Fällen dienen beispielsweise Tabellen als Regeln zur Objektivierung der Beurteilung[105].

bbb) Subjektive Quantifizierung

Anders als bei einer Einordnung nach objektiven Maßstäben verhält es sich zunächst mit Begriffen, die von einer nicht bezifferbaren Intensität ausgehen. Hierbei handelt es sich um Begriffe, denen eine subjektive Komponente wie Schmerz oder Interesse zugrunde liegt. Eigentlich nicht in diese Kategorie gehören Begriffe, die durch subjektiv skalierbare Merkmale bestimmt werden, selbst jedoch letztlich eindeutig digital zu entscheiden sind. Betrachtet man das Beispiel des **Tieralters** nach § 833 BGB[106], so ist dies selbst jedenfalls kein Beispiel für einen skalierbaren Begriff. Für das Entstehen der Schadenersatzpflicht ist vielmehr digital zu entscheiden, ob der In–Anspruch-Genommene Tierhalter ist oder nicht. Statt dessen unterscheidet sich der Begriff des Tierhalters von einem Begriff in *Larenz*' **engerem Sinne** dadurch, daß er nicht durch eine Äquivalenz mit anderen Merkmalen definierbar ist. Skalierbar sind in diesem Fall also allenfalls einzelne Merkmale des Definiens, nicht aber das Definiendum.

Ein Beispiel für einen subjektiv einzuschätzenden Umstand ist der Begriff des **Mitverschuldens** in § 254 BGB. Hier wirkt sich eine skalierbare Größe unmittelbar auf die Rechtsfolge einer Norm aus. Das Gesetz stellt damit eine Regel zur Verfügung, die ein subjektiv zu beurteilendes Kriterium in einen objektiven Betrag umsetzt.

Im Gegensatz zur objektiven Quantifizierung ist der subjektiven immanent, daß eine individuelle Beurteilungsskala geschaffen werden muß. Geht es lediglich darum, eine Zahl von n Sachverhalten in ihrer Intensität miteinander zu vergleichen[107], so kann diese Skala auf genau n unterschiedliche Zahlenwerte reduziert werden. Auf diese Weise lassen sich alle Sachverhaltsmerkmale im Verhältnis *größer*, *kleiner* oder *gleich* zueinander anordnen. In Fällen wie dem Mitverschulden wird die Skala vornehmlich durch die Möglichkeiten und Folgen der Beurteilung bestimmt werden. Enthält die Skala beispielsweise nur ganze Drittel als Verschuldensanteile, so fällt zwar die Beurteilung leicht, die finanziellen Folgen der Einstufung sind jedoch bei hohen Schäden immens. Wählt man hingegen eine Skala mit Prozent oder gar Promillen, so täuscht man womöglich eine Genauigkeit vor, die sich einer subjektiven Beurteilung entzieht. Vielfach ordnet die Norm selbst ein gewisses Maß an Toleranz an[108].

---

[105] So etwa die Tabelle von *Sanden*/*Danner*/*Küppersbusch* zur KFZ–Nutzungsentschädigung, abgedruckt z.B. in Palandt Anhang zu § 249.

[106] Klassisches Beispiel für den **Typus**. *Larenz*, a.a.O. S. 216; ablehnend *Koch*/*Rüßmann* a.a.O. S. 74 ff.

[107] Das entspricht dem **komparativen Begriff** der Begriffsformenlehre.

[108] So etwa bei Kostenaufhebung oder geringfügigem Teilunterliegen in § 92 ZPO.

Zusammenfassend ist festzustellen, daß sich bei jeder Form von quantitativem Begriff tatsächliche Umstände durch die Umsetzung in Zahlenwerte ausdrücken lassen. Wesentlicher Unterschied ist das Verfahren, wie die Wirklichkeit zu interpretieren ist.

ccc) Digitale Begriffe.

Unter digitalen Begriffen sind solche zu verstehen, bei denen der Begriff durch einen tatsächlichen Sachverhalt entweder abgedeckt wird oder nicht. Im Fall einer digitalen Abbildung wird auch von **klassifikatorischen** Begriffen gesprochen[109]. Bei der Beurteilung der Begriffsform ist m.E. davon auszugehen, welchen Charakter ein Begriff in einer Norm hat. Ist das Regelungsziel einer Norm digital (Anspruch besteht oder besteht nicht) und geht der Normaufbau von einer (booleschen) logischen Struktur aus, so ist auch der Begriff, der ein Merkmal umschreibt klassifikatorisch. Enthält die Norm hingegen eine Abwägungsregel oder eine andere Form von Schwellwert[110] oder ist das Normziel bereits quantitativ, so können auch die einzelnen Merkmale quantitativ sein.

Diese Differenzierung steht letztlich nicht im Widerspruch zu der Behauptung von *Larenz*[111], Tatbestandsbegriffe seien fast immer interpretationsbedürftig. Doch auch *Larenz* wird nicht in Frage stellen, daß das Ziel jeder Interpretation letztlich eine eindeutige Entscheidung ist. Sie steht aber ebensowenig im Widerspruch zur Begriffsformenlehre, reduziert allerdings den komparativen Begriff auf eine Quantifizierung innerhalb einer individuell definierten Wertemenge.

Im folgenden wird davon ausgegangen, daß juristische Regeln über juristische Begriffe vernetzt sind. Diese sind an sich ein geeigneter Träger für einzelne Fakten eines Sachverhaltes. Aufgrund der mangelnden Disziplin und Regelung bei der Begriffsbildung ergeben sich jedoch wenige Ansatzpunkte für eine konkrete EDV-Umsetzung. Die unterschiedlichen Formen von Begriffen lassen sich etwa auf die Grundtypen technischer Variablen reduzieren.

c) Juristische *Tatsachen*, zeitliche Veränderung und Streitbarkeit.

Zwei Faktoren besitzen in der juristischen Betrachtungsweise der *Wirklichkeit* eine größere Bedeutung als in den meisten anderen Disziplinen. Dies ist der **zeitliche Wandel**, dem Tatsachen unterworfen sind sowie die Möglichkeit, Tatsachen **anzuzweifeln**. Letztere ist dem praktischen juristischen Vorgehen quasi inhärent.

aa) Zeitkomponente

Der **zeitliche Wandel von Tatsachen** ist deshalb etwas im juristischen Bereich besonderes, weil es oft gerade die Veränderungen der Wirklichkeit sind, die juristische Konsequenzen auslösen oder durch juristisches Handeln bewirkt werden[112]. So dauert etwa eine Ehe von der Heirat bis zur Scheidung. Heirat und Scheidung sind **Ereignisse**, die durch juristische Handlungen ausgelöst werden. Das **Verheiratetsein** ist ein juristisch relevantes Faktum, welches jedoch nur in einer bestimmten Zeitspanne besteht. Hieran können sich nun Ansprüche knüpfen, die vorher nicht bestanden und nachher jedenfalls so nicht weiter bestehen. Mit dem Eigentumsüber-

---

[109] So jedenfalls sind *Koch*/*Rüßmann* (a.a.O. S. 76) zu interpretieren, die bei einem klassifikatorischen Begriff davon ausgehen, daß er die *Einteilung eines Gegenstandsbereichs in Klassen* erlaubt. Dabei gehen sie wohl von einer Dreiwertigkeit aus, da sie auch einen unklaren Zustand vorsehen.

[110] Technisch wird hier von Trigger gesprochen. Ein Trigger hat unterhalb eines definierten Schwellwertes den Zustand *falsch* und darüber den Zustand *wahr*.

[111] A.a.O. S. 195.

[112] Eine Suche nach den Schlagworten *Zeit oder Zeitraum oder Zeitpunkt* im BGB führt zu 312 qualifizierten Dokumenten. Die besondere Bedeutung des Zeitfaktors wird hier augenfällig.

gang einer Sache ändert sich die Beziehung zwischen beiden beteiligten Personen und der Sache. Einer verliert seine Eigentümerstellung, der andere wird Eigentümer. Aber auch wesentlich eindeutigere Phänomene wie etwa die Frage des **Menschseins** unterliegen dem Wandel der Zeit. Dabei kann es eine **rechtliche** und/oder eine **tatsächliche** Frage sein, ab wann das Menschsein beginnt und endet.

Ein einfaches Beispiel für die rechtlichen Folgen selbst eines geringen Zeitfaktors ist die Erbfolge. Sterben zwei Eheleute M und F bei einem Autounfall gleichzeitig, beerben sie sich nicht gegenseitig[113]. Die Verwandten erben das jeweils vollständige Vermögen. Stirbt hingegen eine Ehegatte nachweisbar früher, so beerbt dieser zunächst anteilig seinen Ehepartner (§ 1931 BGB). Diesen Anteil vererbt er letztlich an seine Verwandten. Die Verwandten des Erstverstorbenen müssen sich mit einem geringeren Teil begnügen.

Die zeitliche Abhängigkeit der Gültigkeit von Fakten ist offensichtlich. Eine Aussage über ein Faktum kann nur für einen definierten Zeitraum gelten.

bb) Subjektive Komponente

Eine weitere Besonderheit in der juristischen Betrachtung von Fakten liegt darin, daß prinzipiell **kein Faktum mit Gewißheit** angenommen werden kann. Dabei handelt es sich nicht etwa um eine Frage statistischer Gewißheit[114]. Vielmehr muß im streitigen Prozeß damit gerechnet werden, daß jedes Faktum, welches von einer Partei vorgetragen wird, von der anderen bestritten wird. Hier geht es einerseits darum, daß der rechtliche Weg für verfehlt gehalten wird, der zu einem bestimmten Schluß geführt hat. Andererseits wird in der Mehrzahl der Fälle das rein tatsächliche Faktum teilweise ohne besseres Wissen bestritten.

Die Grenzen dieser Unsicherheit ergeben sich zunächst normativ aus § 138 ZPO. Er zwingt die Parteien in ihrem Vorbringen zur Wahrheit und Vollständigkeit. Dabei geht er von einer subjektiven Wahrheitspflicht aus[115]. Die Partei wird dabei nicht gezwungen, zu schweigen, falls sie eine Tatsache nicht mit letzter Sicherheit kennt[116]. Sie kann auch Tatsachenbehauptungen des Gegners selbst mit Nichtwissen bestreiten. Zudem hängen Sanktionen wegen eines Verstoßes[117] gegen die Wahrheitspflicht wiederum von den Aussagen der Parteien und ihrer Beweisbarkeit ab. Insgesamt führt dies zu einer erheblichen Freiheit im Parteivortrag und somit zu stark divergenten Darstellungen.

Zur Ermittlung des entscheidungserheblichen Sachverhalts dient ein Instrumentarium an Beweismitteln,[118] Beweislastregeln[119] sowie die eigene Expertise oder Lebenserfahrung des Entscheidenden. Die Beweislastverteilung erfaßt insbesondere die Fälle der Beweisarmut. Hier greifen Beweislastregeln, nach denen zu entscheiden ist, welche Behauptungen der Rechtsentscheidung letztendlich zugrunde zu legen sind. Die einzelnen Regelwerke bedürfen hier keiner weiteren Erörterung[120]. Schließlich geht es hier um den Austausch von festgestellten oder zumindest be-

---

[113] *Edenhofer*, Palandt § 1931 RdNr. 5.

[114] Vgl. zur Feststellung des Sachverhalts *Koch/Rüßman*, a.a.O. 3. Teil.

[115] *Thomas*/*Putzo*, § 138 1.b).

[116] BGH, WPM 85, 736.

[117] *Thomas*/*Putzo*, § 138 3.

[118] §§ 355 ff. ZPO.

[119] Vgl. hierzu *Thomas*/*Putzo*, 7. Vor § 284 m.w.N.

[120] Vgl. S. 6.

haupteten Fakten. Fragen der Sachverhalts– oder Wahrheitsfindung sind diesem Austausch entweder vorgelagert oder sie folgen ihm nach.

Geht man davon aus, daß nicht Tatsachen, sondern Aussagen über Tatsachen Gegenstand der Subsumtion sind[121], so kann dies zumindest für ein laufendes Verfahren dahingehend erweitert werden, daß unterschiedliche Aussagen zum selben Sachverhaltsmerkmal vorliegen können. Selbst bei einem rechtskräftigen Urteil eines Verfahrensweges kann in einem anderen Verfahrensweg für denselben realen Tatbestand ein anderer Sachverhalt zugrundegelegt werden. Die unterschiedlichen Aussagen kommen im Parteiprozeß meist von den unterschiedlichen Parteien. Im Strafprozeß kann der strafrechtliche Grundsatz **in dubio pro reo** dazu zwingen, bei demselben realen Sachverhalt im selben Verfahren unterschiedliche Sachverhalte anzunehmen[122]. In diesem Fall ergehen die Aussagen letztlich aufgrund einer differenzierten Interpretation desselben Sachverhaltes aufgrund unterschiedlicher zugrundeliegender **Rechtsbegriffe**.

Derselbe Lebenssachverhalt kann sich also einerseits durch unterschiedliche Aussagen unterschiedlicher Beteiligter bezüglich desselben Rechtsbegriffs unterschiedlich darstellen. Vor einer rechtliche Würdigung muß in diesem Fall eine Entscheidung getroffen werden, welche Aussagen als Grundlage der Entscheidung herangezogen werden. Betrachtet man den Rechtsbegriff als Anknüpfungspunkt für Fakten, so muß hier eine Erkennungsmöglichkeit vorgesehen werden, wessen Darstellung des Merkmals gemeint ist.

Der reale Sachverhalt kann jedoch auch ohne entgegengesetzte Behauptungen an unterschiedlichen Begriffen zu beurteilen sein. Er stellt sich dann an unterschiedlichen Stellen einer rechtlichen Würdigung differenziert dar. In diesem Fall müssen jedoch die scheinbar divergierenden Aussagen beide gleichzeitig zur Urteilsfindung herangezogen werden. In diesem Fall müssen zwei Rechtsbegriffe für den Austausch eines de facto identischen Sachverhaltsmerkmals herangezogen werden.

cc) Mischformen von Unsicherheiten

Hinzu kommt, daß sich Zeitfaktor und Unsicherheit überlagern können und dies im Regelfall auch tun. Dies passiert, wenn sich die einzelnen Parteivorträge im Laufe der Zeit ändern, insbesondere, wenn sich Behauptungen aufgrund der Beweislage nicht mehr halten lassen oder wechselseitig neue Tatsachen vorgebracht werden. Die Partei kann vormals streitige Tatsachen zugestehen oder Verteidigungsmittel vorbringen. Die gesetzlichen Regelungen hierzu[123] weisen nur recht weite Grenzen und sind nicht dazu gedacht und geeignet, die Parteien dauerhaft an eine einmal getätigte Aussage zu binden.

Hieraus ergibt sich eine zweite Form der zeitlichen Änderung von Fakten, nämlich die Änderung des Parteivortrags innerhalb der Verfahrensgeschichte. Fakten können sich also nicht nur rein tatsächlich ändern, sondern die Behauptung eines Faktums kann im Laufe der Zeit verworfen bzw. erst aufgestellt werden. Ein weiterer Schritt der Verschachtelung ergibt sich dann, wenn gerade der Zeitpunkt, in dem ein Faktum entstanden ist, im Streit liegt. So kann es von existentieller Bedeutung sein, ob bei einem Autounfall der Ehemann vor oder nach der Ehefrau gestorben ist. Es kann streitig sein, ob es nach der Trennung noch zu ehelichem Verkehr kam. Dabei sind es nicht unbedingt Fragen des numerischen Datums, die letztendlich im Streit ste-

---

[121] *Larenz*, a.a.O. S. 273.

[122] So etwa bei einer nicht exakt bestimmbaren Tatzeit–BAK; vgl. *Dreher*/*Tröndle* § 20 RdNr. 9f.

[123] Vgl. etwa § 296 ff. ZPO.

hen, sondern die zeitliche Abfolge mehrerer Ereignisse. So ist es im ersten Beispiel an sich unwichtig, an welchem Datum und zu welcher Uhrzeit der Tod der Ehegatten eintrat. Lediglich von Interesse ist, wer als Erster gestorben ist.

Endlich kann es auch noch darüber zum Streit kommen, bis zu welchem Zeitpunkt eine Partei eine bestimmte Behauptung aufgestellt hat. Dies mag bei einer Beurteilung der Kosten oder der Notwendigkeit einer Beweisaufnahme[124] oder eines Rechtsmittels relevant sein[125].

Dieser gesamten Dynamik muß auch beim Austausch von Fakten Rechnung getragen werden. So muß es möglich sein, Fakten daraufhin zu filtern, von welcher Person sie behauptet wurden. Ebenso muß erkennbar sein, wie der aktuelle Stand der Behauptungen ist. Besonders zu Argumentationszwecken muß aber auch jeder beliebige frühere Tatsachenvortrag einzelner Parteien abrufbar sein. Im einzelnen sind folgende zusätzliche Informationen zur Verarbeitung von streitigen Fakten von Interesse

- Zeitraum der Gültigkeit von Fakten
- Partei, die die Tatsache vorträgt
- Zeitpunkt des Tatsachenvortrags
- Beweismittel für den Tatsachenvortrag

Das zu entwickelnde System sollte Werkzeuge zur Verfügung stellen, derartige und womöglich noch weitere Zusatzinformationen zugänglich zu machen.

Das deutsche Rechtssystem arbeitet mit einem Netz von ineinander greifenden Regeln. Die Achsen des Netzes bilden Begriffe. Sie eignen sich prinzipiell auch als *Transportmittel* von Fakten. Ein ganz erhebliches Merkmal der juristischen Behandlung von Fakten liegt in der zeitlichen Veränderung der Wirklichkeit und in der subjektiven Darstellung durch Verfahrensbeteiligte. Die Abbildung dieser Variationen gehört zu den Grundbedingungen für einen praxistauglichen Austausch von Fakten.

3. Ansätze zur Algorithmisierung des Rechts

Vielfach ist bereits versucht worden, das Recht **berechenbarer** zu machen. Diese Versuche gehen nicht zwangsläufig einher mit dem Einsatz von Computern. Wohl aber hat die EDV versucht, sich die Erkenntnisse dieser Überlegungen zunutze zu machen. Da es sich hierbei jedenfalls um Ansätze einer präzisen Beschreibung von Entscheidungsvorgängen handelt, die dem Denken der Informationstechnik entgegenkommen, sollen die Ergebnisse hier kurz resümiert werden.

a) Logik

Die Logik geht bereits weit vor die Zeit einer technischen Implementation mittels EDV zurück. Sie wird an dieser Stelle bewußt von Ansätzen distanziert, die ihren Ursprung in z.T. modischen Strömungen der Informatik oder Naturwissenschaft haben.

aa) Stellenwert der Logik

Juristische Logik beschäftigt sich mit der Übertragung juristischer Entscheidungsmechanismen in formale logische Strukturen[126]. Die Anwendung logischer Kalküle auf juristische Regeln soll diese nicht nur einer maschinellen Auswertung zugänglich machen, sie soll vielmehr als Werkzeug für eine Präzisierung juristischer Arbeitsweisen dienen[127].

---

[124] Z.B. § 96 ZPO.

[125] Z.B. § 97 ZPO.

[126] *Herberger*/*Simon* S. 17 bezeichnen die Logik als *unentbehrliches Hilfsmittel* für den Juristen.

[127] Vgl. *Schneider*, a.a.O. S. 9 f.

Logik ist ein wesentliches Werkzeug (formaler) juristischer Argumentation bzw. Begründung. Sie ist weitaus älter als das zentrale Thema dieser Arbeit, die EDV[128]. Ihre Entwicklung erfolgte zunächst unabhängig von technischen Zwängen. So bezeichnen etwa *Koch* und *Rüßmann*[129] die Deduktion als ein zentrales Postulat juristischer Begründungen. Ebenso werden beispielsweise bei *Bund*[130] Juristische Logik und Begründungslehre als zwei aufeinander aufbauende Gebiete behandelt. *Alexy*[131] bezeichnet die Deduktion als *best method to secure the binding force of positive law ... deduction leads to completeness of justification ....* Dabei zielt die überwiegende Zahl der Veröffentlichungen jedenfalls nicht primär auf eine automatisierte Verwertung dieser Arbeitsergebnisse. Vielmehr wird die Logik zunächst abstrahiert von technischen Zwängen analysiert. Sie dient hier als generelles Mittel, juristische Begründungen auf ihre Lückenlosigkeit zu überprüfen bzw. korrekte Begründungssequenzen zu ermöglichen[132].

Auch die Erkenntnisse der Logik werden im Bereich dieser Arbeit immer unter dem Aspekt des Datenflusses darzustellen sein. Es ist, wie schon vielfach betont, hier nicht das Ziel Regelwerke auch technischer Natur zu erarbeiten, die juristische Entscheidungen treffen können. Vielmehr soll mit dem Ergebnis dieser Arbeit diesen Regelwerken (und sekundären Prozessen) eine Möglichkeit an die Hand gegeben werden, ermittelte Fakten eines konkreten Falls untereinander auszutauschen. Die Behandlung von Fakten in den einzelnen logischen Systemen gilt es hier aufzuzeigen. Die Brauchbarkeit der zugrundeliegenden Algorithmen und Kalküle soll dagegen nur dann betrachtet werden, wenn sich aus ihnen ergibt, daß der Ansatz offensichtlich unbrauchbar oder juristisch objektiv uninteressant ist. Dagegen können Fragen der technischen Realisierbarkeit eines Ansatzes genauso wenig Gegenstand der Arbeit sein, wie ethische oder moralische Fragen bezüglich des Einsatzgebietes von Entscheidungsautomatismen im juristischen Bereich[133].

Als Beispielskonstellation soll folgender einfacher Fall gelten:

F1: Der mittellose ARNO ist das Kind der ebenfalls mittellosen BERTA sowie des CLEMENS (Monatsgehalt DM 4.000). BERTA und CLEMENS sind verheiratet, leben aber getrennt.

Zur Deckung eines angemessenen Lebensunterhaltes werden DM 3.000 monatlich benötigt.

bb) Aussagenlogik

Die Aussagenlogik stellt im Prinzip die Grundform der Logik dar. Sie kennt lediglich Aussagen und logische Konjunktoren als syntaktische Elemente.

⇒ *Unter Aussage versteht man dabei jeden sinnvollen Satz, dessen Inhalt wahr oder falsch sein kann*[134].

---

[128] Die klassische Logik wird auf Aristoteles zurückgeführt. *Schneider*, a.a.O. S. 10.

[129] *Koch/Rüßman* .

[130] *Bund, Elmar* Juristische Logik und Argumentation, S. 9.

[131] A.a.O. S. 69.

[132] *Schneider* a.a.O. S. 10.

[133] Vgl. etwa *Bund*, S. 286 f.

[134] *Herberger*/*Simon* S. 34 gehen davon aus, daß eine Aussage *einzelnen oder mehreren Gegenständen eine Eigenschaft zu oder abspricht.* M.E. ist diese Definition sehr eng gefaßt. Sie reduzieren auch Aussagen über Beziehungen auf die Vergabe von Eigenschaften. Der Satz *A und B haben einen Vertrag V geschlossen* ist zweifellos eine Aussage. Dennoch weder A noch B explizit eine **Eigenschaft** zugewiesen.

Eine Aussage nach dieser Definition ist etwa der Satz: *Berta ist die Mutter von Arno*; oder auch *Berta ist Arno zum Unterhalt verpflichtet.* Unschädlich ist dabei, daß der zweite Satz die Aussage über eine Zahlungsverpflichtung, also eine Handlungsanweisung der BERTA enthält. (Die deontische Logik etwa würde hier einen Unterschied machen[135].) Typischerweise werden in der Literatur insbesondere zur juristischen Logik Aussagen durch Variablen in Kleinbuchstaben aus dem Bereich *p*, *q* etc. verwendet.

Mit Hilfe von **Junktoren** lassen sich Ausdrücke bilden, die mehrere Aussagevariablen miteinander verknüpfen. Diese Junktoren werden in **Wertetabellen** definiert. Für Junktoren mit zwei Stellen ergeben sich dabei insgesamt 2^4 mögliche Definitionen. Hinzu kommt die Definition für einen einstelligen Junktor, die Negation[136],[137]. Wesentliche Anwendungstechnik für logisch transformierte Regeln ist der korrekte logische Schluß. Als Beispiel soll hier der am häufigsten verwendete Schlußmechanismus, der **modus ponendo ponens** kurz gezeigt werden.

$$\frac{p \quad \rightarrow \quad q \qquad p}{q}$$

Dabei ist *p* der Tatbestand etwa einer Norm (in der Terminologie der Logik **Antecedens**) und *q* deren Rechtsfolge (auch **Konsequens**). Diese weithin übliche Darstellung entspricht der Formel $\big((p \rightarrow q) \wedge p\big) \Rightarrow q$ [138]. $p \rightarrow q$ wird dabei als **Vordersatz** bzw. **Obersatz** oder auch **Prämisse** bezeichnet. *p* ist der **Hintersatz** bzw. **Untersatz** und *q* der **Schlußsatz**. Gemeint ist: Unter der Annahme daß $p \rightarrow q$ gilt und der weiteren Annahme, daß *p* vorliegt, so liegt auch *q* vor. Unter der Prämisse $p \rightarrow q$ läßt sich also von *p* auf *q* **schließen**. Die Allgemeingültigkeit dieses Satzes kann z.B. durch die folgende Wertetafel bewiesen werden.

((	*p*	→	*q*	)	∧	*p*	)	⇒	*q*
	w	w	w		w	w		**w**	w
	w	f	f		f	w		**w**	f
	f	w	w		f	f		**w**	w
	f	w	f		f	f		**w**	f

Auf die einzelnen logischen Junktoren und die aus ihnen zulässigen Schlüsse wird in dieser Arbeit nicht weiter eingegangen. Hierzu sei auf die z.T. sehr detaillierten Standardwerke verwiesen[139]. Dagegen soll die Einbringung von Fakten in derartige Formeln näher betrachtet werden. Der hier dargestellte **modus ponendo ponens** wurde bereits als Syllogismus eingeführt[140]. Dort wurde jedoch von einer natürlichsprachlichen Darstellung ausgegangen. Es sind also die Variablen *p* und *q* durch natürlichsprachliche Variablen zu ersetzen. Sie müssen geeignet sein, den Bereich

---

[135] S.u. S. 48.

[136] Eine Tabelle mit den wesentlichen logischen Junktoren findet sich im Anhang der Arbeit, S. 242.

[137] Eine Zusammenstellung der unterschiedlichen Symbole für Junktoren findet sich bei *Herberger*/*Simon*, S. 9. In dieser Arbeit wird für die Junktoren die Notation von *Herberger*/*Simon* für die verwendet. Für die Quantoren der Prädikatenlogik wird allerdings auf die Notation nach *Scholz* zurückgegriffen.

[138] Die Zeichen „→“ und „⇒“ unterscheiden sich nicht in ihrer logischen Bedeutung. „⇒“ symbolisiert lediglich den eigentlichen Schluß.

[139] Z.B. *Herberger*/*Simon*, 3.7. (S. 54 ff.). *Scheider*, S. 95 ff.

[140] S. FN. **Fehler! Textmarke nicht definiert.** (S. **Fehler! Textmarke nicht definiert.**).

von Aussagen zu umschreiben, für den die formulierte Regel gelten soll. Ersetzt man die Bezeichner *p* und *q* etwa zur Darstellung des § 1601 BGB durch die hierin verwendeten Begriffe, so ergibt sich der folgende Schluß:

geradlinige Verwandtschaft → *Unterhaltspflicht*
geradlinige Verwandtschaft

Unterhaltspflicht

Die Norm wird somit formal in Tatbestand und Rechtsfolge aufgespalten. Der Tatbestand wird durch den Rechtsbegriff *geradlinige Verwandtschaft* repräsentiert, die Rechtsfolge durch den Begriff *Unterhaltspflicht*. Beide Begriffe sind noch keine Aussagen im Sinne der Aussagenlogik, da sie weder wahr noch falsch sein können. Sie umschreiben lediglich einen Bereich von Aussagen aus der Wirklichkeit, für deren Einsetzung die Regel gültig ist. Diese Einsetzung von Aussagen aus der Wirklichkeit (meist als **Welt** bezeichnet) ist die **Interpretation** der Variablen. Im Beispielsfall (sozusagen der Beispiels**welt**) läßt sich die Variable *geradlinige Verwandtschaft* durch die Aussage *Arno und Berta sind geradlinig verwandt* ersetzen. Hieraus ergibt sich folgender Schluß

geradlinige Verwandtschaft → *Unterhaltspflicht*
ARNO und BERTA sind geradlinig verwandt

Unterhaltspflicht

Durch eine weitere Interpretation läßt sich auch die Variable *Unterhaltspflicht* durch eine konkrete Aussage substituieren:

geradlinige Verwandtschaft → *Unterhaltspflicht*
ARNO und BERTA sind geradlinig verwandt

ARNO und BERTA sind einander unterhaltspflichtig

Es soll nun geprüft werden, inwieweit diese Erkenntnisse zur Übermittlung von Fakten zwischen Vorgängen hilfreich sind. Hierzu soll zunächst eine weitere Norm (§ 1589 S. 1 BGB) betrachtet werden. Dabei wird von der Annahme ausgegangen, daß die Anwendung jeder Regel einen eigenen Vorgang darstellt. Die Vorschrift bestimme, wann geradlinige Verwandtschaft vorliegt. Auch hier soll eine stark schematische Formalisierung zunächst ausreichen:

Abstammung → *geradlinige Verwandtschaft*

Bindeglied zwischen dieser Regel und der vorangegangenen Regel ist die Variable *geradlinige Verwandtschaft*. Sie entspricht in ihrer Bedeutung einem zugehörigen juristischen Begriff. Allerdings taucht sie in § 1601 BGB als Antecedens der Implikation, in § 1589 BGB hingegen als Konsequens auf. Das macht es möglich, unter der Voraussetzung, daß beide Variablen dieselbe Bedeutung haben, die zugrundeliegenden Begriffe also identisch sind, eine Schlußkette von der *Abstammung* zur *Unterhaltspflicht* aufzubauen:

Abstammung → *geradlinige Verwandtschaft*
ARNO und BERTA sind geradlinig verwandt

ARNO und BERTA sind einander unterhaltspflichtig

geradlinige Verwandtschaft → *Unterhaltspflicht*
ARNO und BERTA sind geradlinig verwandt

ARNO und BERTA sind einander unterhaltspflichtig

Durch die Identität der Variablen kann das Ergebnis des ersten Schlusses als Prämisse des zweiten Schlusses gewonnen werden. Es ergibt sich ein **Kettenschluß** der Form $(p \rightarrow q) \wedge (q \rightarrow r) \Rightarrow (p \rightarrow r)$. Die Aussage *Arno und Berta sind geradlinig verwandt* kann nun zusammen mit ihrem Wahrheitswert, d.h. mit der Feststel-

lung, daß die Aussage wahr oder falsch ist, als Faktum betrachtet werden. Dieser Kettenschluß stellt also einen Mechanismus dar, ein Faktum von einer aussagenlogischen Regel in eine andere zu übertragen. Dieser Mechanismus beruht vor allem auf der Verwendung identischer Variablen. Variablen dürfen wiederum nur dann identisch sein, wenn die zugrundeliegenden Rechtsbegriffe identisch sind. Kurz gesagt, Intension und Extension äquivalent sein.

So bietet die Aussagenlogik zunächst einen Ansatz zur Repräsentation von Fakten als Aussagen. Sie unterliegt jedoch erheblichen Beschränkungen. Das Modell betrachtet einen realen Sachverhalt als eine Menge von Fakten, die durch Aussagen und deren Wahrheitswerte repräsentiert sind. Aussagen werden aufgrund einer Menge von juristischen Begriffen formuliert, die sich letztlich aus den zugrundeliegenden Regeln ergeben. Die Anwendung der logischen Regeln besteht darin, die jeweiligen Variablen durch diese Aussagen auszufüllen und so einen Schluß auf eine weitere Aussage abzuleiten. Das Problem der Aussagenlogik ergibt sich aus der Beschränkung, daß jede Variable jeweils nur mit einer Aussage belegt werden kann. Dies verbietet es beispielsweise, die Variable *Abstammung* gleichzeitig mit den Aussagen *Arno stammt von Berta ab* und *Arno stammt von Clemens ab* zu belegen.

Eine Regel, bei der zwei verschiedene Personen eine identische Eigenschaft haben müssen, ist in der Aussagenlogik nicht ökonomisch formulierbar. Die Regel $p \wedge p \rightarrow r$ ist unsinnig, da wegen $p \leftrightarrow p$ die Variable p nicht zwei verschiedene Aussagen repräsentieren kann. Die Regel $p \wedge q \rightarrow r$ ist unökonomisch, da p und q im Grunde identische Intensionen repräsentieren, ihr unterschiedlicher Bezeichner jedoch auf eine Differenzierung schließen läßt. Käme es also etwa bei Inanspruchnahme beider Eltern darauf an, beide Fakten zu repräsentieren, so wäre dies für das beschriebene System nicht möglich. Der **Begriff** *Abstammung* ist nämlich mit einem Faktum bereits besetzt. Eine weitere Beschränkung liegt darin, daß die (boolesche) Aussagenlogik lediglich digitale Daten (eine Aussage ist wahr oder falsch) verarbeiten kann.

Ein System für den Austausch von Fakten kann deshalb nicht auf der Aussagenlogik aufbauen. Andererseits erfreut sich die Aussagenlogik auch im Bereich von EDV-technischen juristischen Systemen einer erheblichen Beliebtheit[141]. Das System muß also eine Lösung für die Einbindung von auf der Aussagenlogik basierenden Anwendungen bieten.

cc) Prädikatenlogik

Die Prädikatenlogik behebt gewisse Einschränkungen der Aussagenlogik. Sie bietet eine wesentlich komplexere Möglichkeit der Darstellung von Sachverhalten als die Aussagenlogik, indem sie Aussagen als Eigenschaften von einem Individuum oder Beziehungen zwischen mehreren Individuen formuliert. Die Grundzüge werden hier ebenfalls beschrieben[142].

Eine Aussage p der Aussagenlogik wird in der Prädikatenlogik etwa als $P(X)$ dargestellt. Das sogenannte Prädikat[143] P repräsentiert dabei weiterhin die juristische Intension. Hinzukommt das **Individuum** X, auf das P zutrifft. Werden mehrere

---

[141] Ein relativ neues Beispiel ist die Arbeit von *Möller*. Andere Systeme vor allem aus dem universitären Bereich wie *Chung*, **Terminus** oder *Kraft*, **Sophos** basieren ebenfalls auf der Aussagenlogik.

[142] Ausführliche Beschreibungen der Prädikatenlogik finden sich bei *Herberger*/*Simon*, 4.2. (S. 90 ff.) sowie bei *Koch*/*Rüßmann*, § 5 Nr. 3 (S. 39 ff.; sie wird hier als **Quantorenlogik** bezeichnet).

[143] S. zur Bezeichnung *Herberger*/*Simon* S. 90.

Individuen in eine Beziehung zueinander gesetzt, so lautet die Aussage beispielsweise $P(X, Y, Z)$.

aaa) Individuen

Individuen sind Objekte, die in der Wirklichkeit existieren. Sie müssen nicht zwingend gegenständlich sein. Als Objekte in diesem Sinne können vielmehr auch Werte oder Ereignisse aufgefaßt werden[144].

Individuen werden durch **Individuenvariablen** vertreten. Diese werden i.d.R. mit kleinen Buchstaben vom Ende des Alphabetes dargestellt (x, y, z). Reale Individuen werden mit entsprechenden Großbuchstaben geschrieben oder durch sprechende Namen wie hier etwa ANTON, BERTA oder CLEMENS bezeichnet.

bbb) Prädikate

Prädikate bilden die Beziehung der Individuen untereinander. Sie entsprechen der inhaltlichen Bedeutung der Aussage. Dargestellt werden Prädikate durch entsprechende Ausdrücke wie *Vater* oder *Kind*. Abgekürzt werden sie meist mit den entsprechenden Anfangsbuchstaben. Zur Unterscheidung von Individuen verwendet man möglichst nur Buchstaben aus der Mitte des Alphabets.

Prädikate werden differenziert in **Funktionen** und **Relationen**. Funktionen sind geeignet, eine bestimmte Kombination von Individuen (ein sogenanntes n–Tupel) mit **genau einem** weiteren Individuum in Relation zu setzen. Ein Prädikat mit der Bedeutung *ist Vater von* wird etwa jedem Menschen genau einen Vater zuweisen. Hierfür verwendet man oftmals eine eigene Schreibweise: $y = Vater(x)$. Die Individuenvariablen x und y werden mit Elementen aus definierten Mengen belegt, wobei sich der Wertebereich von y aus der Definition von *Vater* ergibt. Man sagt eine **Funktion bildet** eine oder mehrere Mengen auf eine weitere Menge **ab**. Entnimmt man im Beispiel x aus der Menge aller Menschen, so bildet die Funktion *Vater* diese Menge auf die Menge aller männlichen Menschen ab, die mindestens ein Kind haben.

Eine Relation unterliegt nicht der Beschränkung einer eindeutigen Abbildung. Ein Prädikat $y = Kind(x)$ mit der Bedeutung *ist Kind von* kann ein x auf mehrere Kinder y abbilden. Umgekehrt gibt es immer zwei Individuen y für jedes Kind x. Man verwendet dann die übliche Prädikatenschreibweise $K(y, x)$. Die Unterscheidung zwischen Funktion und Relation ergibt sich letztlich aus der Definition des jeweiligen Prädikats. Die Anzahl der Individuenvariablen einer Relation wird als **Stelligkeit** bezeichnet. Relationen mit nur einer Stelle nennt man auch **Eigenschaften**.

In der juristischen Logik werden Funktionen nur wenig Bedeutung zugemessen, da sich jede Funktion prinzipiell auch als Relation darstellen läßt. So kann die oben genannte Funktion $y=Vater(x)$ auch als Relation $Vater(y, x)$ definiert werden, wobei die Individuenvariable y dann als weiteres Glied in die Klammer gezogen wird. Diese Umformung erleichtert es, die gewohnten logischen Junktoren auch auf Prädikate anzuwenden.

Die Darstellung von Prädikaten ging bisher davon aus, daß ein Prädikat P bereits eine definierte Bedeutung besitzt. Will man logische Sätze formulieren, die für mehrere Funktionen oder Relationen gelten, werden sogenannte **Funktions-** bzw. **Relationsvariablen** verwendet. Beispielsweise gilt das Kommutativgesetz $Verwandt(x, y) \leftrightarrow Verwandt(y, x)$ für eine ganze Reihe anderer Prädikate ebenfalls. Eine solche Aussage läßt sich leicht durch die Formel $\forall f (f \in \{Verwandt,$

[144] *Herberger*/*Simon* S. 90.

Verschwägert} $\Rightarrow$ ($f(x, y) \leftrightarrow f(y, x)$))[145] verallgemeinern[146]. Es wird also eine allgemeine Regel formuliert, die für Prädikate mit unterschiedlicher Bedeutung Geltung hat.

Wie bei Individuenvariablen werden auch Prädikatenvariablen mit kleingeschriebenen Wörtern dargestellt oder mit Kleinbuchstaben abgekürzt.

ccc) Interpretation

Sowohl Prädikate als auch Individuen können dementsprechend in Ausdrücken durch Variablen vertreten werden. Will man einen variablen Ausdruck $f(x, y)$ für einen konkreten Sachverhalt in eine Aussage umsetzen, so müssen die einzelnen Variablen **interpretiert** werden.

Prädikatenvariablen stehen als Platzhalter für Sinnzusammenhänge. Sie beschreiben die Eigenschaften von einzelnen oder Relationen von mehreren Objekten. Die Bedeutung eines Prädikats entspricht der Intension eines juristischen Begriffs. Die Menge aller Prädikate ist entsprechend aus einer hypothetischen Menge **aller** juristischen Begriffe zu entnehmen. Der Begriff **geradlinige Verwandtschaft** ließe sich etwa als Relation *Geradlinige Verwandtschaft*(x, y) darstellen. Typischerweise wird man allerdings bei juristischen Regeln nicht auf Prädikatenvariablen stoßen, so daß dieser Schritt entfällt.

Individuenvariablen werden durch Konstanten aus der tatsächlichen Welt, d.h. durch Personen, Sachen aber auch Verträge, Ereignisse usw. substituiert. Durch die vollständige Interpretation aller Variablen durch Elemente der entsprechenden Mengen entsteht eine Aussage. Die Aussage *Arno und Berta sind geradlinig verwandt* würde in Prädikatendarstellung als *Geradlinige Verwandtschaft*$(Arno, Berta)$ dargestellt werden.

Wie bei Aussagen der Aussagenlogik sind Aussagen mittels Prädikaten und konkreten Individuen *wahr* oder *falsch*. Hierzu ist zu überprüfen, ob die entsprechende Beziehung oder Eigenschaft der Realität entspricht oder nicht. Es muß also ein Abgleich aller Beziehungen der betroffenen Individuen mit der Intension des Prädikats erfolgen. Dabei ist von besonderer Wichtigkeit, daß die Rollen der Individuen in der Beziehung gewürdigt werden. Das erwähnte Kommutativgesetz, das eine beliebige Vertauschung der Attribute erlaubt, gilt nur bei wenigen Prädikaten.

ddd) Quantifizierung

Auf Prädikate lassen sich prinzipiell dieselben logischen Junktoren anwenden, die bereits für die Aussagenlogik definiert wurden. Die eigentliche logische Erweiterung sind die sogenannten **Quantoren**[147]. Sie lassen Aussagen darüber zu, welche Individuen in eine Variable eingesetzt werden können. Ausgehend von einer definierten Menge gibt der **Allquantor** $\forall$ an, daß ein Ausdruck für alle Elemente gelten soll. Der **Existenzquantor** $\exists$ bedeutet, daß ein Ausdruck für mindestens ein Individuum gilt. Der so gebildete Ausdruck$x\,(Mensch(x) \rightarrow y\,(Vater(y,\ x)))$ bedeutet beispielsweise im Klartext *jeder Mensch hat einen Vater*. Genauer gesagt, für alle Individuen gilt, wenn ein Individuum ein Mensch ist, dann gibt es ein Individuum, das sein Vater ist. Dabei schließt diese Regel nicht einmal aus, daß beide Individuen auch identisch sein können. Dieselbe Behauptung läßt sich auch dadurch

---

[145] Lies: Immer wenn *f* die Bedeutung *Verwandt* oder die Bedeutung *Verschwägert* hat, gilt $f(x, y) \leftrightarrow f(y, x)$. Zur Bedeutung von Quantoren wie „$\forall$“ s.S. 43.

[146] Der Einsatz von Prädikatenvariablen in logischen Ausdrücken führt letztlich zur Prädikatenlogik der sogenannten **zweiten Stufe**. Auf diese soll hier nicht näher eingegangen werden.

[147] Deshalb benutzen *Koch*/*Rüßmann* (S. 39) auch den Ausdruck „Quantorenlogik“.

formulieren, daß die Quantifizierung auf eine Menge von Individuen begrenzt wird: $\forall x \in M \exists y \in M (Vater(y, x))$, M=Menge aller Menschen [148].

Enthält die Menge, über die quantifiziert wird, d.h. aus der Elemente in die Variablen einsetzbar sind, lediglich Individuen, so spricht man von **Prädikatenlogik erster Stufe**. Kann dagegen mit einem Prädikat auch eine Aussage über eine Menge weiterer Prädikate getroffen werden, so führt dies zur **Prädikatenlogik zweiter Stufe**. Das ist beispielsweise bei der Anwendung von Grundrechten auf Normen einfachen Rechts denkbar. Auf die mit der Prädikatenlogik zweiter Stufe verbundene Problematik soll hier nicht näher eingegangen werden. Da Regeln nicht zum Gegenstand des hier behandelten Austauschs von Fakten gemacht werden sollen[149], spielt auch eine mögliche Quantifizierung von Regeln in der folgenden Betrachtung keine Rolle.

Die Quantifizierung ist ein wesentlicher Teil des prädikatenlogischen Kalküls. Sie ist ein Hauptelement der formulierten Regeln. Hiermit werden Konstellationen beschrieben, die sich am einfachsten mit der Problematik der Tatbeteiligung im Strafrecht vergleichen lassen. Hier gibt es Tatbestandsmerkmale, die von allen Beteiligten erfüllt sein müssen, andere Merkmale müssen lediglich von (mindestens) einem Täter erfüllt werden. Fragen der Quantifizierung gehören zur Regelbildung. Sie sind deshalb im Rahmen dieser Arbeit von geringerem Interesse. Auf eine genaue Beschreibung des prädikatenlogischen Kalküls wird deshalb im folgenden verzichtet[150].

eee) Identität und Kennzeichnung

Identität ist eine Relation, die feststellt, daß zwei Variablen genau dasselbe Individuum bezeichnen. Anstelle der konventionellen Prädikatenschreibweise *Identität*(x, y) wird hierfür eine eigene Syntax, das Gleichheitszeichen, verwendet: $x = y$. Die Feststellung der Identität unterschiedlich beschriebener Individuen ist ein zentraler Punkt des Austauschs von Fakten. Betrachtet man Eigenschaften und Beziehungen von Individuen untereinander als Fakten, so kann ein Faktum lediglich dann korrekt beschrieben werden, wenn die Individuen identifizierbar sind.

Eine wesentliche Form der Identifizierung von Individuen ist die **Kennzeichnung**[151]. Sie erfolgt immer dann, wenn es Eigenschaften oder Beziehungen gibt, die eine eindeutige Bestimmung eines Individuums zulassen. In der Formel $\forall x \exists y (x, y \in M \wedge A(x) \rightarrow x = y)$ ist beispielsweise $A(x)$ Kennzeichnung für y. D.h. jedes x, das die Eigenschaft A hat, ist identisch mit y. Dabei gibt es für y nur ein Element in der Menge **M**. Eine Kennzeichnung wird durch ein **Jota** dargestellt mit der Syntax $\iota x A(x)$. Unter der Bedingung, daß lediglich ein Element der Individuenmenge die Eigenschaft A haben kann, kann die Kennzeichnung ι anstelle einer Konstanten auftreten. So kann beispielsweise anstelle der Konstanten BERTA der Ausdruck *Mutter von Arno* stehen: $Berta = \iota x\, Mutter(x, Arno)$. Hingegen wäre eine Relation $Kind(x, y)$ als Kennzeichnung ungeeignet, da jede Mutter mehrere Kinder haben kann. Eine eindeutige Zuordnung ist somit nicht möglich. Bereits der Sprachgebrauch zeigt diesen Unterschied. So wird der Ausdruck *Arnos Mutter* sicherlich als

---

[148] Der Ausdruck unterscheidet sich in einer Nuance, da er auch die Information beinhaltet, daß der Vater ebenfalls ein Mensch ist; $\forall x (Mensch(x) \rightarrow \exists y (Vater(y, x) \wedge Mensch(y)))$.

[149] Zur Einschränkung des Arbeitsfeldes s.S. 6.

[150] Eine sehr detaillierte Beschreibung findet sich bei *Herberger*/*Simon*, S. 93 ff.

[151] Ausführlich *Herberger*/*Simon*, 4.8.3. (S. 159 ff.).

eindeutig empfunden. Hingegen wird man *Bertas Kind* nur dann sagen, wenn BERTA lediglich ein Kind hat[152].

Die Kennzeichnung entspricht prinzipiell der Definition von **Funktionen**[153]. Sie stellt ein sehr brauchbares Mittel zur Verfügung, unbekannte Personen aufgrund juristisch relevanter Kriterien zu identifizieren. Sie bildet damit sicherlich, wie noch in den weiteren Ausführungen zu zeigen sein wird, eine gute Basis für den Austausch von Fakten zwischen inhomogenen Vorgängen.

fff) Bildung von Prädikaten

Ähnlich wie es im juristischen Bereich an einer Zunft der Formulierung von Begriffen mangelt, mangelt es in der juristischen Logik an Ausführungen zur Bildung von Prädikaten. Lediglich en passant wird die Frage nach der Notation gestreift.

So kursieren die unterschiedlichsten **syntaktischen Varianten** der Prädikatendarstellung, ohne daß sich hieraus ein tatsächlicher Unterschied herleiten ließe. Beispielsweise wird ein dreistelliges Prädikat in der mathematisch orientierten Literatur oft wie auch hier als $R(x, y, z)$, teilweise mit Tiefstellung als Rxyz[154] und in der als *Relationenlogik* bezeichneten Form als xRyz dargestellt. Eine besondere Darstellung bietet die **Sortenlogik** durch die Angabe einer Teilmenge des Universums für die einzelnen Variablen. Die Bezeichnung $R(A{:}x,\ B{:}y)$[155] entspricht dabei dem Ausdruck $A(x) \wedge B(y) \rightarrow R(x, y)$ oder mit einer Einbeziehung von Mengen dem Ausdruck $x \in \mathbf{A}$, $y \in \mathbf{B}$: $R(x, y)$. Die Notation der Sortenlogik erhöht die Lesbarkeit der Prädikate. Die hierdurch bedingte Einschränkung des Gültigkeitsbereiches erleichtert zudem die logische Behandlung derartiger Ausdrücke. Sie bringt jedoch gleichzeitig logische Regeln in die Prädikate ein und widerspricht so dem hier verfolgten Ansatz. Sie soll deshalb letztlich außer Betracht bleiben.

Zentrale Frage der Prädikatenbildung ist einerseits die Formulierung der **Prädikatenkonstanten** und andererseits die Auswahl und Behandlung der **Objektvariablen**. Die Umschreibung der tatsächlichen Bedeutung erfolgt unter Verwendung juristischer Begriffe. Ein Prädikat ist also inhaltsgleich mit einem juristischen Begriff. Die Beschreibung der Intension eines Prädikats erfolgt analog zu der eines Begriffs[156].

In der Prädikatenlogik verkörpert jedoch der Begriff ein Szenario mit der Beteiligung mehrerer Individuen. Entsprechend müssen die beteiligten Individuen in Form von Objektvariablen definiert werden. In der gängigen Literatur wird dieser Frage eine sehr untergeordnete Bedeutung zugewiesen. Bei der Beispielsbildung wird sie oft eher intuitiv als analytisch gelöst. Bei der Erstellung komplexerer Systeme ist jedoch ein derart unbewußtes Vorgehen nicht geeignet, die notwendige Einheitlichkeit zu erreichen. Die Problematik sei am Beispiel der Ehe kurz erläutert:

Einfachste Form der Abbildung ist das Prädikat $Ehe(m, f)$. Das Prädikat formuliert, daß zwischen den Variablen m und f die Relation *Ehe* besteht. Die Ehe selbst ist dabei kein Objekt, sondern eine abstrakte Beziehung. Dies führt unter anderem zu Schwierigkeiten, wenn weitere Aussagen über die Ehe selbst zu machen sind. Betrachtet man beispielsweise die Heiratsregisternummer so erscheint die Darstellung $Registernummer(m, f)$, wobei m und f wiederum die Eheleute sind, offensichtlich

---

[152] S.a. *Herberger*/*Simon*, S. 159 f.

[153] S.o. bbb) (S. 42).

[154] So bei *Herberger*/*Simon* und *Koch*/*Rüßmann*.

[155] Vgl. *Maier, A.*, a.a.O. S. 40.

[156] S. S. 31.

verfehlt, da *m* und *f* durch die Nummer nicht in irgendeine Beziehung zueinander gesetzt werden. Vielmehr ist diese Nummer eine Eigenschaft der Ehe als eigenständiges Objekt. Hierfür wäre die Darstellung x = *Registernummer*(*Ehe*(*m, f*)) wesentlich praxisnäher. In dieser Variante stellt das Prädikat *Ehe* quasi eine Funktion dar, die jeweils zwei Elemente des Universums *m* und *f* auf ein weiteres Element abbildet. Auch *Registernummer* ist hiernach eine Funktion, die die Ehe auf die zugehörige Registernummer abbildet.

Die Darstellung ist in dieser Form jedoch nicht eindeutig. Aus der Definition der Ehe ergibt sich, daß zwei Personen auch mehrmals eine Ehe eingehen können. Letztendlich handelt es sich also nicht um eine (eindeutige) Funktion, sondern um eine (mehrdeutige) Relation. Diese wird bisher dargestellt als *Ehe*(*e, m, f*). Die Ehe *e* stellt hiernach ein eigenes Element des Universums dar, welches zu *m* und *f* die definierte Relation *e* hat. Hieraus ergibt sich die einsichtigere Formulierung *Registernummer*(*x, e*) Auch die Registernummer wird dabei als eigenständiges Objekt durch die Variable *x* repräsentiert. Die Infixdarstellung in der **Relationenlogik** betont diese **Substantivierung** der Prädikate, indem sie eine **Hauptvariable** der Relationenkonstante/-variable voranstellt. Die Ehe ist nach dieser Notation als *e Ehe m, f* abzubilden. Obwohl diese Darstellung m.E. deutlicher ist, wird für die folgende Abhandlung weiterhin die verbreitetere Präfixsyntax verwendet.

Gegen die Bildung einer Hauptvariablen spricht vor allem, daß diese einer möglichen **Symmetrie** von Variablen eines Prädikats widerspricht. Eine Symmetrie liegt etwa dann vor, wenn zwei Variablen prinzipiell jederzeit vertauschbar sind. Das Prädikat *Verheiratet*(v_1, v_2) ist ein Beispiel für die Gleichwertigkeit zweier Variablen. Die Reihenfolge ist hier rein zufällig. Ein ähnliches Phänomen ergibt sich nicht nur bei zwei Variablen, sondern auch bei einer beliebig langen **Liste** etwa in einem Prädikat *Verwandt*(v_1, v_2,...v_n). Beide Fälle schließen dem Anschein nach in der gewählten Darstellung die Bildung einer Hauptvariablen aus, da alle Variablen dieselbe Wertigkeit und Stellung besitzen.

Die Feststellung der Symmetrie ist allerdings nicht in jedem Fall zwingend. Vielmehr handelt es sich bereits um die Anwendung einer **Symmetrieregel**, dem sogenannten Kommutativgesetz *Verheiratet*(*m, f*) $\leftrightarrow$ *Verheiratet*(*f, m*). Im Gegensatz zu einfach gelagerten Fällen wie diesem Beispiel, sind Prädikate denkbar, in denen die Symmetrie nicht ohne Zweifel angenommen werden kann. Vielmehr kann die Symmetrie abhängig sein von der Definition des entsprechenden Begriffs. Definiert man etwa *Feind*(*x, y*) aus dem Prädikat *Haßt*(*x, y*), so mag man dies definieren können als *Haßt*(*x, y*) $\longrightarrow$ *Feind*(*x, y*). Nachdem Haß eine einseitige Angelegenheit sein kann, wäre nach dieser Regel auch die Feindschaft nicht zwingend symmetrisch. Engt man die Voraussetzungen ein (*Haßt*(*x, y*) $\wedge$ *Haßt*(*y, x*) $\longrightarrow$ *Feind*(*x, y*)) oder erweitert man sie (*Haßt*(*x, y*) $\vee$ *Haßt*(*y, x*) $\longrightarrow$ *Feind*(*x, y*)), so wird das Prädikat *Feind* (*x, y*) hingegen symmetrisch.

Bildet man nun die Symmetrie als besondere **syntaktische** Variante (*Feind* ([v_1, v_2])) ab, so läuft man Gefahr, daß derartige Zweifelsfälle falsch dargestellt werden. Das Prädikat wird dann nämlich mit womöglich fehlerhaftem Regelwissen belastet. Oder aber es werden nicht die vollen syntaktischen Möglichkeiten genutzt, wenn eine Symmetrie fälschlich nicht dargestellt wird. Eine Information, die jedoch nicht gleichmäßig syntaktisch repräsentiert ist, ist letztendlich nur wenig wertvoll.

Diese Fehler lassen sich umgehen, indem man auf die syntaktische Auflösung der Symmetrie ganz verzichtet und sie in den entsprechenden Regeln implementiert. Das Prädikat *Verwandt* läßt sich so etwa durch ein **un**symmetrisches Prädikat *Verwandter*(*v, p*) ersetzen. Hier wurde eine Variable künstlich zur Hauptvariablen gemacht. Die Formulierung des Prädikats erfolgt nun nicht mehr durch ein Adjektiv,

sondern durch ein Substantiv, welches die Hauptvariable beschreibt. Das Verfahren ermöglicht eine Darstellung der Prädikate unabhängig von Regeln (insbesondere für Symmetrie), wie sie in dieser Arbeit generell angestrebt wird.

Neben der **Substantiierung** der Prädikate in einer obligatorischen Hauptvariablen, stellt sich die weitere Frage, **wie komplex** die Variablenliste sein muß. Eine Forderung mag etwa sein, daß der Zeitfaktor als Variable der Relation aufzunehmen ist. Die Ehe wäre dann allgemein als $Ehe(e, m, f, t)$ abzubilden. Weitergehend ist ein Ansatz, nach dem auch alle diejenigen Variablen in ein Prädikat aufzunehmen sind, die in den Prädikaten der zugrundeliegenden juristischen Definition enthalten sind. Dies soll anhand eines abstrakten Beispiels erläutert werden:

Angenommen die Ehe käme zustande durch zwei Willenserklärungen der Eheleute: Zunächst wäre die Willenserklärung als Prädikat darzustellen: $W(w, p, i)$, wobei w die Erklärung selbst, p der Erklärende und i der Erklärungsinhalt wäre. Die Tatsache, daß zwei Personen eine Erklärung desselben Inhalts *Ehe* abgeben, kann als $W(w_1, p_1, i_1) \wedge W(w_2, p_2, i_2) \wedge (i_1=e) \wedge (i_2=e)$ dargestellt werden. Zu diskutieren wäre, ob nun alle Variablen (w_1, p_1, i_1, w_2, p_2, i_1, und e) auch als Variablen des Prädikats *Ehe* abzubilden wären: $\forall e \forall w_1 \forall p_1 \forall i_1 \forall w_2 \forall p_2 \forall i_2 \ (W(w_1, p_1, i_1) \wedge W(w_2, p_2, i_2) \wedge (i_1=e) \wedge (i_2=e)) \Rightarrow Ehe(e, w_1, p_1, i_1, w_2, p_2, i_2))$. Diese Darstellung ist noch erheblich vereinfacht. (Die Ehe muß vor einem Standesbeamten geschlossen sein, bedarf einer Form etc.) Die Liste ist entsprechend weiter zu vergrößern. Bezieht man noch die möglichen weiteren Gründe ein, die einer Ehe entgegenstehen (Doppelehe, Verwandtschaft etc.), so würden auch Variablen etwa für die potentielle zweite Ehefrau in die Liste aufzunehmen sein.

Das Beispiel zeigt, daß diese Form der Prädikatenbildung unrealistisch ist. Sie ist auch nicht notwendig: Prädikate dienen zur Darstellung einer Menge von Aussagen, die über ein definiertes Universum getroffen werden. Diese Aussagenmenge eines n-stelligen Prädikats P wird als eine Menge von n-Tuppeln über das Universum abgebildet. Ein Element der Aussagenmenge ist also eine Gruppe von jeweils n Objekten des Universums, die die Relation P verbindet. So ließe sich aus dem Universum $\{E_1, E_2, A, B, C, D\}$ die Menge der Trippel $\{(E_1, A, B), (E_2, C, D)\}$ bilden, die beispielsweise die Beziehung *Ehe* verbindet. Es spricht jedoch auch nichts dagegen, daß in einem Universum eine Menge $\{(E_1, A, B), (E_2, C, D)\}$ existiert, in der A und B zweimal verheiratet, C und D hingegen Singles sind. Eine Aussage, d.h. ein n-Tuppel ist dann durch eine Regel verifizierbar, wenn es eindeutig ist. Die Aussage $Ehe(A, B)$ ist nicht eindeutig, da sie in einem bestimmten Zeitabschnitt wahr, in einem anderen hingegen falsch sein kann. Hingegen ist die Aussage $Ehe(A, B, T)$, wobei T der Zeitpunkt ist, immer eindeutig zu verifizieren, da zwei Personen nicht gleichzeitig miteinander verheiratet und nicht miteinander verheiratet sein können. Entsprechendes gilt für die Aussage $Ehe(E_1, A, B)$. Auch hier ist eine Zeitkomponente notwendig, um den Ausdruck verifizierbar zu machen. Zwar gibt es im deutschen Recht keine Möglichkeit, daß ein Element E_1 beispielsweise von einem einfachen Vertrag zu einem Ehevertrag mutieren kann. Die Relation ist also nach deutschem Recht eindeutig verifizierbar. In einer (beliebigen) Rechtsordnung ist aber ein Fall denkbar, in dem E_1 zunächst als eine $Gesellschaft(E_1, A, B)$ zu betrachten ist und erst etwa durch das Nachholen eines Formerfordernisses zu einer Ehe wird. In diesem Fall wäre wiederum eine Zeitkomponente zur Verifizierung notwendig. Mit der Formulierung $Ehe(E_1, A, B, T)$ ist die zeitliche Abhängigkeit gelöst.

Ist eine Aussage verifizierbar, so läßt sich auch eine (wenn auch noch so kasuistische) Regel entwickeln, die entscheidet welche aller möglichen Aussagen innerhalb eines Universums wahr und welche falsch sind. Zur Darstellung aller möglichen

Aussagen, die überhaupt durch ein Prädikat $Ehe(e, m, f, t)$ gemacht werden können, dient die Allquantifizierung als Regelvoraussetzung: $\forall e \forall m \forall f \forall t$.

Eine Aussage $Ehe(E_1, A, B, T)$ kann dabei von weiteren Faktoren abhängen, die nicht als Variablen eingeführt sind. Das Verbot einer Doppelehe etwa verbietet die Existenz eines weiteren Elements im Universum, welches eins der Eheleute zu einem beliebigen anderen Element gleichzeitig in die Relation Ehe setzt. Dieses Element kann innerhalb der Regel eingeführt werden:

$$\forall e \forall m \forall f \forall t \forall x (Ehe(e, m, f, t) \rightarrow \neg \exists y (y \neq e \wedge (Ehe(y, m, x, t) \vee Ehe(y, x, f, t)))).$$

Für jede Ehe *e* gilt also, daß kein zweites ($y \neq e$) Element *y* des Universums existiert, welches *m* oder *f* mit einem beliebigen Element *x* zur gleichen Zeit *t* in das Verhältnis *Ehe* setzt. Obwohl mit dem Verbot der Doppelehe eine notwendige Bedingung für das Vorliegen einer Ehe formuliert wird, ist es wie das Beispiel verdeutlicht nicht notwendig und letztlich auch kaum praktisch die Variablen *x* und *y* in die Attributleiste des Prädikats *Ehe* aufzunehmen.

Die für die Bildung eines Prädikats notwendigen Variablen sind also nicht zwingend abhängig von denjenigen, die zu seiner Definition benötigt werden. Es können demnach syntaktisch betrachtet beliebige Prädikate gebildet werden. Fraglich ist lediglich, ob die Prädikate, d.h. die mit ihnen formulierbare Aussage fachlich (juristisch) einen Sinn ergibt. So ist auch das Prädikat $Ehe(x, t)$ denkbar, wenn es so formuliert wird, daß *x* zur Zeit *t* mit einem beliebigen Element des Universums verheiratet ist. Der Sinn dieses Prädikats bestimmt sich nach juristischen Gesichtspunkten.

Bei der Prädikatenbildung ist deshalb darauf zu achten, daß ein Prädikat selten eine isolierte Existenz führt. Insbesondere bei der Verwendung von Prädikaten als Datenträgern ist darauf zu achten, daß diese nur insoweit brauchbar sind, als die durch sie verkörperte Information anderweitig im System brauchbar ist. Hierzu müssen Kriterien erarbeitet werden, die eine möglichst abstrakte Bildung von Prädikaten möglich machen.

Für die weiteren Darstellungen wird wie folgt verfahren:

- *Prädikate werden substantiviert. D.h. es wird immer eine Hauptvariable eingeführt, die die durch das Prädikat beschriebene Eigenschaft/Relation verkörpert.*
- *Es werden nur Attribute eingeführt, die die wesentlichen an dem verkörperten Rechtsbegriff beteiligten Objekte vertreten.*
- *Das Zeitattribut wird generell weggelassen. Es ist implizit in jedem Prädikat enthalten und wird deshalb syntaktisch gesondert behandelt.*

dd) Deontische Logik

Die deontische Logik (auch normative Logik oder Normlogik genannt) versucht, dem Umstand Rechnung zu tragen, daß gerade die durch Normen ausgesprochenen Rechtsfolgen oftmals keine Tatsachen darstellen, sondern meist eine Handlung gebieten, verbieten oder erlauben[157]. Zur adäquaten Abbildung dieser Besonderheit werden neue Operatoren eingeführt:

- $O(x)$: *x* ist geboten
- $P(x)$: *x* ist erlaubt
- $F(x)$: *x* ist verboten
- $I(x)$: *x* ist freigestellt

[157] Eine ausführliche Einführung findet sich in der Einführung zu *Weinberger* (S. 1 ff.) sowie in den folgenden Beiträgen.

x steht dabei für ein Prädikat, das eine Handlung bezeichnet. Die Operatoren werden nicht einheitlich bezeichnet. *Koch*/*Rüssmann* verwenden beispielsweise die Anfangsbuchstaben der deutschen Begriffe V (verboten), E (erlaubt). Ebenso verzichten sie auf den Operator *freigestellt*. Anderorts wird der Operator $I(x)$ auch für den **Imperativ** also das **Sollen** verwendet[158].

Die **Unterhaltspflicht** beispielsweise wird im Sinne der deontischen Logik mit den eingangs beschriebenen Operatoren als $O(\textit{Unterhalt leisten}(x,\ y))$ dargestellt. Durch die Einführung dieser Operatoren soll es nun möglich sein, Abhängigkeiten der einzelnen Gebotsformen und damit der möglichen Rechtsfolgen einer Norm aufzuzeigen. Diese Beziehungen werden beispielsweise als $F(x) \leftrightarrow O(\neg x)$ **definiert**. Ebenso läßt sich zwischen Erlaubnis und Gebot eine allerdings nicht äquivalente Beziehung definieren. Hierzu wieder das Beispiel Unterhaltspflicht: $O(\textit{Unterhalt leisten}(\mathrm{x},\ \mathrm{y})) \rightarrow P(\textit{Unterhalt leisten}(\mathrm{x},\ \mathrm{y}))$. Von der Leistungspflicht läßt sich auf die Erlaubnis, von der Erlaubnis jedoch nicht auf die Pflicht schließen.

Nutzen und Existenzberechtigung der deontischen Logik sind umstritten.[159] Da nicht ausgeschlossen werden kann, daß sich bei ihrer Berücksichtigung Konsequenzen für die Darstellung von Fakten ergeben können, soll kurz auf die Problematik eingegangen werden. Die Vertreter der Normlogik sehen ihre Notwendigkeit in dem Umstand, daß die Rechtsfolge einer Norm keine Aussage darstellt, sondern eine Handlung gebietet, verbietet, erlaubt etc. Daraus resultiere, daß eine Rechtsfolge nicht **wahr** oder **falsch** sein könne, sondern eben einen gestaltenden bzw. anordnenden Charakter besäße. Diese Folge sei nun durch eine geeignete Repräsentation auf syntaktischer Ebene zu berücksichtigen. Diese Wirkung ließe sich m.E. auch mit Hilfe eines neuen Wertetyps erzielen, der auf die Werte aus der Menge der Anweisungen **M** = {*müssen, sollen, dürfen, verboten sein*} begrenzt wäre. Für Variablen dieses Typs gilt dann eine entsprechend zu definierende Algebra. Hierdurch würde eine neue Form der Tatsachenrepräsentation geschaffen, deren Transport von einem System, wie es hier zu entwickeln ist, zu berücksichtigen wäre.

Auch die deontische Logik erlaubt das Formulieren von Schlüssen, die von einer **Aussage** auf eine **Rechtsfolge** zielen. So bedeutet etwa $p \longrightarrow O(x)$, daß unter dem Umstand *p* die Handlung *x* geboten ist. Auf der linken Seite des Satzes steht eine Aussage, die lediglich wahr oder falsch sein kann. Die Implikation mittels des Operators $\rightarrow$ bildet diese Wahrheitswerte seinerseits per Definition wiederum auf Wahrheitswerte ab. Somit kann auch der Ausdruck $O(x)$ nur wahr oder falsch sein. Der genannte Ausdruck scheint also gegenüber dem klassischen Prädikatenkalkül kaum Originalität aufzuweisen. Vielmehr ist der Operator $O(x)$ ein Prädikat mit eben der Intension *geboten sein*. Hiernach wird mit dem Ausspruch der Rechtsfolge implizit die **Aussage** getroffen, daß eine bestimmte Handlung geboten oder erlaubt ist. Diese Aussage kann bezogen auf die zugrundeliegenden Regeln durchaus wahr oder falsch sein.

Ebenso lassen sich bei der Behandlung der Operatoren als klassische Prädikate ihre Beziehungen untereinander gleichermaßen darstellen, wie sie auch in der Normlogik definiert werden[160]. Diese Vereinfachung reduziert die Frage nach dem Sinn der Normlogik darauf, ob die Einführung der Prädikate $\textit{Geboten}(x)$, $\textit{Verboten}(x)$ etc. aus juristischer Sicht notwendig ist und welche logischen Beziehungen zwischen diesen Prädikaten existieren. Betrachtet werden soll als Beispiel die Frage, ob die

---

[158] S. *Weinberger* S. 39 f.

[159] Zur Auseinandersetzung um die Normlogik s. *Herberger*/*Simon* 5.1. (S. 179 ff.).

[160] *Koch/Rüssmann* bezeichnen dieses Phänomen als **Normsatzlogik** (S. 45).

Implikation $O(\neg x) \rightarrow \neg O(x)$ allgemeingültig ist[161]. Aus Sicht der Prädikatenlogik ist der Ausdruck äquivalent zu $\neg(O(\neg x) \wedge O(x))$. O kann also hiernach nicht gleichzeitig auf *x* und ¬*x* zutreffen. Ob dies letztlich gilt, ist eine Frage der Interpretation des Begriffs *geboten sein*. Formal logisch ausgeschlossen ist dies jedoch zunächst nicht.

Letztlich handelt es sich bei den deontischen Operatoren um Prädikate mit einer vorgegebenen Intension, die sich insbesondere in definierten Beziehungen der Prädikate untereinander äußert. Die hierfür nötige Begriffsbildung spielt sich dabei auf einem anderen Abstraktionsniveau ab als die Verwendung juristischer Begriffe selbst. Es sind vielmehr Metharegeln, die allen eine Handlung gebietenden, verbietenden oder erlaubenden Normen vorgelagert sind. Entsprechend führen Regeln über die Beziehungen der deontischen *Prädikate*, wie etwa $\forall x(O(x) \rightarrow P(x))$ in die Prädikatenlogik zweiter Stufe. Sie quantifizieren über eine Menge von Prädikaten.

Ob es Situationen gibt, in denen die deontische Logik Probleme zu lösen vermag, die die klassische Prädikatenlogik nicht lösen kann, soll hier letztlich nicht entschieden werden. M.E. lassen sich die relevanten Rechtssachverhalte über die Begriffsbildung womöglich sogar differenzierter erfassen. Der Begriff der **Herausgabepflicht** etwa wird im Sinne der deontischen Logik als $O(\text{Herausgeben}(e, b, s))$ und in der Prädikatenlogik als $\text{Herausgabepflicht}(e, b, s)$ bezeichnet werden. Beide Ausdrücke sind, bezogen auf konkrete Individuen, Aussagen, deren Wahrheitswert sich an dem zugrundeliegenden Rechtssystem orientiert. Sollte nun die Frage nach einem **Recht** zur Herausgabe rechtlich relevant sein, so spricht praktisch nichts dagegen, hieraus einen eigenen Begriff $\text{Herausgabeerlaubnis}(e, b, s)$ zu bilden und **bei Bedarf** die Regel $\text{Herausgabepflicht}(e, b, s) \rightarrow \text{Herausgabeerlaubnis}(e, b, s)$ ausdrücklich zu formulieren. Geht man hingegen mit der deontischen Logik von $\forall x\,(O(x) \rightarrow P(x))$ also auch $O(\text{Herausgeben}(e, b, s)) \rightarrow P(\text{Herausgeben}(e, b, s))$ aus, so setzt dies ein ideales Rechtssystem, frei von rechtlichen Widersprüchen voraus[162]. Die Forderung nach einem solchen Rechtssystem entspricht letztlich einer weiteren Metharegel, die besagt, daß alle Regeln untereinander widerspruchsfrei sein müssen. Eine solche Regel ist jedoch in der realen Welt nicht selbstverständlich und mitnichten zwingend.

Beschränkt man sich auf das Ziel der Darstellungen in dieser Arbeit, so kann man davon ausgehen, daß es über die Prädikatenlogik hinaus keiner weiteren zu beachtenden Darstellungsmethode bedarf. Vielmehr sind Informationen, wie sie die Normenlogik verarbeitet, durchaus als Prädikate darstellbar und als solche auch transportierbar. Insbesondere angesichts der geringen Festigung der Begrifflichkeit innerhalb der deontischen Logik erscheint es auch wenig sinnvoll, für die eine oder andere Begriffsbildung Partei zu ergreifen. Dagegen ermöglicht es gerade die Offenheit der allgemeinen Prädikatendarstellung, unterschiedliche Strömungen dieses Bereiches der Logik abzubilden.

b) Analoge Deduktionsmethoden

Nicht unmittelbar Gegenstand der klassischen juristischen Logik sind die Formen der Entscheidungsfindung, die hier als *analoge* Deduktionsmethoden bezeichnet werden sollen. Gemeint sind Algorithmen und Verfahren, die es erlauben aus **nicht digitalen** Ausgangszuständen ein digitales oder ebenfalls analoges Ergebnis zu liefern. Derartige analoge Eingaben sind in folgenden Fällen anzunehmen:

---

[161] *Koch/Rüssmann*, S. 46.

[162] Ausführlicher *Koch/Rüssmann*, S. 47.

– Das Vorliegen eines Merkmals kann nicht mit absoluter Sicherheit angenommen werden.
– Ein Merkmal kann in unterschiedlichen Intensitäten auftreten.

Ebenso mitbeachtet werden sollen Verfahren, die zwar digitale Eingaben verarbeiten und digitale Ergebnisse liefern, sich jedoch intern nicht boolescher Verfahren bedienen. Nicht hierher gehören klassische mathematische Berechnungen, die in einer Norm fixiert sind[163].

Die Algorithmen variieren in unterschiedlichen Ansätzen. In einer einfachen Form können unterschiedliche Kriterien nach intellektuell zu vergebenden Punktesystemen gewichtet werden. Weiterhin können Kalküle der Wahrscheinlichkeitsrechnung herangezogen werden[164]. In einer Vielzahl der Fälle sind die Entscheidungsmechanismen allerdings so komplex, daß sie lediglich vom Computer selbst anhand von exemplarischen Situationen und Ergebnissen erstellt oder justiert werden können. Eine **Lernkomponente** ist dann ein wesentlicher Bestandteil solch komplexer Ansätze. Die Entscheidungsfindung ist für den Anwender nur noch nachvollziehbar, wenn das System ebenfalls über eine **Erklärungskomponente** verfügt und diese allgemeinverständliche, juristisch relevante Aussagen liefert.

Solche Systeme, unabhängig von der Komplexität, sehen sich immer einer Kritik mangelnder praktischer Relevanz ausgesetzt: Dienen sie der Entscheidung über das Vorliegen oder Nichtvorliegen von Merkmalen, so ist dies eine Tatsachenentscheidung des Richters, die er aufgrund der Beweislage als Ergebnis der mündlichen Verhandlung zu treffen hat. Insofern bleibt für die Anwendung von Wahrscheinlichkeitsmethoden jedenfalls für juristische Anwendungen wenig Raum[165]. Etwas anderes gilt lediglich, wenn sich Sachverhaltsfeststellungen auf **Erfahrungssätze** oder auf **Prognosen** stützen müssen[166]. In diesen Fällen ist eine statistische Analyse der Sachverhaltserfassung immanent. Eine weitere Ausnahme mag eine Wahrscheinlichkeitsabschätzung mit dem Ziel eines gerechten Vergleichs sein. Für den Anwalt ist die praktische Brauchbarkeit ebenso fraglich. Ihm ist in Zweifelsfällen mit einer quasi genauen Wahrscheinlichkeitsentscheidung eines Algorithmus oft überhaupt nicht geholfen. Er muß vielmehr alle denkbaren **Varianten durchdenken**, um für alle Möglichkeiten gewappnet zu sein. Lediglich im Fall der Knappheit von Zeit oder anderen Ressourcen wird er sich darauf verlassen müssen, daß irgendein Ereignis mit einer gewissen Wahrscheinlichkeit eintritt oder eben nicht.

Für die Gewichtung nicht digitaler Eingaben, etwa der Intensität bestimmter Merkmale (Höhe der Schmerzen, Grad der Schuld etc.) schreibt das Gesetz allerdings in vielen Fällen eine nicht scharfe Deduktion vor. Gemeint sind hiermit besonders Fragen des Ermessens oder der Wertung. Sie **triggern** eine Liste analoger Werte mit dem Ziel, eine alternative Entscheidung herbeizuführen. Für diese Fragen liegen meistens keine eindeutige Formeln vor. Moderne Formen der künstlichen Intelligenz und der Logik entwickeln deshalb fortwährend neue Methoden zur Beherrschung selbst stark kasuistischer Entscheidungsmechanismen. Hierunter fallen insbesondere

---

[163] S. nächsten Abschnitt (S. 52).

[164] Über erfahrungswissenschaftliche Methoden im Recht: *Herberger*/*Simons*, Kapitel 14; *Koch*/*Rüßmann* §§ 30 ff.

[165] Im medizinischen Bereich beispielsweise erscheint der Einsatz statistischer Systeme weitaus praxisrelevanter.

[166] Auf die Einzelheiten der Wahrscheinlichkeitsrechnung soll hier gar nicht eingegangen werden. Hierzu *Herberger*/*Simon*, S. 349 ff.; Sehr ausführlich *Koch*/*Rüßmann*, §§ 30 ff. (S. 287 ff.).

Entwicklungen aus den Gebieten der neuronalen Netze, der Fuzzy-Logik oder der Chaostheorie.

In einer Gesamtbetrachtung lassen sich algebraische wie durch Erfahrung gesteuerte Methoden als Relationen zwischen Individuen aus nicht–digitalen Menge darstellen. Ergibt sich eine Rechtsfolge *r* aus einer Gewichtung der Merkmale *x*, *y* und *z*, so besteht zwischen Ihnen die Relation $G(x, y, z) \longrightarrow r$. Die Intension des Begriffs *G* wird in dem, womöglich durch ein rein iteratives Verfahren entwickelten, Algorithmus definiert. Die vorangegangenen Ausführungen zur Prädikatenlogik und zur juristischen Begriffsbildung zeigen, daß diese Probleme und Erkenntnisse keine Erweiterung der bisher beschriebenen Methoden verlangen. Für die begriffliche Behandlung nicht-digitaler Merkmale wurde bereits darauf verwiesen, daß die Einführung von unterschiedlichen Begriffsformen juristisch notwendig ist. Mit der Einführung von Begriffsformen wurde es ermöglicht, die Intensität von Merkmalen oder eine analoge Wertigkeit begrifflich zu fassen[167].

Ebensowenig ist die Prädikatenlogik nicht auf boolesche Werte beschränkt. So kann ein Prädikat beliebige Beziehungen auch analoger Werte beschreiben. Die Prädikatendarstellung unterliegt auch keiner Beschränkung auf klassische algebraische Formeln. Indes kann in einem Prädikat $p(x, y)$ die Variable *p* auch aus der Menge nichtlinearer Beziehungen entnommen sein. Insofern lassen sich auch diejenigen Beziehungen in Form von Prädikaten darstellen, die etwa durch neuronale Netze oder andere Methoden algorithmisiert werden.

c) Formeln im Recht

Der Vollständigkeit halber sei hier erwähnt, daß das Rechtssystem eine ganze Reihe von klassischen algebraischen Formeln enthält, die per se einen Algorithmus beschreiben. Diese lassen sich zwar einfach als Prädikat in das System der Logik einordnen. Sie fallen dennoch aus der üblichen Methode der Rechtsanwendung heraus, die auf der Subsumtion von Tatbestandsmerkmalen beruht.

So liegt § 472 BGB der einfache Dreisatz

$$\textit{geminderter Kaufpreis} = \frac{\textit{tatsächlicher Kaufpreis} \times \textit{Wert der mangelhaften Sache}}{\textit{Wert der Sache in mangelfreiem Zustand}}$$

zugrunde. Die unzähligen Vorschriften des Steuerrechts seien hier nur in Erinnerung gerufen. Letztlich fallen auch tabellarisch abgefaßte Regeln in diesen Bereich. Zum einen sind nämlich viele Tabellen lediglich Ausfluß einer einzelnen oder einer Kombination mathematischer Formeln. Zum anderen, sind in ihnen **objektive** Regeln zur Umsetzung von Zahlenwerten in weitere Zahlenwerte enthalten.

Diese Regeln sind zwar in sich objektiv mit oft einfachen Algorithmen abzubilden[168]. Will man jedoch ihren Wirkungsbereich im Gesamtzusammenhang eines Sachverhaltes sehen, bedarf es wiederum einer komplexeren Beschreibungsmöglichkeit wie der Prädikatenlogik. Die Algorithmen geben keine Auskunft über die Einordnung der Input- oder Outputvariablen im Wirklichkeitszusammenhang.

Von allen Ansätzen, Rechtsregeln zu algorithmisieren, ist der prädikatenlogische Ansatz m.E. der brauchbarste. Zum einen lassen sich alle weiteren Ansätze in Form von Relationen oder Funktionen darstellen, zum anderen ist es der einzige Ansatz, der die Abbildung eines komplexen Bildes der Wirklichkeit erlaubt.

---

[167] S.o. S. 32.

[168] Eindrucksvollstes Beispiel ist hier die Abbildung der Lohnsteuerberechnung durch den Programmablaufplan des Bundesfinanzministers (BStBl. I 1995, 634 ff.).

4. Analyse der praktischen Vorgehensweisen unter Berücksichtigung der prozessualen Vorgaben

Im vorangegangenen Abschnitt wurden im wesentlichen theoretische Fragen der Rechtsfindung angesprochen. Das zu entwerfende System soll jedoch über die Unterstützung der reinen Rechtsfindung hinaus auch Vorgänge einbeziehen, die rein praktischer Natur sind. Dies beginnt bei der ersten Begegnung des Anwalts mit seinen Klienten und endet meist bei der Abfassung der Honorarabrechnung. Aus diesen Gründen widmet sich das folgende Kapitel den mehr praktischen Vorgehensweisen. Hierzu werden zunächst relevante Normen der eher anwendungsorientierten Zivilprozeßordnung analysiert. In einem weiteren Teil erfolgt eine Untersuchung der praktischen Arbeitsmethoden anhand von verfügbaren Arbeitsmitteln. Die Analyse wird die Chronologie eines typischen Verfahrens verfolgen, um einen Überblick über den Datenfluß zu erhalten.

a) Mandatsaufnahme

Rechtlich bindende Vorschriften für den Vorgang des ersten Kontaktes zwischen Mandant und Rechtsanwalt existieren nicht. Dies heißt jedoch nicht, daß sich der Vorgang in völlig rechtsfreiem Raum bewegt. Es gelten vielmehr die für vorvertragliche Beziehungen sowie für Verträge anzuwendenden Vorschriften auch in diesem Fall. Kommt der Anwalt seinen Sorgfalts- und Betreuungspflichten aus dem Anwaltsvertrag (regelmäßig ein Dienstvertrag, in Einzelfällen auch ein Werkvertrag[169]) nicht nach, so droht ihm im Schadensfall der Rückgriff des Mandanten[170].

Die Mandatsaufnahme läßt sich zunächst in zwei Hauptvorgänge aufteilen, nämlich die Erfassung der für das weitere Prozedere **organisatorisch** notwendigen Daten und die eigentliche erste **juristische** Betreuung des Mandanten. Beide Vorgänge können zeitlich, räumlich und personell getrennt abgewickelt werden. Insbesondere die Erfassung der Personaldaten des Mandaten, aber auch diejenigen der Gegner muß nicht von einem rechtlich versierten Bearbeiter vorgenommen werden[171]. Diese Daten sind jedoch bereits in einem frühen Stadium, d.h. beim ersten Kontakt mit dem Mandanten notwendig, da ab sofort mit einem auch schriftlichem Kontakt zwischen Anwalt und Mandant zu rechnen ist. Insbesondere werden sofort Gebühren fällig, die abzurechnen sind. Für die Erfassung dieser **Stammdaten** ist es allerdings zweckmäßig, die formalen und inhaltlichen Erfordernisse im Auge zu behalten, die für einen späteren Schriftwechsel im Fall eines Prozesses zu beachten sind. Zur Erfüllung aller Voraussetzungen wird es nicht ausbleiben, daß auch der Rechtsanwalt selbst zumindest eine Überprüfung und Vervollständigung der Angaben vornimmt. Dies gilt besonders beim Auftreten von juristischen Personen oder in anderen Fällen von Vertretung oder Prozeßstandschaft. Bei der juristischen Überprüfung kommt es allerdings weniger auf die Korrektheit der eigentlichen Daten als auf die korrekte Zuschreibung der Daten zu den prozessualen Rollen an. Es ist also besonders darauf zu achten, wer die ladungsfähigen Parteien, also Kläger oder Beklagter sind, und wer als Zeuge auftritt oder auftreten kann.

Ein erhebliches Problem bei der Mandatsaufnahme stellt die Prüfung der **Kollision** des Mandats mit anderen Mandaten des Anwalts dar[172]. Hierzu muß überprüft wer-

---

[169] *Ponschab*, Rechtsanwaltshandbuch E I. RdNr. 1.

[170] Eine korrekte Rechtsanwendung liegt also auch im eigenen Interesse des Anwalts. Womöglich werden deshalb Fragen der Büroorganisation etwa von *Heussen*/*Büchting* (Hrsg.), Rechtsanwaltshandbuch 1995/96 (s. Vorwort) unter dem Aspekt der Haftung und nicht unter dem Aspekt optimaler rechtlicher Betreuung betrachtet.

[171] Laut *Mähler*, S. 37 wird die Stammdatenerfassung noch zu häufig vom Anwalt übernommen.

[172] *Waltl*, CoR 6/92 S. 18 ff.

den, ob der zukünftige Mandant in einem anderen Verfahren auf der Gegenseite oder der aktuelle Gegner in einem anderen Verfahren auf der Mandatsseite steht. Das erfordert folgendes Vorgehen:

- **Kennzeichnung** von allen beteiligten Personen des aktuellen Mandats. Hierzu müssen ein oder mehrere Merkmale herangezogen werden, die eine eindeutige Bestimmung der Person zumindest sehr wahrscheinlich machen.
- Überprüfen, ob entsprechende Personen an **anderen Verfahren** beteiligt sind.
- Kontrolle, welche **Rolle** die Personen in den anderen Verfahren innehaben. Dabei ist nicht nur interessant, ob der aktuelle Mandant in einem anderen Verfahren Gegner ist, sondern ob er zu einem Gegner in einer Beziehung (Vertreter, Geschäftsführer etc.) steht, die eine Interessenkollision befürchten läßt.[173]

Die genaue Aufklärung des Sachverhalts ist vertragliche Hauptpflicht des Rechtsanwalts und kann bei mangelhafter Durchführung zur Haftung führen[174]. Dabei muß der Anwalt nach dem Vortrag des Mandanten durch gezielte Fragen weitere **juristisch relevante** Tatsachen ermitteln. Dies erfordert eine simultane rechtliche Würdigung der jeweils bis dahin bekannten Tatsachen. Dabei ist es zweckmäßig möglichst viele denkbare Ziele zu überprüfen und die entsprechenden Anspruchsgrundlagen auf ihre Einschlägigkeit hin zu untersuchen. Fehlende Tatbestandsvoraussetzungen müssen durch Nachfragen abgeklärt werden. Bereits in diesem Gespräch erfolgt also eine Kategorisierung der Tatsachen nach juristischen Kriterien. Tatsachen werden also Tatbestandsmerkmalen der denkbaren Anspruchsgrundlagen oder entsprechender weiterführender Normen zugeordnet. Hier erfolgt die juristische Würdigung meist aus dem eigenen Wissen des Rechtsanwalts bzw. sehr oft eher intuitiv. Es stört nicht nur den Gesprächsfluß, sondern hat auch eine psychologisch eher negative Wirkung, wenn sich der Anwalt während des Gespräches bereits aus sekundären Quellen informieren müßte[175].

Diese juristische Voranalyse besteht also in einer groben Abschätzung des zu erreichenden Ziels und einer anspruchsorientierten Zuordnung der einzelnen Sachverhaltsmerkmale zu den Tatbestandsmerkmalen der für einschlägig erachteten Normen. Dieser Vorgang muß sich jedoch auf einem Level bewegen, der den Blick für unterschiedliche Verwendungsmöglichkeiten der einzelnen Fakten nicht verstellt. Es können also nur diejenigen Fakten als irrelevant vernachlässigt werden, die aus jeder denkbaren Sicht abwegig erscheinen und sich entsprechend keinem berührten Tatbestandsmerkmal zuordnen lassen.

Bei klassischer Arbeitsweise erfolgt die Niederlegung der erfaßten Tatsachen in einer Aktennotiz[176]. Hier wird der Sachverhalt so beschrieben, wie er sich nach dem Gespräch aus juristischer Sicht darstellt. In sehr stereotypen Verfahren, wie etwa in Ehesachen, kann eine Sachverhaltsaufnahme zunächst auch mittels eines vorgefertigten Formulars erfolgen. Ein derartiges Formular ermöglicht es, eine konkrete Analyse der bereits zu Beginn eines Verfahrens erfaßten Fakten sowie der jeweiligen juristischen Würdigung vorzunehmen.

b) Der Aufnahmebogen *Familiensachen*

Formulare sind in der Regel das Ergebnis einer erheblichen systematischen Aufarbeitung eines Vorgangs und der mit ihm verbundenen Rechtsfragen. Die besondere Lei-

---

[173] *Ponschab*, Rechtsanwaltshandbuch E I. 22.

[174] *Commichau*, RdNr. 6 ff, m.w.N.

[175] *Bährig*/*Roschmann*/*Schäfer* S. 57 f.

[176] *Commichau*, RdNr. *9.*

stung liegt dabei in der strengen Formalisierung des Sachverhaltes. Es lohnt sich deshalb, einen Blick auf ein Formular zu werfen. Wesentlicher Gesichtspunkt hierbei ist die Art der Umsetzung von praktischen und juristischen Bedürfnissen in eine formale Eingabe. Der Aufnahmebogen für Familiensachen[177] ist durch seine Komplexität und Ausgereiftheit von besonderem Interesse.

Im Zuge der folgenden Analyse des Formulars soll en passant die Prämisse überprüft werden, daß die sich die dort repräsentierten Fakten mit Hilfe der Relationendarstellung repräsentieren lassen. Weiterhin sollen Besonderheiten aufgezeigt werden, die bisherige Annahmen bestätigen oder erweitern.

Das Formular zeigt bereits auf der ersten halben Seite die fast triviale Standardkonstellation einer Familiensache. Es gibt zwei Ehepartner sowie eine Anzahl von Kindern. Von den Beteiligten sind eine Reihe persönlicher Daten anzugeben. Diese Personen sind - jedenfalls im Sinne der Prädikatenlogik - keine Fakten, sondern Individuen. Ihre **Existenz** hingegen ist ein Faktum genauso wie die weiteren Angaben (Name, Geburtstag/-ort, Nettoeinkommen etc.) Die ersten Zeilen stellen also formal gesehen die Existenz der beiden Ehepartner sowie eine Reihe von Eigenschaften fest. Eine Familiensache im Sinne des Fragebogens ist regelmäßig mit einer Scheidung verbunden.

aa) Datum

Der Fragebogen beginnt mit der Angabe des Datums der Verfahrensaufnahme. Dieses Datum hat im eigentlichen Verfahren prozessual und materiellrechtlich keine weitere Bedeutung. Es dient vielmehr der Aufzeichnung der Verfahrensgeschichte. Insbesondere kann man so eine zeitliche Abfolge sich möglicherweise ändernder Sachverhaltsdarstellungen rekonstruieren.

Die besondere Bedeutung der zeitlichen Veränderung von Fakten und der Veränderung von Angaben über Fakten wurde bereits festgestellt[178]. Sie wird mit der exponierten Stellung innerhalb des Fragebogens nochmals betont. Der Bewertung dieser Information in einem technisch unterstützten System kommt eine eigene Rolle zu. Das Datum wird quasi jeder weiterhin erfaßten Information anzuheften sein. Hingegen muß es nicht zwingend als eigene Relation repräsentiert werden.

bb) Mandant / Antragsteller

Das Formular legt zuerst fest, wer Mandant und wer Antragsteller ist. Die Information wird vielfach benötigt, sei es für die Formulierung und Adressierung der fallspezifischen oder auch nur der mandatsspezifischen Schriftsätze. Die Intelligenz in der hier gewählten Lösung liegt in dem Verzicht, Adresse und Anschrift von Mandant, Antragsteller sowie Ehefrau und Ehemann jeweils getrennt zu erfassen. Vielmehr werden gerade die Adressen der Beteiligten redundanzfrei an einer Stelle erfaßt. Die möglichen Rollen der Beteiligten werden hiervon getrennt erfragt.

Bei dem Begriff *Mandant* handelt es sich dem ersten Anschein nach um eine Eigenschaft des Ehemannes oder der Ehefrau. Die Darstellung im Formular impliziert eine Exklusivität, d.h., daß jeweils nur eine Person diese Eigenschaft haben kann. Bei weiterer Analyse wird man allerdings von einer Beziehung wie etwa *ist Mandant von* auszugehen haben, die Mandant und Anwalt verbinden. Zieht man allerdings das unter a) zur Interessenkollision Gesagte in Betracht, so ist zu erkennen, daß sich die hier gemeinte Beziehung auf ein einziges Mandat beschränkt. Die ent-

---

[177] *Commichau,* RdNr. *345;* Anhang C, S. 242 ff zeigt ein Formular der *Hans Soldan GmbH*. Die Randnummern sind zur Orientierung eingefügt.

[178] S. S. 34.

sprechende Person ist also Mandant in einem Mandat. Sie kann nebenher auch noch in anderen Mandaten Mandant oder Gegner sein. Letzteres wäre allerdings rechtlich unzulässig. Die Relation ließe sich also als *Mandant*(*mandant, mandat*) formulieren. Der Anwalt selbst kann in dieser Attributleiste fehlen. Er steht zu dem **Mandat** in einer anderen Relation etwa *Mandatar*(*mandatar, mandat*). Eine Aufnahme in die erste Attributleiste würde zu einer unnötigen Redundanz führen. Für die Angabe **Antragsteller** gilt im Prinzip dasselbe. Auch dieser Begriff beschreibt eine Beziehung *Antragsteller*(*antragsteller, antrag*) der angekreuzten Personen zu dem im Raum stehenden Antrag. Aufgrund des zugrundeliegenden Szenarios **Scheidung** impliziert diese Relation gleichzeitig auch, daß die nicht angekreuzte Person **Antragsgegner** desselben Antrags ist.

cc) Ehevertrag

Die Frage, ob ein **Ehevertrag** vorliegt, beeinflußt den Fortgang des Verfahrens erheblich, da in diesem Fall in der Regel ein Zugewinnausgleich obsolet ist. Es sind einige Darstellungsformen eines Ehevertrages in Form einer Relation denkbar. Es ist nämlich Interpretationssache, zu welchem Individuum des Sachverhaltes der Vertrag in Relation steht. So kann er etwa zu den Ehepartnern in Beziehung stehen oder zu der Ehe als eigenständiges Objekt.

Der hier gemeinte Ehevertrag stellt eine Abänderung der sich aus der Ehe (*Ehe*(*ehe, ehemann, ehefrau*))gem. §§ 1353 ff. ergebenden Rechte und Pflichten dar. Der Ehevertrag bindet zwar die Parteien, ist jedoch untrennbar mit der Ehe verbunden[179]. Es ist deshalb denkbar, den Begriff Ehevertrag als Relation *Ehevertrag*(*ehevertrag, ehe*) zu begreifen. Einer Relation *Ehevertrag*(*ehevertrag, ehemann, ehefrau*) hingegen mangelt es an der Bindung an die **Ehe** selbst. Eine Formulierung *Ehevertrag*(*ehevertrag, ehe, ehemann, ehefrau*) enthielte eine Redundanz, da sich die Ehepartner bereits zwangsläufig aus der Relation zum Element *Ehe*(*ehe, ehemann, ehefrau*) ergeben würden.

dd) Gegenstandswert

Die Angabe des Gegenstandswertes hat in Familiensachen nicht die herausragende Bedeutung wie in anderen Verfahren. Aufgrund der Allzuständigkeit des Amtsgerichtes gem. § 606 ZPO, hat er keinen Einfluß auf die sachliche Zuständigkeit. Vielmehr beschränkt sich seine Wirkung auf die Verfahrenskosten, insbesondere auf die Gebühren des befaßten Anwalts.

Der Gegenstandswert steht in unmittelbarer Beziehung zum Verfahren selbst. Dieses muß zur Abbildung prozessualer Informationen als eigenes Element eingebracht werden: *Gegenstandswert*(*gegenstandswert, verfahren*).

ee) Personendaten

Personendaten haben die unterschiedlichsten Aufgaben im Verfahren. Sie dienen etwa als Adresse für die Zustellung des Schriftverkehrs (§ 130 ZPO) sowie der Klage- bzw. Antragsschrift (§ 253 ZPO). Sie sind notwendiger Bestandteil des angestrebten Urteils und bestimmen den Ort eventueller Vollstreckungsmaßnahmen (§§ 750, 313 ZPO). Geburtsdaten dienen der taggenauen Altersberechnung. Angaben über das Einkommen werden vielfältig benötigt, etwa für die Beantragung von Prozeßkostenhilfe (§§ 114 ff. ZPO) oder für die Ermittlung des Lebensstandards als Grundlage eines Unterhaltsanspruchs (§ 1578 BGB). Die Adresse des Arbeitgebers ist notwendig, da dieser als Zeuge geladen werden kann, oder für eine Vollstreckung

[179] Etwas anders mag die Sache bei der **fortgesetzten Gütergemeinschaft** liegen. Sie reicht über das Ende der Ehe durch Tod eines Ehegatten (nicht aber durch Scheidung) hinaus.

in das Gehalt eines Schuldners. Formal wird die Anschrift auch im *Fragebogen zum Versorgungsausgleich*[180] benötigt, sofern eine betriebliche Altersversorgung zugesagt wurde. Die Anzahl der nichtehelichen Kinder oder Kinder aus früheren Ehen gibt Aufschluß über eventuelle weitere Unterhaltspflichten eines Teils und eine notwendige Neuverteilung des Unterhalts.

aaa) Nicht digitale Daten

Die Eigenart der an sich geläufigen Begriffe, *Name*, *Geburtstag*, *Nettoeinkommen* etc. ist, daß sie keine Wahrheitswerte, sondern Zeichenfolgen oder Zahlenwerte bezeichnen. Begriffe dieser Art wurden bereits erwähnt[181]. Ebenso wurde klargestellt, daß jedenfalls die Prädikatenlogik Elemente unterschiedlicher Objektmengen, also auch aus Mengen von Zahlen oder Zeichenketten verarbeiten kann. An dieser Stelle kann man sich deshalb zunächst damit behelfen, den entsprechenden Wert in beliebigen Variablen (*x*, *y* oder *z*) aufzunehmen. Hieraus ergeben sich bezüglich einer beliebigen Person (*p*) z.B. folgende Relationen: $Name(x, p)$, $Geburtstag(y, p)$ oder $Nettoeinkommen(z, p)$.

Der Güterstand kann eine Zugewinngemeinschaft, eine Gütergemeinschaft oder eine Gütertrennung sein. Entgegen der Darstellung im Formular gilt der Güterstand für Ehemann und Ehefrau gleichermaßen. Eine Besonderheit des Begriffs *Güterstand* liegt darin, daß er einen Oberbegriff zu den Formen der Güterstände darstellt. Die juristisch relevante Frage wird selten lauten: *Wie ist der Güterstand?* Vielmehr werden Angaben darüber benötigt, ob abweichend vom gesetzlichen Güterstand der Zugewinngemeinschaft konkret Gütertrennung (§ 1414 BGB) oder Gütergemeinschaft (§§ 1415 ff. BGB) vereinbart ist. Es ist also zum einen möglich, einen Sammelbegriff $Güterstand(x, e)$ zu bilden, wobei *x* aus der Menge {*Zugewinngemeinschaft*, *Gütertrennung*, *Gütergemeinschaft*} zu entnehmen ist. Zweckmäßiger ist jedoch die Bildung der Prädikate $Zugewinngemeinschaft(ehe)$, $Gütergemeinschaft(ehe)$ und $Gütertrennung(ehe)$.

bbb) Anzahl von Objekten

Die Anzahl der Kinder der Beteiligten aus früheren Ehen oder auch der nichtehelichen Kinder stellt im Prinzip die Anzahl von Individuen dar, die das Merkmal haben, Kind des einen oder anderen Ehegatten zu sein. Zugrunde liegen also hier die Begriffe $Kind\ aus\ früherer\ Ehe(kind, ehe)$ sowie $nichteheliches\ Kind(kind, ehe)$. *ehe* bezeichnet dabei die aktuell im Streit stehende Ehe. Wird für *ehe* eine Konstante, etwa *Ehe von A und B* eingesetzt, so ergibt sich eine Menge von Individuen *k*, für die diese Relation wahr ist. Die gesuchte Anzahl der Personen entspricht der **Mächtigkeit** der Menge. Sie stellt also eher eine mathematische Funktion als einen juristischen Begriff dar. Unterstellt, man möchte mit einem minimalen Begriffsumfang auskommen, so erscheint es wenig sinnvoll zwei Prädikate, $Kind\ aus\ früherer\ Ehe(k, e)$ und $Anzahl\ Kinder\ aus\ früherer\ Ehe(k, e)$ zu bilden. Vielmehr sollte die Funktionalität $x = Anzahl(y)$ syntaktisch getrennt werden. Dies eröffnet ohne weitere Begriffsbildung die Möglichkeit, für jeden Begriff die Anzahl der Individuen zu bestimmen, die auf ihn unter einer bestimmten Konstellation zutreffen.

ccc) Tabellarische Informationen

Neben der Frage nach der Anzahl von Individuen, die sich aufgrund eines Merkmals auszeichnen, steht das Bedürfnis, die Eigenschaften der Individuen darzustellen.

---

[180] S. Anhang 0, S. 247 ff.

[181] Zu **Begriffsformen** s.S. 32.

Das Formular bedient sich hierzu an mehreren Stellen einer tabellarischen Darstellung. So werden für die persönlichen Daten der Ehepartner die Informationen vertikal und die Individuen *Ehemann* und *Ehefrau* nebeneinander dargestellt. Für die Daten der Kinder werden die Individuen vertikal, die Daten selbst horizontal angeordnet. Diese Anordnung ist nicht nur aus Platzgründen sinnvoll. Sie ergibt sich einfach aus der Tatsache, daß die Anzahl der Ehepartner fest steht, die Anzahl der Kinder hingegen theoretisch unbegrenzt sein kann. In diesen Fällen ist das Formular auf eine plausible Begrenzung der Eingabemöglichkeiten (hier vier Kinder) angewiesen.

Die Relation *Kind*(*kind*, *vater*, *mutter*) ist nicht geeignet ein Kind eindeutig zu identifizieren[182]. Ein System zur Beschreibung von Fakten muß in der Lage sein, den Zugriff auf alle Daten zu einer beliebigen Anzahl von Individuen zu erlauben, die sich durch ein Merkmal auszeichnen. Eine Begrenzung auf eine willkürlich limitierte Anzahl von Individuen entspricht nicht dem hier verfolgten Ziel. Ein Ansatz hierzu ist die Kennzeichnung der Objekte durch einen Index, wie es auch in der Tabelle des Formulars geschehen ist. Die Relation wäre dann auf den Ausdruck *Kind*(*kind*, *vater*, *mutter*, *x*) zu erweitern, wobei *x* die Position des jeweiligen Kindes in der Menge aller Kinder angibt.

ddd) Redundante Informationen

Näher zu betrachten sind nun noch die Begriffe *Alter* sowie *geboren am*. Ihre Besonderheit besteht in der gänzlich **unstreitigen** linearen Abhängigkeit voneinander. So ergibt sich das Alter zwangsläufig aus dem Geburtstag einer Person und dem aktuellen Datum. Wird das Alter an einem anderen Datum gesucht, kann dies ebenfalls eindeutig berechnet werden.

Im Gegensatz zu den meisten rechtlichen Beurteilungen, denen ein immer bleibender Rest an Spielraum eine automatisierte Berechnung versagt, stellen Alter und Geburtsdatum zweifelsfrei redundant vorhandene Informationen dar. Da redundante Informationen immer die Gefahr von Fehlern in sich bergen, scheint es vernünftiger, für nicht meinungsabhängige Schlußmöglichkeiten Sonderfunktionen einzuführen, die eine direkte Konvertierung der gesuchten Information ermöglichen. Benötigt also ein Vorgang die Information **Alter der Person x am Datum y**, so kann ein Mechanismus greifen, der das Geburtsdatum ermittelt und das Alter ungefragt errechnet.

Alternativ hierzu ist es auch denkbar, die Information **Alter** grundsätzlich gar nicht vom System her zu übermitteln. Jeder Vorgang ist dann für die Berechnung selbst verantwortlich. Im angesprochenen Fall des Alters ist dies auch nicht sonderlich aufwendig. Entsprechende Begriffe sind dann wie in der ersten Alternative als Elemente für den Datentransport für tabu zu erklären. Es wird jedoch auch keine Einzelfallregelung implementiert, wenn sie zu einem Systembruch führen würde. Vermittelnd ist auch zu erwägen, ein System so zu gestalten, daß es für derartige Einzelfallerweiterungen grundsätzlich offen ist.

ff) Trennungszeitpunkt

Der Trennungszeitpunkt ist an sich ein eindeutiger Datumswert. Er kann zunächst als *Trennungszeitpunkt*(*t*, *m*, *f*) dargestellt werden. Er wird jedoch in vielen Fällen umstritten sein. Insbesondere wenn ein Ehegatte an der Ehe festhält, wird er sich bemühen, den Trennungszeitpunkt möglichst spät anzusetzen, um so die Frist des § 1566 II BGB zu erreichen. Der Scheidungswillige strebt nach dem Gegenteil.

---

[182] S. S. 44.

Bei der weiteren Kategorisierung von Tatsachen, die den Trennungszeitpunkt betreffen, verweigert das Formblatt seine Unterstützung und verweist auf eine Freitexteingabe. Benötigt wird statt dessen allerdings eine Reihe von Zeitpunkten, an denen bestimmte Verhaltensweisen begonnen haben, die wiederum auf eine Trennung schließen lassen. Der Trennungszeitpunkt ließe sich dann danach ermitteln, wann eine gewisse Kumulation von Verhaltensweisen eine Trennung indiziert. Die Entscheidung hierüber hängt stark von dem Einzelfall ab. Eine Kategorisierung kann anhand von Begriffen erfolgen, die aus der Rechtsprechung zu § 1567 BGB zu bilden sind. Letztendlich abschließend kann eine Begriffsliste diesbezüglich aufgrund der starken Kasuistik[183] jedoch nie sein. Beispiele sind etwa die Begriffe[184]:

- getrennte Wohnungen
- getrennte Schlafzimmer
- Aufteilung der Küchenbenutzung
- kein ehelicher Verkehr etc.

Hier eröffnet sich ein Gebiet kasuistischer Entscheidungsfindung mit einer nicht mehr überschaubaren Begrifflichkeit. Auch aus diesem Grunde lassen sich alle möglicherweise zu erhebenden Fakten nicht mehr auf dem begrenzten Feld eines Formulars unterbringen. Auch der Versuch einer Algorithmisierung dieser Kasuistik mag scheitern[185]. In jedem Fall ist die Begriffsbildung erheblich von den Algorithmen abhängig, die zu diesem Zweck entwickelt werden. Die Herausarbeitung von Begriffen ohne eine vernünftige Möglichkeit, diese Begriffe in einzelnen Vorgängen weiter zu verarbeiten, ist kaum sinnvoll. Vielmehr bleibt es mangels definierter Vorgänge dem Rechtsanwender überlassen, den seiner Ansicht nach gültigen Trennungszeitpunkt festzustellen.

Zu überdenken bleibt, ob und wie eine Freitexteingabe, wie sie hier vorgesehen wurde, auch in dem hier angestrebten EDV-System zu implementieren ist.

gg) Verfahrenstechnische Informationen

Bei Punkten wie *Vollmacht* oder *Prozeßkostenhilfeunterlagen* handelt es sich um Informationen, die vor allem das weitere Prozedere des Verfahrens betreffen. Insbesondere dienen sie als Hinweis, welche Unterlagen bis zur Antragstellung noch nachgereicht werden müssen.

Eine Vollmacht ist ein Objekt im Sinne der Prädikatenlogik, das die Eigenschaft besitzt, eben eine *Vollmacht* zu sein. Genauer betrachtet handelt es sich allerdings um eine Relation $\textit{Vollmacht gem 609 ZPO}(\textit{vollmacht, verfahren, anwalt, mandant})$ zwischen dem Mandanten, dem Anwalt und dem Verfahren (§ 609 ZPO). Die Bejahung der Frage *Vollmacht* bedeutet die Bejahung der Aussage $\exists x(\textit{Vollmacht gem 609 ZPO}(x, \textit{Verfahren, Anwalt, Mandant}))$. Ein EDV-System sollte den Zugriff auf die Information über die Existenz eines gesuchten Objekts behandeln können.

Ein Dokument wie die Vollmacht ist organisatorisch zu verwalten. Dies kann durch einen Verweis auf die Seite der Akte, das Ausstellungsdatum oder einen Verweis auf einen Dateinamen im Computer geschehen. Um eine optimale Arbeitsunterstützung zu gewähren, ist es sinnvoll, einen Verweis auf diese Kategorie zuzulassen. Ein derartiger Verweis ist eine Information, die sich zwar nicht auf die rechtliche

---

[183] *Wolf*, MüKo, § 1567 RdNr. 28.

[184] *Wolf*, MüKo, § 1567 RdNr. 28 ff.; *Diederuchsen*, Palandt, § 1567 RdNr. 3 ff.

[185] Nach *Philipps*, jur-pc 90, 820, 824 f. ist dies wohl ein klassischer Fall für die Anwendung eines Neuronalen Netzes..

Beurteilung des Sachverhalts bezieht, wohl aber die Bearbeitung des Falls beeinflußt. Ähnliches gilt für die meisten anderen Unterlagen.

Die übrigen Punkte des Formulars *Aufnahmebogen* zeigen keine weiteren Besonderheiten auf. Vielmehr werden diverse klassische Tatbestandsmerkmale in Form von Checklisten abgefragt. Dabei handelt es sich teilweise um Rechnungsposten etwa für die Berechnung des Zugewinnausgleichs. Teilweise sind es Tatbestandsmerkmale entsprechender Vorschriften.

Festzustellen ist, daß Daten aller bisher auch erwähnter Typen erhoben werden. Die Mitführung des Erhebungsdatums ist dabei wesentlich für die Rekonstruktion des Informationswertes. Redundante Informationen wie Geburtsdatum und Alter werden auf eine Information, also das Grunddatum zurückgeführt. Für die Berechnung der Anzahl von Objekten mit einem gemeinsamen Merkmal (Kinder von X) sowie für den Zugriff auf deren Eigenschaften und Relationen sind Zugriffsmethoden zu entwickeln. Weiterhin ist an den Grenzen der Kategorisierbarkeit von Fakten auf eine freie Begründungskomponente in Form einer Notiz zurückzugreifen.

c) Weitere Formulare

Das Formblatt **Aufnahmebogen für Ehe- und Familiensachen** bildet sozusagen die zentrale Grundlage für die Fakten eines familienrechtlichen Sachverhalts[186]. Als Beispiel für weitere Formulare seien hier noch kurz der *Fragebogen zum Versorgungsausgleich*[187] sowie die *Erklärung über die persönlichen und wirtschaftlichen Verhältnisse*[188] erwähnt. Dies wird insbesondere zum Anlaß genommen, auf Möglichkeiten der Datenübernahme hinzuweisen.

aa) Persönliche Daten

Zunächst einmal sind in jedem der Formulare die allgemeinen persönlichen Daten der bezogenen Person anzugeben. Diese Daten sind zwischen den Vorgängen (die Bearbeitung eines Formulars wird hier als **Vorgang** bezeichnet) geradlinig austauschbar. Die Frage nach den **unterhaltsberechtigten Angehörigen** ist bei der PKH von den ehelichen Kindern zu unterscheiden. Sie bilden eine Teilmenge der Angehörigen. Nur insoweit können die Angaben aus dem Aufnahmebogen übernommen werden. Die hier gelisteten Objekte müssen also eine Beziehung zu dem Antragsteller wie *Unterhaltsberechtigter Angehöriger*$(angeh, as)$ haben. Diese Beziehung kann der Aufnahmebogen zumindest den minderjährigen Kindern ohne Einkommen automatisch zuschreiben (§ 1602 II BGB). Dabei muß er den Geltungszeitraum der Aussage zunächst entsprechend limitieren. Alternativ kann ein eigener Vorgang zwischengeschaltet werden, der zum Zeitpunkt der Anfrage durch den Vorgang *PKH–Antrag* überprüft, welche Objekte unterhaltsberechtigt sind.

bb) Nettoeinkommen

Die PKH–Erklärung enthält eine detaillierte Aufstellung der einzelnen *Einnahmen (brutto)* des Antragstellers sowie der Abzugsposten. Sie erwartet dabei prinzipiell alle Angaben, die zu einer Ermittlung des Nettoeinkommens erforderlich sind. Hingegen wird im Aufnahmebogen lediglich eine Gesamtsumme des *mtl. Nettoeinkommens* angegeben. Es stellt sich nun die Frage, inwieweit die Angaben miteinander korrelieren. Problematisch ist dabei, daß der Begriff des monatlichen Nettoeinkommens im Aufnahmebogen nicht näher spezifiziert ist. Der Aufnahmebogen selbst hat jedoch auch keine rechtliche Bedeutung. Er dient lediglich der Sammlung von Fak-

---

[186] *Commichau*, RdNr. *345*.

[187] S. Anhang 0, S. 247 ff.

[188] S. Anhang D, S. 246 f.

ten für die spätere Weiterverwertung. Die hier verwendeten Begriffe definieren sich also letztlich aus den später durchzuführenden Vorgängen. Dabei wird das Nettoeinkommen letztlich zur Beurteilung der PKH oder zur Bemessung des Unterhalts des Kindes sowie des Ehegatten herangezogen.

In der Praxis wird der Unterhalt aus $\frac{3}{7}$ des Erwerbseinkommens[189] und $\frac{1}{2}$ des anrechenbaren sonstigen Einkommens[190] errechnet. Hier ist also das Nettoeinkommen als undefinierte Größe wenig hilfreich. Der Kindesunterhalt bemißt sich nach der **Düsseldorfer Tabelle**[191] auf Grundlage des monatlichen Nettoeinkommens. Welche Abzüge dabei berücksichtigt werden, ist teilweise regional unterschiedlich[192]. Es ist anzunehmen, daß der entsprechende Begriff im Fragebogen mit demjenigen aus der Düsseldorfer Tabelle übereinstimmt. Ebenso bemißt sich die **Prozeßkostenhilfe** nach dem monatlichen Nettoeinkommen (§ 114 ZPO). Die Abzüge zur Berechnung des Nettobetrags ergeben sich hier aus den §§ 114 ff. ZPO sowie dem BSHG. Damit sind die Begriffe des **monatlichen Nettoeinkommens** jedenfalls nicht inhaltsgleich und somit nicht gegeneinander austauschbar. Der Sammelbegriff im Aufnahmebogen geht formal also ins Leere, da er für die unterschiedlichen Berechnungen eingesetzt werden kann. Er beschreibt vielmehr eine Art Regelfall für die Nettoberechnung.

Für eine Implementierung des Themenbereiches ist sinnvoll, daß die Einzelposten soweit wie möglich zerlegt und einzeln erhoben werden. Dies entspricht dem Vorgehen im PKH–Formular. Die einzelnen Nettowerte sind dann anhand der Einzelposten zu berechnen.

cc) Exemplarische Posten

Die Vielfältigkeit möglicher rechtserheblicher Tatsachen führt in Formularen wie bereits festgestellt zu Bereichen mit Freitexteingabe. Im Fall des PKH–Formlars können beispielsweise weitere Abzugsposten benannt werden. Die Angabe der Posten erfolgt in einem Erklärungsfeld exemplarisch.

Bei einer technischen Implementierung ist anzustreben, auch die hier genannten Beispiele als Relationen auszuformulieren. Dennoch wird es nicht ausbleiben, derartige freie Bezeichnungen zuzulassen. Hierzu wird ein Begriff zu formulieren sein, der eine Oberkategorie bildet, etwa *abzugsfähiger Posten*. Das Objekt, welches dieses Kriterium erfüllt, muß dann über einen entsprechenden Textwert bezeichnet werden.

dd) Vermögen

Die Auflistung Vermögensgegenstände des Aufnahmebogens dient zwar der Ermittlung des Zugewinnausgleichs, ist aber mit der des PKH–Antrags weitgehend deckungsgleich. Ein Problem mag sich daraus ergeben, daß im PKH–Formular die Summe des Vermögens des Antragstellers und seines Ehegatten erwartet werden. Dies erscheint im Scheidungsverfahren widersinnig. Vielmehr wird man in diesem Fall die Einzelposten ausweisen müssen und die Summe getrennt bilden.

---

[189] Zum *Arbeitseinkommen* vgl. *Kalthoener*/*Büttner*, RdNr. 693 ff.;

[190] Düsseldorfer Tabelle S. *Brudermüller-Klattenhof*, **TzFamR**, S. 15.; weitere Nachweise in *Wendl*/*Staudigl*, S. 2 f.

[191] S. etwa *Brudermüller-Klattenhof*, **TzFamR**, S. 13.

[192] Vgl. die Leitlinien der Oberlandesgerichte zum Unterhaltsrecht; Nachweise der einzelnen Fundstellen in *Wendl*/*Staudigl*, S. 3.

ee) Versorgungsausgleich

Der Fragebogen zum Versorgungsausgleich enthält strukturell kaum Besonderheiten. Es sind auch wenig Redundanzen zum Aufnahmebogen erkennbar. Ein Augenmerk ist auf die Angaben zum aktuellen und den früheren Arbeitgebern zu richten. Zum einen manifestieren diese Angaben die bereits dargestellte Forderung nach einer zeitlichen Einordnung von Fakten[193]. Es müssen also Beziehungen zu mehreren Individuen des Inhalts *Arbeitgeber* angelegt werden, wobei für die Relationen der Geltungszeitraum anzulegen ist.

Etwas eigenwillig ist dabei die Darstellung der einzelnen Arbeitgeber. Die Informationen werden an mehreren Stellen getrennt abgefragt. Abhängig ist dies von der Frage, ob der Arbeitnehmer die Fragen zur Zusicherung einer Altersversorgung beantworten kann oder nicht. In einer technischen Implementierung ist es sicherlich sinnvoller, die Daten aller Arbeitgeber zu erfassen. Die Frage nach der Altersversorgung muß in diesem Fall auf einen bewußt unsicheren Wert (also weder *ja* noch *nein*) gesetzt werden können.

d) Weiterer vorprozessualer Ablauf

Das weitere Verfahren vor einem Prozeß erfolgt ebenfalls formlos. Das Mindestmaß an Korrespondenz ergibt sich aus § 93 ZPO. Die folgende Beschreibung wird sich daran orientieren, wie der Abgleich der vorhandenen Informationen im Dialog mit dem Gegner erfolgt.

Typischerweise wird der Gegner den Sachverhalt bereits in der vorprozessualen Phase anders darstellen als der Mandant[194]. Aufgrund dieser Darstellung und der Beweislage werden weitere Fakten benötigt oder ein Teil des Tatsachenvortrages wird umgestellt. Weiterhin können durch den Schriftwechsel auch Tatsachen zutage treten, die der Mandant bisher aus zahllosen Gründen nicht erwähnt hat. Hinzu kommt die Suche nach erfolgversprechenden Beweismitteln. Sollten sich Probleme bei der Rechtsanwendung zeigen, so sucht der Anwalt nach Rechtsmeinungen, die die für seinen Mandanten günstigste Rechtsfolge vertreten[195]. Wesentlicher Arbeitsvorgang während dieser Phase ist der ständige Abgleich der neu vorgetragenen Tatsachen zum einen mit den bisherigen Behauptungen und zum anderen mit der sich hieraus ergebenden Rechtslage.

---

[193] S. 34.

[194] Etwas anderes gilt für den Fall der einverständigen Scheidung i.S. von §§ 1566 I, 1565 BGB sowie § 630 II ZPO.

[195] So z.B. *Nack*, **Informationstechnik..**, S. 47.

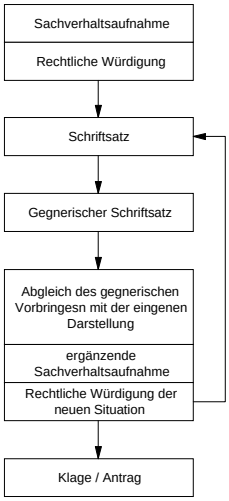


Abbildung 2: Prozedere vor Klageerhebung / Antragstellung

Die Vorgehensweise soll zunächst an einem Beispiel exerziert werden:

Auf den Versuch des Anwalts hin, mit Schreiben vom 10.3.92 auf eine einverständliche Scheidung hinzuwirken, verweigert der gegnerische Anwalt in einer Replik vom 15.3.92 die Zustimmung. Er trägt unter anderem vor, die Ehegatten wären vom 5.1.92 bis zum 7.2.92 gemeinsam in Australien gewesen. Dort sei es auch zum ehelichen Verkehr gekommen. Die Ehegatten hätten sich dort versöhnt und erst aufgrund eines Streits auf dem Rückflug wären sie erneut eigene Wege gegangen.

Des weiteren beziffert er den Nettomonatslohn seines Klienten auf DM 4.500, -.

Der Gegner behauptet also global betrachtet drei Fakten: den **gemeinsamen Urlaub**, die **Versöhnung** sowie sein **Jahresgehalt**. Nun sind zur Bearbeitung des Schriftsatzes alle älteren Angaben heranzuziehen. Nahezu eindeutig ist zunächst das Nettogehalt. Es stellt einen eigenen juristischen Begriff dar. Im Aufnahmebogen wurde das $Nettoeinkommen(x, p)$ erfaßt. Durch eine entsprechende Anfrage $Nettoeinkommen(x, M)$ sollte dieser Wert (x) zu ermitteln sein. Zu beachten ist dabei noch die notwendige Abklärung des Begriffs **Nettomonatslohn**. Es muß klargestellt werden, welche Abzüge bei den einzelnen Angaben tatsächlich getätigt wurden, was also **Netto** letztlich meint. Eine Abweichung der Angaben kann sich zum einen aus unterschiedlichen Tatsachenannahmen, zum anderen aber aus unterschiedlichen Berechnungsmodalitäten ergeben. Zunächst einmal wird die Zuordnung der Angabe zu dem Begriff erfolgen, der in dem jeweiligen Kontext üblich ist. Das Einkommen aus dem Fragebogen wird somit demselben Begriff zugeordnet wie die neue Angabe, da dem Augenschein nach beide Angaben denselben Zweck verfolgen, nämlich eine Berechnungsgrundlage für eventuelle Unterhaltsansprüche zu bieten. Stellt sich im Laufe des Verfahrens heraus, daß die Angaben nicht auf denselben Berechnungsgrundlagen beruhen, etwa weil unterschiedliche Abzüge getätigt wurden, so muß eine Korrektur entweder des Wertes oder der Zuordnung erfolgen.

Mit der wie auch immer gearteten Zuweisung des neuen Wertes zum Begriff **Nettoeinkommen** gibt es zwei (gleiche oder unterschiedliche) Werte für diesen Begriff bezogen auf den Sachverhalt. Sie stammen allerdings aus unterschiedlichen Vorgängen und/oder von unterschiedlichen Quellen. Benötigt ein weiterer Vorgang den Wert des Nettoeinkommens, so wird er entscheiden müssen, welchen der Werte er wünscht. Hierzu benötigt er primär folgende Angaben:

– Anzahl der verfügbaren Werte

– Quelle der einzelnen Werte
– Zeitpunkt der Behauptung des Wertes

Weiterhin muß im System ein Mechanismus existieren, der entscheidet, welcher Wert an einen Vorgang übermittelt wird, wenn dieser ohne weitere Angaben nach dem **Nettoeinkommen** fragt und mehrere Werte verfügbar sind.

Für die ersten Feststellungen im Ausgangsfall, also bezüglich des Urlaubs, fällt eine Klassifizierung schwerer als für die Einkommensangabe. Sie enthalten die lediglich **implizite** Behauptung, daß kein ausreichend langes Getrenntleben i.S. von § 1567 BGB vorliegt. Zunächst einmal ergibt sich also die Frage, von welcher Trennungszeit bisher ausgegangen wurde. Hieraus ist eine Anfrage nach dem Trennungszeitpunkt zu formulieren, der im Aufnahmebogen erfaßt wurde. Sie lautet beispielsweise *Trennungszeitpunkt*(t, M, F). Die erhaltene Antwort ist mit den neuen Behauptungen zu vergleichen. Ergibt sich ein Unterschied, so kann dies zum einen daran liegen, daß von unterschiedlichen Ereignissen *t* ausgegangen wird, an denen die Trennung stattgefunden haben soll. Oder demselben Ereignis entspricht nach den einzelnen Ansichten ein anderer kalendarischer Wert. Im einen Fall wäre das Ergebnis der Anfrage also, daß die Trennung nicht mit dem Rückflug erfolgte, sondern etwa zu einem früheren **Anlaß**. Im zweiten Fall wird zwar übereinstimmend der Rückflug als Trennungszeitpunkt angenommen, dieser wird jedoch unterschiedlich **datiert**.

Für beide Fälle ergeben sich beim weiteren Vorgehen unterschiedliche Strategien. Im Fall differierender Datierungen werden Anhaltspunkte und Beweise zu suchen sein, die die eigene Datierung belegen. Das Flugticket würde hier sicher genügen. Im anderen Fall wird das Argument des Urlaubs generell zu entkräften sein. Dies kann dadurch geschehen, daß der Urlaub als Ganzes bestritten wird. Ihm kann aber auch die Wirkung abgesprochen werden, daß die Parteien während dieser Zeit zusammengelebt haben und somit die Frist des § 1566 II BGB neu in Lauf gesetzt haben. Hier wird man neben der Tatsachenfindung auch auf die einschlägige Rechtsprechung zurückgreifen müssen, die einen gemeinsamen Urlaub nicht zwangsläufig als Beendigung einer Trennungszeit sieht[196]. Zur Erfassung der vom Gegner behaupteten einzelnen Fakten müssen diese also unterschiedlichen Begriffen zugeordnet werden. Dabei ist es eine Frage der Begriffsbildung, ob ausreichend präzise Begriffe verfügbar sind, denen die Ereignisse zugeordnet werden können. Aus § 1566 II BGB ergibt sich, daß eine Zeit des Getrenntlebens von 3 Jahren eine notwendige Bedingung für eine streitige Scheidung darstellt. Es werden also Angaben darüber benötigt, welche Daten bisher für diese Zeit angenommen wurden. Die gesuchten Angaben ließen sich über folgende Relationen beschreiben:
– *Getrenntleben*(*t*, *M*, *F*)
– *Ende*(*x*, *t*)
– *Beginn*(*y*, *t*)

Wobei *t* den Zeit**raum** des Getrenntlebens und x und y den Anfangs- bzw. Endzeitpunkt des Getrenntlebens bezeichnen sollen. Diese Betrachtung erlaubt nun die Einordnung des gegnerischen Vortrags: Er behauptet, der Urlaub *U* habe am Zeitpunkt *T* begonnen und die Trennung (**X**) beendet: So lassen sich die in dem Schriftsatz behaupteten Fakten mit Hilfe folgender Relationen formulieren:
– *Getrenntleben*(*X*, *M*, *F*)
– *Ende*(*T*, *X*)

[196] RGZ 160, 280, 285.

Diese Kategorisierung wird jedoch nicht allen Aspekten des Vorbringens *Urlaub* gerecht. So kann gerade die zeitliche Begrenzung des Urlaubs ein Beispiel für einen befristeten Versöhnungsversuch gem. § 1567 II BGB darstellen[197]. Man könnte also eine entsprechende Begriffsbildung proklamieren die zu folgender Relation führen würde:

– *Gemeinsamer_Urlaub*(*U, M, F*)

– *Beginn*(*T, U*)

Es liegt nun im Interesse der Partei, die die Scheidung will, insbesondere die Relation *Ende*(*T, X*) abzustreiten oder ihr die Relevanz zu nehmen. Dies gelingt dadurch, daß *U* und somit *X* gänzlich bestritten wird oder ein wesentlich früherer Zeitpunkt für *X* behauptet wird oder daß folgende neue Relation *Versöhnungsversuch*(*U, M, F*) behauptet wird. Dies kann unter Berufung auf die Relation *Gemeinsamer_Urlaub*(*U, M, F*) und die Regel aus der Rechtsprechung oder etwa eine Verallgemeinerung diesbezüglich aus dem *Palandt*[198] $Gemeinsamer\_Urlaub(U, M, F) \rightarrow Versöhnungsversuch(U, M, F)$ geschehen.

Der Gegner wird im nächsten Schriftsatz die Relation *Echte_Versöhnung*(*U, M, F*) entgegnen und auf besondere Umstände hinweisen, die gerade bei diesem Urlaub vorgelegen haben. Hierzu wird er auch die Regel um eine ihm genehme Ausnahme erweitern und mit einer Reihe von weiteren Regeln seine Position verteidigen:

$$Gemeinsamer\_Urlaub(U, M, F) \wedge \neg Besondere\_Umstände(x, U) \rightarrow Versöhnungsversuch(U, M, F)$$ [199]

$$Gemeinsamer\_Urlaub(U, M, F) \wedge Besondere\_Umstände(x, U) \rightarrow Echte\_Versöhnung(U, M, F)$$

$$Echte\_Versöhnung(U, M, F) \times Versöhnungsversuch(U, M, F)$$

Die praktische Rechtsanwendung vor einer juristischen Entscheidung besteht also aus einem ständigen Wechselspiel aus Actio und Reactio der Parteien. Die Actio besteht meist in der Behauptung von Fakten, die Reactio im Bestreiten des gegnerischen Vorbringens. Zur Formulierung einer schlagkräftigen Reaktion müssen jeweils alle Fakten aus den vorangegangenen Vorgängen präsent sein. Tatsachen können dabei dadurch bestritten werden, daß die Existenz bestimmter Objekte hier etwa des Urlaubs völlig abgestritten wird. Es kann aber auch nur der Wert eines Objekts, hier etwa das Datum des Urlaubsbeginns oder -endes in Frage gestellt werden. Letztendlich können die Eigenschaften eines Objekts bezweifelt werden, so etwa dadurch, daß der Urlaub von einem als Versöhnungsversuch vom anderen aber als echte Versöhnung bezeichnet wird. Welche der Möglichkeiten eingeschlagen wird, hängt von der Beweislage sowie von den vorangegangenen Einlassungen ab.

e) Klageerhebung und schriftliches Vorverfahren

Mit der Erhebung der Klage bzw. der Antragstellung bei der Ehesache beginnt ein förmliches Verfahren. Wesentliches Kommunikationsmedium der Parteien untereinander, aber auch zwischen Gericht und Prozeßbeteiligten ist der Schriftsatz. Der notwendige Inhalt von Schriftsätzen ergibt sich nun aus den §§ 130, 253 ZPO. Für Scheidungssachen, die Hauptgegenstand der Erörterungen sind, gelten einige Be-

---

[197] RGZ 160, 280, 285.

[198] *Palandt, Diedrichsen* § 1567 RdNr. 3. Auch ein Kommentar formuliert eine **Regel**. Die Relevanz dieser Regeln mag dabei streitbar sein. Der Praktiker jedenfalls wird sicherlich gerne den Palandt zitieren.

[199] Die Bildung der Prädikate ist lediglich exemplarisch und kann nicht als abgeschlossen angesehen werden.

sonderheiten in den §§ 622, 630 ZPO. Hinzu kommt, daß im Gegensatz zu anderen Zivilverfahren in Ehesachen der Untersuchungsgrundsatz gilt (§ 616 ZPO).

Es ist zu unterscheiden zwischen Angaben, die für die formal korrekte Fortführung und Beendigung des Verfahrens notwendig sind, und Angaben zum Sach- und Streitstand. Erstere sind vor allem die Personalangaben der beteiligten Personen einschließlich der Zustelladresse (§ 130 ZPO) sowie die Anträge (§ 253 II ZPO). Diese Angaben sind auch notwendig zur Abfassung eines vollstreckungsfähigen Titels. Jedenfalls in der Klageschrift sind diese Angaben zwingend erforderlich. Regelmäßig nur geboten (Sollvorschriften) sind hingegen Angaben sowohl über die tatsächlichen Umstände als auch über Beweismittel. Ebenso nicht zwingend sind Rechtsausführungen oder Ausführungen zum Vorbringen des Gegners. Im Regelfall werden dennoch alle Angaben mit mehr oder weniger Ausführlichkeit in einem Schriftsatz auftauchen müssen.

aa) Formalien

§ 130 ZPO. *Die vorbereitenden Schriftsätze sollen enthalten:*
1. die Bezeichnung der Parteien und ihrer gesetzlichen Vertreter nach Namen, Stand oder Gewerbe, Wohnort und Parteistellung; die Bezeichnung des Gerichts und des Streitgegenstandes; die Zahl der Anlagen;
...

§ 253 ZPO ...
(2) Die Klageschrift muß enthalten:
1. die Bezeichnung der Parteien und des Gerichts;
2. die bestimmte Angabe des Gegenstandes und des Grundes des erhobenen Anspruchs, sowie einen bestimmten Antrag.
...
(4) Außerdem sind die allgemeinen Vorschriften über die vorbereitenden Schriftsätze auch auf die Klageschrift anzuwenden.

Die §§ 130 Nr. 1 ZPO sowie §53 I ZPO schreiben diejenigen Angaben fest, die jeder Schriftsatz gleichermaßen enthalten muß oder soll. Es sind Fakten, die prinzipiell in jedem Verfahren einheitlich zu erheben sind. So gibt es immer Parteien, diese haben einen Wohnsitz, es gibt einen Streitgegenstand etc. Die Formulierung eines Schriftsatzes ist ein eigenständiger Vorgang. Zur Optimierung des Datenflusses muß überprüft werden, wie die benötigten Fakten von vorangegangenen Vorgängen übernommen werden können.

Im Schriftsatz müssen die *Parteien* angegeben sein. Der Begriff ist neutral und meint sowohl die Antragsteller (im normalen Verfahren die Kläger) als auch die Antragsgegners (bzw. Beklagten). Das folgende Flußdiagramm versucht, den Vorgang der Abfrage der Informationen über die *Parteien* in dieser allgemeinen Form darzustellen, die der gesetzlichen Formulierung entspricht:

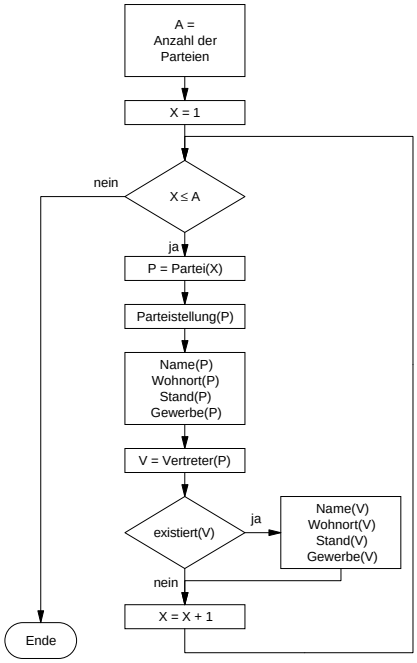


Abbildung 3: Flußdiagramm zur Erfassung der Parteien in Anlehnung an § 130 Nr. 1 ZPO

Das dargestellte Verfahren erfaßt alle Parteien, indem zunächst die Anzahl der Parteien abgefragt wird und danach sukzessive die Angaben zu den einzelnen Parteien inklusive ihrer Vertreter. In der Praxis wird dieses Verfahren jedoch zu Schwierigkeiten führen. So wird die Information über eine Person, ob sie **Partei** ist, als solche nicht explizit erfaßt. Im Aufnahmebogen sind beispielsweise keine derartigen Angaben vorgesehen. Vielmehr wurde lediglich die Relation $Antragsteller(a, v)$ erfaßt. Es bedarf also entweder weiterer Regeln, um aus diesen Angaben die gesuchten Parteiangaben zu ermitteln, oder einer anderen Vorgehensweise.

Eine Regel könnte etwa $Antragsteller(as, verf) \rightarrow Partei(as, verf)$lauten. Die Regel für die Feststellung der Parteifunktion des Ehepartners bedürfte bereits eines Kettenschlusses:

$$Ehesache(verf) \wedge (Ehemann(pers, ehe) \vee Ehefrau(pers, ehe)) \wedge \neg$$
$$Antragsteller(pers, verf) \rightarrow Antragsgegner(pers, verf)$$
$$Antragsgegner(ag, verf) \rightarrow Partei(ag, verf)$$

Einfacher ist das Verfahren, wenn man bei der Abfrage der Daten aus den vorangegangenen Vorgängen nicht auf das allgemeine Merkmal *Partei*, sondern direkt auf die Parteirolle (Antragsteller, Antragsgegner etc.) abstellt. Zwar muß in diesem Fall das obige Schema in ähnlicher Form für jede Rolle erneut durchlaufen werden, die Umsetzung der typischerweise erfaßten Rollen auf die Parteirolle und zurück entfällt jedoch. Hierfür spricht auch die Praxis, bei der Formulierung des Schriftsatzes nach Rollen zu unterscheiden und dies auch durch die Anordnung der Daten auf dem Papier zu verdeutlichen. Dagegen spricht lediglich der höchst unwahrscheinliche Fall, daß der Numerus Clausus an Parteirollen erweitert wird. Ein Verfahren, daß auf diese Rollen abstellt, müßte dann rechtzeitig angepaßt werden.

Die zuletzt beschriebenen Regeln zur Definition der **Partei** beschränken sich auf Familiensachen. Diese sind von ihrer Konstellation her sehr stark schematisiert und

deshalb mit noch einfacheren Verfahren zu handhaben. Typischerweise gibt es in Ehesachen lediglich zwei Parteien, die Eheleute. Beide besitzen ebenfalls im Regelfall keine **gesetzlichen** Vertreter. Unter dieser Annahme und der Prämisse, daß alle Angaben mit dem Aufnahmebogen erfaßt wurden, ist die Ermittlung der Parteiangaben wesentlich einfacher:

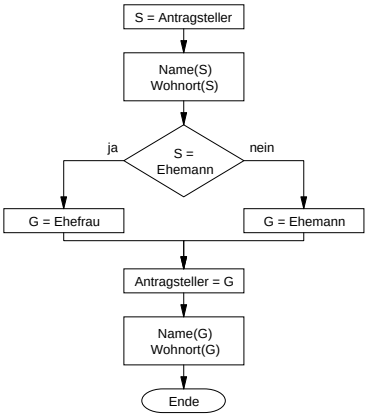


Abbildung 4: Flußdiagramm zur Feststellung der Parteien im Scheidungsverfahren unter Annahme der Angaben aus dem Aufnahmebogen

bb) Schriftsatzmuster

Ähnliche Vereinfachungen sind auch für andere, stark systematisierte Verfahren denkbar. Sie ermöglichen die sehr einfache Behandlung stereotyp gelagerter Fallgruppen. Ein Ausfluß dieser Schematisierung sind sogenannte **(Prozeß-) Formulare**. Dabei handelt es sich um vorgefertigte Schriftsätze für typische rechtliche Szenarien. Ein derartiges Standardformular zeigt das folgende Abbild auszugsweise. Es baut auf einem Szenario der **Ehescheidung** auf, bei dem es als Parteien lediglich die Eheleute gibt. In diesem Fall ist der Mann der Antragsteller:

An das
Amtsgericht
- Familiengericht -

....., den 15.3.1986

Antrag

des .....① **-Antragsteller-**②

Verfahrensbevollmächtigter: Rechtsanwalt .....③

gegen

.....④ **-Antragsgegnerin-**②

wegen streitiger Ehescheidung⑥

vorläufiger Streitwert: DM .....⑦

Unter Vorlage besonderer Prozeßvollmacht bitten wir namens und im Auftrag des Antragstellers um Termin zur mündlichen Verhandlung und stellen für diesen gegen die Antragsgegnerin folgende Anträge:

I. Die am ..... vor dem Standesamt ....., Heiratsregister Nr...../....., geschlossene Ehe der Parteien wird geschieden.⑧

II. Die Kosten des Rechtsstreits werden gegeneinander aufgehoben.

Abbildung 5: Auszug aus einem Formular: *Antrag auf streitige Scheidung*[200] (kein Faksimile)

Die gepunkteten Felder bilden die Platzhalter für Fakten, die aus der Akte, also aus früheren **Vorgängen** zu ermitteln sind. Sie bilden die individuellen Bestandteile des ansonsten schematischen Sachverhalts.

Das anschließend dargestellte Formular des ähnlichen Inhalts ist in besonderem Maße auf eine methodisch optimale Umsetzung des Datenflusses ausgelegt. Im Gegensatz zu vielen anderen Formularsammlungen beschreiben die Formulare dieses Bandes die notwendigen Inhalte der einzelnen Felder durch abstrakte Begriffe. Diese können als Variablennamen verstanden werden, die im Fall der Wiederholung denselben Inhalt repräsentieren. Eine weitere Besonderheit ist die Unterteilung des Formulars in einzelne Textbausteine[201]. So wird erreicht, daß nicht nur Inhalte von Feldern in einem Formular variabel sind, sondern aufgrund von konkret bezeichneten Umständen des Sachverhalts das Formular als Ganzes umgestaltet werden kann.

Amtsgericht **- Familiengericht -** *** Ort *⓪** **ANTRAG (SCH)** **In Sachen** *** der/des Beruf, Name, Anschrift Mandant/in *①** **antragstellender Ehegatte②** **Verfahrensbevollmächtigte: * RA(e) Name, Ort *③** **gegen** *** der/des Beruf, Name, Anschrift Gegner/in *④** **gegnerischer Ehegatte②** *** Verfahrensbevollmächtigte: RA(e) Name, Ort *⑤**	**F** **Rubrum**
wegen streitiger Scheidung der Ehe,⑥ **vorläufiger Streitwert: DM * Betrag *⑦** **wird in * anliegender/nachzureichender * besonderer Vollmacht i.S. des § 609 ZPO beantragt,⑨**	**025** **Sachgebiet** **Streitwert** **Vollmacht**
Die vor dem Standesbeamten in * Ort der Eheschließung * am * Datum der Eheschließung * unter der Registernummer * Reg. Nr. * geschlossene Ehe der Parteien wird geschieden.⑧	**02** **Scheidungsantrag**
Begründung: **…**	**Begründung**

[200] *Strohm*, **Beck'sches Prozeßformularbuch**., S. 494.

[201] Vgl. *Bauer*/*Lichtner*, S. 96; *Buschbell*, Beck'sches Rechtsanwaltshandbuch (1. Aufl.), K V, RdNr. 12 bezeichnete die Texthandbücher als ersten *Weg, zur programmierten Textverarbeitung zu kommen*. In der Ausgabe 93/94 wird hingegen ein Ganztextsystem bevorzugt (K III RdNr. 95). *Herberger* (jur-pc 89, 116 ff.) hält hingegen die klassischen Textbausteine für nicht ausreichend differenziert und empfiehlt eine Programmierung mittels eines *Document Drafting Processors*.

Abbildung 6: Formularelemente: zum *Antrag auf streitige Scheidung*[202] (kein Faksimile)

Die einzelnen Punkte der Formulare sollen nun daraufhin untersucht werden, ob und wie die Fakten aus vorangegangenen Vorgängen, insbesondere aus dem Aufnahmebogen zu übernehmen sind.

aaa) Örtlich zuständiges Gericht(0)

Die Frage nach dem örtlich zuständigen Gericht (§ 606 ZPO) wird nur im zweiten Formular berücksichtigt. Die Frage ist zu präzisieren: *Ort des in der Sache örtlich zuständigen Gerichts*. Sie kann lediglich durch Rechtsanwendung gelöst werden. Die Antwort ergibt sich gemäß § 606 ZPO primär aus den Wohnorten der Ehegatten zum Zeitpunkt der Klageerhebung. Diese Wohnorte sind bereits aus dem Aufnahmebogen zu entnehmen, falls sich nicht im Laufe des Vorverfahrens eine Änderung ergeben hat.

Eine EDV–Unterstützung des Rechtsanwendungsvorgangs *Ermittlung des örtlich zuständigen Gerichts* ist mittels einer geeigneten Datenbank denkbar[203]. Der Gerichtsstand wird an dieser Stelle zum ersten Mal im Verfahren benötigt. Die Unterstützung des Vorgangs *Bestimmung des Gerichtsstandes* muß deshalb durch die Anfrage des Vorgangs *Schriftsatz erstellen* eingeleitet werden. Bisher konnten Fakten von anderen Vorgängen, etwa dem Aufnahmebogen, direkt übernommen werden. In diesem Fall muß der Vorgang nun wissen, wohin er seine Anfrage richten muß. Für die Adressierung einer solchen Anfrage gibt es im Prinzip zwei Möglichkeiten: zum einen kann eine **zentrale unabhängige Stelle** derartige Anfragen entgegen nehmen und weiterleiten. Zum anderen kann sich der Vorgang mit seinem Problem **direkt an die Instanz** wenden, die die Gerichtsstandsbestimmung vornimmt. Beide Varianten müssen bei einer technischen Umsetzung in Erwägung gezogen werden.

bbb) Persönliche Daten von Antragsteller und -gegner (1-4)

Unter Punkt 1 ist der Name (Vor– und Nachname), der Beruf, die Anschrift etc. des Antragstellers einzutragen. Dies entspricht den Erfordernissen an den Schriftsatz, wie sie zuvor ausführlich dargelegt wurden. Die Fakten können aus dem Aufnahmebogen für Familiensachen entnommen werden. Dies erfolgt über die Information im Aufnahmebogen, wer der Antragsteller ist. Dessen Personendaten sind dann zu erfragen. Die weiteren Angaben zu der identifizierten Person können ebenfalls aus dem Aufnahmebogen übernommen werden.

Punkt 3 (Daten des Rechtsanwalts) ist über die Organisation der Kanzlei zu klären. Hier werden individuell unterschiedliche Lösungen für einzelne Kanzleien zu suchen sein. Für ein auf das individuelle Verfahren bezogenes System stellen dies konstante Informationen dar. Bei der Systemgestaltung wird zu entscheiden sein, ob diese Informationen für jedes Mandat automatisiert dupliziert werden, ob sie bei jeder Anfrage aus einer externen Quelle entnommen werden oder ob man ganz auf ihre Behandlung verzichtet. In diesem Fall bleibt es den einzelnen Vorgängen selbst überlassen, die Information konstant mitzuführen oder an einer zentralen Stelle abzulegen. Das entspräche dem Vordruck von Briefköpfen und wäre aufgrund der wenigen zu erwartenden Änderungen der Daten keine ungewöhnliche Verfahrensweise.

Unter Punkt 4 ist der Name des **Antragsgegners** einzutragen. Diese Anfrage kann von keinem Vorgang unmittelbar beantwortet werden, da die Information, wer An-

---

[202] *Vespermann*, **Scheidungs- und Scheidungsverbundverfahren**, S. 8.

[203] Über die Probleme hierzu *Waltl*, CoR 1995, S. 188.

tragsgegner ist, beispielsweise im Aufnahmebogen explizit nie erfaßt wurde. Vielmehr ist ein Mechanismus nötig, der den Antragsgegner ermittelt. Dieser Mechanismus kann unmittelbar von dem Vorgang *Aufnahmebogen* übernommen werden oder in einem selbständigen Vorgang nachgeschaltet werden. Erstere Variante dürfte die wohl praktikablere darstellen. Hiernach werden bereits beim Ausfüllen des Aufnahmebogens alle zulässigen und naheliegenden Schlüsse auf weitere Fakten gezogen. Die Frage nach dem Geschlecht (Antragsteller oder Antragstellerin) läßt sich ebenfalls nur mit Hilfe einer Regel beantworten, die auf die bereits bekannten Beziehungen $Ehefrau(pers, ehe)$ und $Ehemann(pers, ehe)$ abstellt.

ccc) Antrag und Verfahren (5-9)

Streitige Ehescheidung (Punkt 5) ist die Voraussetzung für die Verwendung des Formulars. Hierauf basiert die Annahme über die Beteiligten sowie den Sachverhalt. Der Antrag (Punkt 8) ist für die streitige Scheidung stereotyp. Die genauen Angaben (§ 253 II Nr. 1 ZPO) werden benötigt, da die zu scheidende Ehe sich mit diesen Daten (Hochzeitsdatum, Standesamt sowie Heiratsregister) eindeutig **kennzeichnen**[204] läßt.

Die Angabe, ob eine Prozeßvollmacht **beiliegt** (Punkt 9), ist keiner vorangegangenen Eingabe unmittelbar zu entnehmen. Der Aufnahmebogen enthält allerdings die Information, ob eine Prozeßvollmacht vorliegt[205].

cc) Ausführungen zum Sachverhalt

Im Bereich der Begründung unterscheiden sich die Formulare erheblich. Das Formular des Prozeßformularbuchs folgt der eher typischen Form eines beispielhaften Schriftsatzes. Das Formular von *Vespermann* hingegen liefert unterschiedliche Formulierungsvorschläge für verschiedene Situationen.

aaa) Beweismittel

Die Anzahl der variablen Elemente innerhalb des Begründungstextes ist gering. Das Formular aus dem Beck'schen Formularbuch folgt zudem keiner rechten Systematik bei der Kennzeichnung der Variablen. So behandelt die Passage *mit der er inzwischen das am 1.1.1985 geborene Kind ... hat* den Namen des Kindes als variabel (...), dessen Geburtstag allerdings als Fixum. Die Sachverhaltsangaben sind bei diesem Formular ohnehin in einer Weise individuell fingiert, daß eine systematische Analyse nur wenig erfolgversprechend ist.

Wesentliches Merkmal der Texte ist jedoch die Belegung auch der allgemeinen Fakten durch entsprechende **Beweise**. So dient das Familienstammbuch als Beweis für die Daten der ehelichen Kinder. Die potentielle Bestreitbarkeit eines jeden Faktums[206] bringt die Notwendigkeit mit sich, jede behauptete Tatsache nach den Regeln der Beweislast auch beweisen zu müssen. Dies führt zu einem Wechsel von Tatsachenbehauptung und Beweisangebot im Schriftsatzaufbau[207]. Entsprechend der Monotonie der Sachverhalte im Familienrecht, sind auch die Einzelfakten und mithin die Beweismittel vielfach monoton. Hierzu gehören Familienstammbuch, Lohnbescheinigung, Versicherungsbescheinigungen etc. Folglich fehlen auch in dem ursprünglich behandelten Aufnahmebogen Hinweise auf Beweise. Sachverhalts-

---

[204] Vgl. S. 44.

[205] Vgl. S. 59.

[206] Vgl. S. 35.

[207] *Michel*, S. 161, *Thomas*/*Putzo* § 284 Anm. 1.

merkmale wie die Zerrüttung einer Ehe werden hingegen durch etwas individuellere Fakten und Beweise zu belegen sein.

Da letztlich die Verwaltung von Fakten für die juristische Praxis nur dann wirklich sinnvoll erscheint, wenn zu (möglicherweise) streitigen Fakten auch Beweise vorliegen, ist es wünschenswert, daß Beweise und Fakten eng aneinander gekoppelt werden. In vielen Fällen kann dies sehr einfach dadurch geschehen, daß bestimmten Fakten, wie dem Geburtsdatum, der Eheschließung etc. stereotype Beweise wie die Geburtsurkunde oder der Registereintrag gegenüberstehen. Ein Formular hat dann wie im Beispiel die Möglichkeit, dieses Beweismittel als Bestandteil des Formulartextes mit anzugeben.

Bei individuelleren Sachverhalten wird dies nicht ausreichen. Auch in den analysierten Formularen gibt es Passagen, in denen das Beweismittel als offenes Feld angelegt ist (z.B. Beweismittel zum sogenannten *Sündenregister*). Daraus ergibt sich die Forderung, daß zu jeder Behauptung eines Faktums auch die Angaben von einem oder mehreren Beweismitteln abrufbar sein müssen. Ein Vorgang muß in der Lage sein, nicht nur die Tatsachen, sondern auch deren Beweise darzulegen. Nur so ist die optimale EDV–Unterstützung für die Schriftsatzgenerierung möglich. Ebenso sind für eine bestmögliche Anwaltsunterstützung Vorgänge zur Analyse der Beweislage und Erkennung von Beweisnot denkbar. Der Anwalt muß bei der Einarbeitung des Gegenvortrags in sein System entsprechende Hinweise erhalten. Subsumtionsunterstützende Systeme können auf die Beweislage eingehen und wasserdichte Auswege suchen.

bbb) Variable Formulierungen

Der gesamte Inhalt des Schriftsatzes wird durch den tatsächlichen Sachverhalt gesteuert. Dies äußert sich praktisch zunächst in der Wahl des entsprechenden Formulars. Die unterschiedlichen Formulare ergeben sich im Fall der Scheidung aus den §§ 1565 ff. BGB. Nach § 1565 II BGB kann eine Ehe bei einem Getrenntleben von unter einem Jahr nur **streitig** geschieden werden, wobei den Antragsteller eine erhebliche Begründungslast trifft. Leben die Ehegatten zwischen ein und drei Jahren getrennt, so ist zwischen einer **einverständigen** (§ 1566 I BGB) und einer streitigen Scheidung zu unterscheiden. Bei letzterer trifft wiederum den Antragsteller die Pflicht, eine Zerrüttung gem. § 1565 I BGB zu begründen und zu beweisen. Im Fall eines Getrenntlebens von mehr als drei Jahren muß der Antragsteller bei Bestreiten lediglich die Dauer des Getrenntlebens (§ 1565 II BGB) selbst beweisen. Für jedes dieser Szenarien ist zunächst ein eigenes Grundformular verfügbar.

Im Bereich des Schriftsatzkopfes ergeben sich dann geringfügige Unterschiede, wenn die Parteien bei einer (vermutlich einverständigen) Scheidung lediglich einen Rechtsanwalt benennen. Die übrigen Unterschiede liegen im Bereich der Begründung. Die notwendigen Fakten für die Wahl des Formulars können aus dem Aufnahmebogen entnommen werden. Die Frage der Streitigkeit ergibt sich aus den expliziten Punkten *Einverständige Scheidung* und *Streitige Scheidung* im Aufnahmebogen, die sich prinzipiell ausschließen. Der Trennungszeitraum ergibt sich aus dem *Trennungszeitpunkt* und dem Tagesdatum[208].

Eine interessante Situation ergibt sich, wenn zwei Ehegatten noch vor Ablauf des Pflichtjahres einverständlich eine Scheidung wünschen. In diesem Falle muß das Vorgehen zunächst auf eine streitige Scheidung wegen unzumutbarer Härte gerichtet sein. Der Antragsgegner darf dabei die vorgetragenen Tatsachen nicht ernsthaft

[208] Vgl. S. 58.

bestreiten. Verstreicht aufgrund irgendwelcher Umstände bereits im vorgerichtlichen Verfahren die Jahresfrist, so wird die Scheidung nun einfacher als einverständliche Scheidung beantragt. Ein Vorgang sollte so ausgestaltet sein, daß er derartige Spitzfindigkeiten erkennt. Er müßte vor Ablauf der Jahresfrist auf diese Möglichkeiten hinweisen und sie explizit ermöglichen.

Das Formular im Diktat– und Arbeitsbuch ermöglicht neben der Grundauswahl des *Formulars*[209] den Austausch bestimmter Textpassagen in Abhängigkeit von bestimmten Fakten. So wird beispielsweise die Begründung der örtlichen Zuständigkeit des Gerichtes danach variiert, welcher Ehegatte und welche Kinder im Bezirk des Gerichtes wohnen. Diese Frage kann automatisch nur unter Zuhilfenahme einer Gerichtsdatenbank erfolgen, die die Adressen der Parteien und Kinder mit entsprechend detaillierten Angaben aus der Datenbank abgleicht.

Eine weitere Differenzierung ergibt sich aus § 622 ZPO. Die Frage nach der Anzahl der gemeinschaftlichen Kinder wird durch ein bereits dargestelltes Zählen[210] der im Fragebogen angegebenen Kinder zu lösen sein. Für die Auswahl wird also auf die geforderte systeminterne Zählfunktion zurückzugreifen sein.

ccc) Offene Textpassagen

Selbst wenn wie bei den Textbausteinen von *Vespermann* ein sehr hohes Maß an Systematisierung erreicht werden kann, so ergeben sich doch gerade bei gedruckten Schriftsatzmustern immer wieder Grenzen dort, wo es auf individuelle Begründungen ankommt. Das klassische Schriftsatzmuster behilft sich hier mit der Formulierung eines Beispiels. Dieses Verfahren ist einer Automatisierung der Schriftsatzerstellung nicht zugänglich. *Vespermann* verwendet hingegen ein eigenes Feld mit der ausführlichen Beschreibung des hier geforderten Inhalts. Ein geschicktes Tutorensystem kann hier sicherlich unter intensiver Verarbeitung der kasuistischen Rechtsprechung sowie exemplarischer Fälle aus Lehre und Praxis den Anwender weiter unterstützen. Für eine Verwendung der dabei erhobenen Erkenntnisse über den einen Schriftsatz hinaus, müßten die einzelnen Gesichtspunkte und ihre Wirkung systematisiert werden. Es ist nicht das Ziel dieser Arbeit, derartige Systeme zu entwickeln oder vorzustellen[211]. Indessen ist für die Zukunft zu erwarten, daß die Erfassungstiefe von Fakten noch weit über die Systematik des reinen Gesetzestextes hinausgeht.

Die Fakten, die für die analysierten Schriftsätze zur Antragstellung benötigt werden, lassen sich auch aus dem Fragebogen entnehmen. Für ein zukunftssicheres System wird es jedoch notwendig sein, Fakten auch außerhalb der hier abgebildeten Gesetzessystematik zu behandeln. Je weiter die Generierung eines Textes durch ein System unterstützt wird, desto wichtiger ist, daß für die behaupteten Fakten auch Beweismittel dargestellt werden.

5. Abschließende Darstellung der juristischen Analyse

Es ist nun denkbar, eine Reihe von weiteren in der juristischen Praxis relevanten Vorgänge zu prüfen. Etwa könnte die Praxis der *Relation* dargestellt werden oder die Aufnahme der ersten Replik etc. Praktisch sind jedoch die wesentlichen Gesichtspunkte im vorangegangenen Teil angerissen worden. Hierzu gehören insbesondere die prädikatenlogischen Ansätze bei der Algorithmisierung rechtlicher Entscheidungsfindung sowie

---

[209] In einer EDV-Unterstützung würde es eine derartige Vorauswahl gar nicht geben, sondern die unterschiedlichen Alternativen würden in Abhängigkeit der geschilderten Faktoren automatisch gewählt.

[210] S. 57.

[211] Einen Überblick vermittelt *Kowalewski*, *Schnneeberger*, **KOKON**, Informatik Fachberichte 127, 1986.

die praktisch eingesetzten weitgehend systematischen Arbeitshilfen in der Anwaltspraxis. Erstere zeigte ein denkbares Konzept der Repräsentation von Fakten auf. Letztere Analyse eröffnete zum einen den Blick für die bereits geleisteten Vorarbeiten und verdeutlichte die praktische Relevanz von unklaren Tatbeständen.

B. Datenaustausch und Retrieval in der EDV

Die bisherigen Ausführungen beschränkten sich auf eine vornehmlich juristische Betrachtung des Problems des Datenaustauschs. Die Abhandlung wäre jedoch nicht vollständig, wenn nicht auch auf die Ansätze der Informatik eingegangen würde. Der folgende Abschnitt wird die hier wesentlichen Gesichtspunkte des Datenaustauschs aus Sicht der EDV darstellen. Die Darstellung ist dabei auf die Grundlagen beschränkt, um das Verständnis des juristischen Lesers zu erlangen.

In den technischen Disziplinen ist die Problematik, zwischen **inhomogenen Systemen** Daten auszutauschen, ein bekanntes und ständig im Fluß befindliches Thema. Man bezeichnet derartige Systemübergänge als *Schnittstellen* bzw. ihre Beschreibungen als Schnittstellendefinitionen. Derartige Schnittstellen existieren auf ganz unterschiedlichen Niveaus. Sie reichen von der Realisierung der physikalischen Verbindung zweier Geräte bis hin zur softwaretechnischen Umsetzung komplexer Datenaustauschprotokolle.

Diese Arbeit beschäftigt sich mit einer Schnittstelle auf einer sehr abstrakten Ebene[212]. Es wird ein Datenaustauschverfahren zunächst auf einer beliebig gearteten Hardwareschnittstelle und einem softwaretechnischen Basisprotokoll entwickelt. Es ist vornehmlich für den Transport der Inhalte verantwortlich. Es werden hier **zunächst** weder Hardwareschnittstellen noch technische Schnittstellen zwischen einzelnen Programmen oder Programmodulen besprochen. Erst im Rahmen der Realisierung wird zu entscheiden sein, welche Hard– und Softwareschnittstellen als Grundlage des Datenaustauschs zu wählen sind.

1. Moderne Konzepte des Datenaustauschs

Dem erheblichen Wandel in der Hardwarelandschaft hat sich in der jüngsten Zeit auch die Softwarearchitektur angepaßt. Dies ist insbesondere im Bereich der Datenbanken erkennbar. Schlagworte wie CAD[213]/CAM[214] und CIM[215] beruhen auf diesem Wandel. Vor allem die für die **sogenannte Client-Server–Architektur** entwickelten Programmiersprachen der 4. und 5. Generation (kurz 4GL und 5GL) stehen im Mittelpunkt der Diskussion. Sie sind auch an dieser Stelle von Interesse, da sie die modernsten im Einsatz befindlichen Konzepte für den Datenaustausch, aber auch für die Datenintegration zur Verfügung stellen.

Ausgangspunkt für die genannten Konzepte ist die immer größere Leistungsfähigkeit der Endgeräte am Arbeitsplatz des Anwenders. Während in stark Großrechner–basierten Systemen die Ein-/Ausgabegeräte meistens keinerlei Intelligenz benötigen

---

[212] Das OSI–Schnittstellenmodell (DIN/ISO 7498) geht von 7 Ebenen aus. Die unterste Ebene beschäftigt sich mit dem Physikalischen Transport. Es folgen diverse Ebenen zur Sicherung und Steuerung. Die hier beschriebene Schnittstelle ist wohl der Ebene 6 zuzuordnen, die sich mit der Darstellung der Daten beschäftigt. Die Programmbeispiele betreffen die Anwendungsebene (7).

[213] Computer Aided Design: Entwicklung von Konstruktionsplänen mit dem Computer. Schnittstellen bestehen hier oft zur Werkzeug- und Bauteileverwaltung.

[214] Computer Aided Manufacturing: Computergesteuerte Herstellung von Werkzeugen und Steuerung von Produktionsprozessen.

[215] Computer Integrated Manufacturing: Integration von CAD und CAM zu einer vollautomatischen Produktion ausgehend vom Bauplan.

und der zentrale Rechner die gesamte Datenein- und -ausgabe kontrolliert, besitzen moderne PCs jedenfalls in Teilbereichen eine größere Leistungsfähigkeit als die Hosts[216], auf die sie zugreifen.

Klassische Datenbankanwendungen auf **Großrechnersystemen** beruhen darauf, daß die Programme auf dem Host neben der eigentlichen Datenverwaltung auch den gesamten Dialog mit oft mehreren Endnutzern gleichzeitig übernehmen. Dabei gibt der Endnutzer seine Suchanfrage in einer Form ein, die regelmäßig einen Kompromiß zwischen Lesbarkeit für den Menschen und die Maschine darstellt. Die Ausgabe erfolgt in Listen oder Volltexten und wird so gestaltet, daß sie auch noch auf einem endlosen Papierstreifen etwa eines Fernschreibers lesbar ist. Die Schnittstelle zwischen den beiden Geräten (Host und Terminal) besitzt neben einer einfachen Hardwareebene (einfaches serielles Textprotokoll) auch nur eine minimale Softwareschicht. Das Terminal fungiert selbst als Schnittstelle, nämlich zwischen dem Endanwender und dem Host. Jeder Tastendruck des Anwenders wird vom Terminal unmittelbar an den Großrechner weitergeleitet. Umgekehrt wird jeder Buchstabe, den der Benutzer am Bildschirm oder Fernschreiber sieht, direkt vom Host dort plaziert. Dies gilt sogar für die Zeichen, die der Benutzer über Tastatur eingibt. (Sie werden also von der Tastatur zum Host geschickt und von dort wieder zurück an den Bildschirm.) Ein typisches Beispiel für diese Architektur stellt die Datenbank **juris** dar, die auf einem SNI BS2000 Host arbeitet und noch heute mit einfachsten Endgeräten bedienbar ist.

Aufgrund des Vormarsches des PCs[217] als intelligentem **Stand-Allone-Computer** ist die Datenhaltung auch großer Datenmengen am Arbeitsplatz dessen möglich, der die Daten im wesentlichen benötigt[218]. Durch eine auf die Hardware des PCs maßgeschneiderte Software bedarf es keiner engen Schnittstellen, wie sie für heterogene Ein-/Ausgabegeräte zu definieren sind. Als Folge ist die Benutzeroberfläche dieser Programme oft sehr komfortabel gestaltet. Der Benutzer bemerkt von den technischen Abläufen nur wenig. Ein sehr typisches Beispiel für diese Softwareform ist **F&A** von **Symantec** oder etwa **Microsoft Works**. In beiden Programmen wird eine Datenbank mit ihrer gesamten Struktur simultan mit der Herstellung der Eingabemaske, also der Benutzeroberfläche generiert. Die Oberfläche des Endanwenders und die Oberfläche für das Datenbankdesign sind also unmittelbar aneinander gekoppelt.

Ein erheblicher Nachteil dieser lokalen Datenhaltung ist die starke Dezentralisierung der Daten. Hierdurch stehen vor allem größere Unternehmen oder Institutionen heute vor einem erheblichen Wildwuchs in ihrer Datenhaltung. Daten werden oft redundant und in unterschiedlicher Qualität vorgehalten. Nicht selten fehlt der Überblick über die überhaupt vorhandenen Datenbanken sowie die verwendeten Strukturen. Hinzu kommt, daß auch die modernen hochleistungsfähigen Rechner durch die immer größer werdenden Anforderungen an die Benutzeroberflächen sowie die große Leistungsfähigkeit typischer lokaler Anwendungen (z.B. Textverarbeitung oder Tabellenkalkulation) stark ausgelastet sind. Auch bieten moderne

---

[216] Wörtlich *Gastgeber*. Das ist meist ein Großrechner, auf den von anderen Arbeitsplätzen über ein Netzwerk oder Datenfernübertragung zugegriffen wird.

[217] *Personalcomputer*. Computer mit einer eigenen sehr leistungsfähigen Recheneinheit. (Zum Zeitpunkt dieser Arbeit ist ein sehr leistungsfähiger IBM–kompatibler PC mit einer INTEL Pentium Pro CPU (Zentrale Recheneinheit) mit 200 MHz Taktfrequenz, 16-32 MB Arbeitsspeicher und einer Festplatte mit von 1-2 GB ausgestattet.).

[218] Zum Einsatz von PCs im juristischen Bereich *Hoffmann*, S. 49 ff.; zu Datenbanken insbesondere S. 67 ff.

Großrechner weitaus höhere Kapazitäten an Massenspeichern als PCs. Oft sind ihre Datensicherungssysteme entsprechend leistungsfähiger. Letztendlich ist eine stark steigende Vernetzung von PCs untereinander und mit Rechnern der mittleren[219] und Großdatentechnik zu beobachten.

Hieraus entwickelte sich ein arbeitsteiliges Konzept, das die Datensuche und Datenpflege von der Benutzeroberfläche gänzlich abkoppelt. Dieses als **Client-Server–Architektur**[220] bezeichnete Konzept basiert darauf, daß ein Prozeß auf dem Server Aufträge abarbeitet, die ihm von anderen Prozessen, den sogenannten Clients, übermittelt werden. Hierbei kann es sich z.B. um Aufgaben der Datenbankpflege oder Entwicklung handeln. Vor allem aber werden dies Datenanfragen und Dateneingaben von Endnutzern sein. Dabei müssen die beiden Prozesse (Client und Server) nicht zwangsläufig auf unterschiedlichen Geräten ablaufen. Diese Architektur wird auch rein softwaretechnisch bei parallel laufenden Prozessen auf einem Computer angewendet.

Die Kommunikation der Prozesse erfolgt über eine einheitliche Schnittstelle. Oberste Ebene dieser Schnittstelle ist die **Datenbanksprache**. Aufgrund der starken Abstraktion von Benutzeroberfläche und Maschine müssen weder die Arbeitsanweisungen, die der Client dem Server übermittelt, noch die zurückgeschickten Daten für den Menschen lesbar und verständlich sein. Vielmehr erfolgt die Umsetzung der Eingaben des Benutzers in die Datenbanksprache sowie die Darstellung der Daten beim Benutzer erst nach einer Verarbeitung durch den Clientprozeß. Diese Architektur ermöglicht eine zentrale, hardwareunabhängige Datenhaltung unter Wahrung der individuellen Bedürfnisse des jeweiligen Endbenutzers[221].

Das Interesse dieser Arbeit richtet sich zum einen auf die Architektur der Datenaustauschvorgänge, die sich aus dem Client-Server-Prinzip ergibt. Sie stellt eine denkbare Variante der Organisation auch des juristischen Datenflusses dar. Zum anderen sind die in diesem Zusammenhang bereits entwickelten Abfragesprachen von hohem Interesse. Sie können bereits Hinweise auf die Entwicklung einer spezifisch juristischen Abfragetechnik zulassen. Sie ermöglichen es, den Stand von Forschung und Praxis aus technischer Sicht darzulegen und auf seine Brauchbarkeit bezüglich des hier angestrebten Systems hin zu untersuchen.

2. Datenflußarchitekturen, insbesondere das Client-Server-Prinzip

Wie bereits dargestellt, beruht die Client-Server-Architektur darauf, daß neben den **datenverarbeitenden** Vorgängen - sie werden hier Prozesse genannt -(den Clients) ein weiterer selbständiger Prozeß (der Server) für die Datenhaltung zuständig ist.

Die **Clientprozesse** sind für die Verarbeitung der Daten verantwortlich. Sie stellen die Daten am Bildschirm zusammen, erledigen die Druckausgabe oder verarbeiten sie direkt weiter. So können einzelne oder Serienbriefe ausgegeben werden, Termine optisch aufbereitet oder Statistiken in Form von Graphiken dargestellt werden. Umgekehrt werden die Daten etwa in Masken oder Tabellen innerhalb eines Clientprozesses eingegeben. Der **Serverprozeß** übernimmt den gesamten Zugriff auf den Massenspeicher (also vorwiegend auf die Festplatte), insbesondere die Suchoperationen, aber auch klassische Pflegeaufgaben wie Ändern von Tabellen, Optimieren

---

[219] Unter mittlerer Datentechnik versteht man meist sehr leistungsfähige Computer, die mit dem Betriebssystem UNIX ausgestattet sind.

[220] S. hierzu auch *Neske*, CoR 3/94, S. 146 ff.

[221] Zu den Anforderungen an EDV-Unterstützung zumindest in der Justiz *Juristar*, 7.2.

der Daten oder Mitführen von Transaktionsprotokollen[222]. Er erlaubt die Datensicherung und überprüft bei allen Aktionen die Daten auf Konsistenz. Aufträge, die diese Konsistenz verletzen, werden vom Server zurückgewiesen.

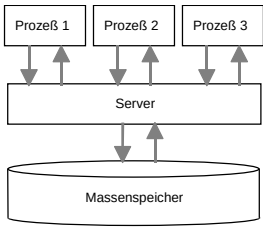


Abbildung 7: Schematische Darstellung der Client-Server-Architektur.

Dabei werden nach dem Grundgedanken der Architektur die einzelnen Prozesse meist auch auf unterschiedlichen, physikalisch verbundenen (vernetzten) Geräten betrieben. Als Alternative zum Client-Server-Prinzip bietet sich auf der einen Seite der klassische ganzheitliche Ansatz und auf der anderen Seite eine gänzlich verteilte Architektur an.

Im ganzheitlichen Ansatz werden alle Datenzugriffsoperationen von dem Prozeß vorgenommen, der die Daten benötigt. Hierzu gibt es entweder ein großes Programm, das sämtliche Daten bearbeitet und alle Wünsche der Endanwender (inklusive des Systemoperators) befriedigt, oder mehrere einzelne Prozesse erledigen bestimmte Arbeitsteile. Dabei greift jeder Prozeß selbständig auf alle benötigten Daten zu.

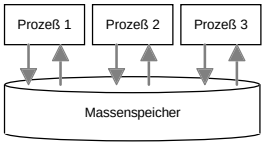


Abbildung 8: Schematische Darstellung einer ganzheitlichen Architektur mit direktem Zugriff eines jeden Prozesses auf den gesamten Datenbestand.

Der verteilte Ansatz geht davon aus, daß unterschiedliche Prozesse unterschiedliche Daten benötigen und für die Datenhaltung selbst verantwortlich sind. Benötigt ein Prozeß *A* einen Teil der Daten, die von Prozeß *B* verwaltet werden, so richtet er eine Anfrage unmittelbar an *B* und erhält von diesem die Daten übermittelt. Diese Architektur wurde in den vorangegangenen Analysen stillschweigend angenommen, wenn vom Austausch von Fakten zwischen Vorgängen die Rede war.

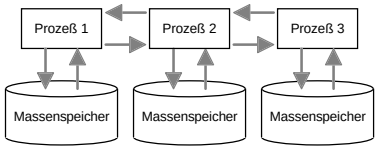


Abbildung 9: Schematische Darstellung einer verteilten Architektur, bei der jeder Prozeß für die Haltung seiner (primären) Daten zuständig ist.

Eine interessante Mischlösung wird derzeit von einigen Softwarehäusern angeboten. Die wohl gängigste Variante ist ODBC[223] von *Microsoft*. Das System stellt jedem beliebigen Prozeß einen Satz von Funktionen (sogenannte API[224]) zur Verfügung,

---

[222] Transaktionsprotokolle erlauben es über eine gewisse Zeit einen älteren Zustand herzustellen, falls Fehler in einem Vorgang aufgetreten sind.

[223] Open Database Connectivity.

[224] Applications Programmers Interface.

mit der er auf verteilte Daten zugreifen kann. Als interne Datenbanksprache findet ein SQL-Dialekt Anwendung. Die Besonderheit liegt darin, daß es für den Prozeß, der auf Daten zugreifen muß, völlig unerheblich ist, wo die Datenquelle liegt und ob weitere Stationen zum Erreichen der Daten zwischengeschaltet werden. Er kann also gleichzeitig quasi direkt auf eine eigene Datenbank zugreifen, aber auch als Client eines Client–Server–Systems auftreten. Besonders interessant dabei ist, daß die Daten unterschiedlichster Quellen (relational) miteinander verknüpft werden können. Abbildung 10 stellt diese Architektur beispielhaft graphisch dar:

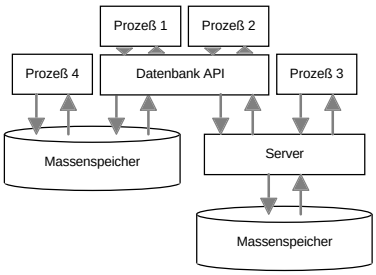


Abbildung 10: Schematische Darstellung einer Architektur mit datenbankunabhängiger Datenbankmaschine.

Wie die Abbildung verdeutlicht, können neben Prozessen, die sich der Datenbank API bedienen (Prozesse 1 und 2) auch Prozesse über die herkömmlichen Zugriffswege auf die Daten zugreifen (Prozesse 3 und 4).

Nicht zwangsläufig, aber in vielen Fällen geht mit der Software- auch eine entsprechende Hardwarearchitektur einher. So wird die Client-Server-Architektur vor allem in inhomogenen Systemen eingesetzt, bei denen die Datenhaltung auf zentralen Rechnern der mittleren und Großdatentechnik erfolgt. Die Verarbeitung erfolgt jedoch auf mit diesen vernetzten intelligenten Terminals, meist PCs. Ganzheitliche Lösungen finden sich zum einen in reinen Hostumgebungen (mit Alphaterminals oder wenig intelligenten graphischen Endgeräten), aber auch auf lokalen PCs oder Workstations . Verteilte Datenbanksysteme wiederum finden sich oft auf Netzwerken mit weitgehend einheitlich leistungsfähigen Endgeräten ohne einen zentralen intelligenten Host[225], wobei die Netzwerkarchitektur heute oft auf Direktverbindungen der Geräte basiert (sogenannte Pear-To-Pear–Netze).

Eine Entscheidung für das eine oder andere Modell kann global nicht getroffen werden. So hat ein ganzheitlicher Ansatz den erheblichen Vorteil, daß die Daten verhältnismäßig sicher sind. Es können keine anderen Zugriffe vorgenommen werden als diejenigen, die im System originär vorgesehen sind. Zudem entfällt durch den Wegfall der Schnittstellen auch ein ganz erheblicher Normungsaufwand. Andererseits ist das System extrem inflexibel. Eine Anpassung auf die Bedürfnisse des Endnutzers oder auf neue technologische Entwicklungen ist nur sehr schwer möglich. Neue Aufgaben können nicht einfach hinzugefügt, sondern müssen direkt hineinprogrammiert werden. Das System ist demzufolge dann vorzuziehen, wenn es auf hohe Datensicherheit ankommt und die Arbeitsaufgaben der Endanwender sehr stereotyp sind.

Ein verteiltes System ermöglicht hingegen eine sehr flexible Gestaltung der Endanwendungen. Es ermöglicht nicht nur eine sehr schnelle Anpassung der Applikatio-

[225] Auch in Netzwerken mit zentralem Datenserver wird dieser vielfach nicht als Host mit eigener Intelligenz eingesetzt. Er liefert lediglich das Netzwerkbetriebssystem und stellt die Dateiverwaltung sicher.

nen an neue Aufgaben, sondern es können ebenso schnell die Datenstrukturen geändert werden, da jede Anwendung für ihre Datenhaltung selbst zuständig ist. Der Nachteil dieser Systeme ist ein erheblicher Normungsaufwand für die Kommunikation der einzelnen Objekte sowie die Gefahr des Datenwildwuchses. Wenn die Kommunikation, d.h. der Datenabgleich nicht sehr strengen Regeln unterliegt, ist die Gefahr von redundanten Daten insbesondere unterschiedlicher Qualität erheblich. Zudem ist es für den Austausch von Daten immer notwendig, den Prozeß zu kennen, der für die gesuchten Daten verantwortlich ist. Diese Architektur sollte deshalb insbesondere dann Anwendung finden, wenn es viele Aufgaben gibt, die aufgrund ihrer Eigenarten eine sehr individuelle Datenaufbereitung benötigen und bei denen die Daten in besonderem Maße mit einzelnen Aufgaben verknüpft sind.

Die Client-Server-Architektur ermöglicht zum einen eine verhältnismäßig sichere Datenhaltung, da die eigentlichen Datenzugriffe von einem einzigen Prozeß gesteuert werden. Hierdurch können auch Redundanzen leicht überprüft und vermieden werden. Zum anderen erlaubt sie eine recht flexible Einrichtung der datenverarbeitenden Anwendungen. Diese werden nämlich getrennt von der Datenhaltungsanwendung auf einer beliebigen Plattform realisiert. Datenhaltung und Datenverarbeitung können also technologisch auf unterschiedlichem Niveau arbeiten. Wenn auch die Nachteile der beiden anderen Systeme zunächst ausgeräumt zu sein scheinen, so handelt es sich selbstverständlich nur um einen Kompromiß. So ist in einem offenen System die Datensicherheit nie so zu optimieren, wie es in einer geschlossenen Anwendung möglich ist. Durch das Nadelöhr der Schnittstelle, die ja regelmäßig eine Art größten gemeinsamen Teiler darstellt, können Daten nicht unbedingt in der individuellen Form übermittelt werden, wie sie bestimmte Anwendungen benötigen würden. Dennoch erscheint diese Architektur für ein sehr flexibles zukunftsorientiertes System am besten geeignet.

Sowohl auf Ebene auf der Hardware als auch der Software sind Systeme zu unterscheiden, bei denen der Datenzugriff mit dem Verarbeitungsprozeß gekoppelt ist und solche, bei denen im Wege der Arbeitsteilung unterschiedliche Prozesse unterschiedliche Aufgaben übernehmen (Client-Server). Während die beiden autonomen die Nachteile des jeweils anderen Systems überkompensieren, scheint die Client-Server Lösung einen brauchbaren Kompromiß darzustellen.

3. Datenbankabfragesprachen

Die Datenbankabfragesprachen, die hier von Interesse sind, stellen in der Regel Schnittstellen zwischen zwei Maschinenprozessen dar, bei denen der eine Prozeß bei einem anderen Prozeß Daten anfordert. Ihre Entwicklung ist zum einen stark von den Tendenzen der Client-Server-Technik bestimmt(4GLs), zum anderen sind Einflüsse der künstlichen Intelligenz erkennbar(5GLs). Die Sprachen sind typischerweise plattformunabhängig. Sie sind nicht notwendig für den Menschen verständlich, da sie primär der Maschine-Maschine-Kommunikation dienen.

Die modernen Abfragesprachen verfolgen ein teilweise ähnliches Ziel wie die vorliegende Arbeit. Es fehlt ihnen jedoch an einem fachspezifischen Ansatz. Dennoch erscheint es sinnvoll innerhalb einer umfassenden Abhandlung des Themas, auf die aktuellen Entwicklungen auch etwas allgemeinerer Disziplinen einzugehen.

a) Sprachen der vierten Generation am Beispiel von SQL

Bei den Sprachen der 4. Generation[226] handelt es sich um Datenbankentwicklungs– und Abfragesprachen, die einen Zugriff auf Datenbanken unabhängig vom konkre-

[226] **4**th **G**eneration **L**anguages.

ten Datenbanksystem ermöglichen sollen. Der populärste Vertreter dieser Generation ist die in den 80er Jahren von IBM entwickelte Structured Query Language (SQL). Sie soll an dieser Stelle aufgrund der erheblichen Marktbedeutung und wegen ihres recht einfachen Aufbaus als Anschauungsbeispiel dienen.

Grundlage von SQL ist das relationale Datenbankmodell. Es geht davon aus, daß die einzelnen Daten in unterschiedlichen Tabellen abgelegt werden. Eine Zeile einer Tabelle wird als Datensatz bezeichnet, eine Spalte als Feld. Mehrere Tabellen lassen sich nach diesem Ansatz so miteinander verknüpfen, daß einem Datensatz einer Tabelle ein (1:1) oder mehrere (1:n) Datensätze einer anderen Tabelle zugeordnet werden. Die Zuweisung erfolgt über einen eindeutigen Schlüssel (Primärschlüssel)in der führenden Tabelle und einen entsprechenden nicht notwendig eindeutigen Schlüssel in der verknüpften Tabelle (Sekundärschlüssel). Ebenso sind Beziehungen mehrerer Datensätze einer Tabelle zu mehreren Datensätzen einer anderen Tabelle (n:m) über die Generierung einer künstlichen Zwischentabelle mit zwei Sekundärschlüsseln möglich.[227]

Das Ziel der Verknüpfung ist es, jede Information in einer Datenbank nur einmal zu halten[228]. So ist es etwa möglich, daß die Adressen von Personen auch dann nur einmal gespeichert werden, wenn sie in mehreren Sachen auch in unterschiedlicher Weise beteiligt sind. Enthält eine Tabelle beispielsweise die Angaben zu einer Akte, so benutzt sie lediglich eine Referenz auf einen Datensatz in der Tabelle, die die Personalangaben enthält. Unterschieden werden unterschiedliche Grade der Aufspaltung von Daten in einzelne Tabellen. Diese Grade werden als *n. Normalform* bezeichnet. Die Normalform bildet einen Gradmesser der Redundanz der vorgehaltenen Daten. Ein detailliertes Eingehen auf dieses Problem ist jedoch für das Verständnis der Abfragesprache SQL und für den weiteren Fortgang der Arbeit nicht notwendig.[229]

Ein praxisrelevantes Beispiel mag das Problem sein, zu den Daten eines Verfahrens eine beliebige Anzahl von Beteiligten und deren Adressen hinzuzubinden. Im Stammdatensatz des Verfahrens direkt kann nur Raum für eine begrenzte Anzahl von Beteiligten eingeräumt werden, da die Anzahl der Tabellenspalten nicht beliebig ist. Bindet man die Beteiligtendaten mit einer 1:n Beziehung an die Stammdaten, so enthält die zweite Tabelle in einem Sekundärschlüssel einen Verweis auf den entsprechenden Datensatz der Verfahrensdaten. Da jeweils nur ein Sekundärschlüssel für eine Tabelle verfügbar ist, müssen Daten dann doppelt gehalten werden, wenn eine Person an mehreren Verfahren beteiligt ist.

Einen Ausweg bietet eine n:m Verknüpfung. Neben den Tabellen für die Verfahrens- und die Beteiligtendaten enthält eine dritte Tabelle Informationen darüber, welche Personen in welcher Weise an welchem Verfahren beteiligt sind[230].

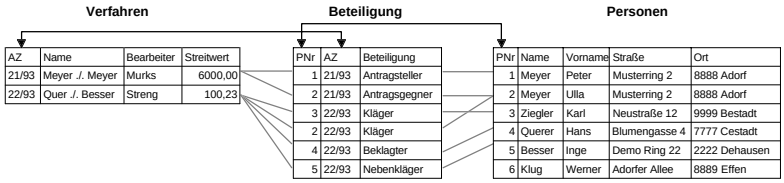
---

[227] Zur Einführung *Korte*, jur-pc 91, S. 1004 ff.

[228] *Neske*, CoR 3/92, S. 8 ff.

[229] Vgl. hierzu *Bund*, S. 255 ff. S.a. Fußnote 227.

[230] Vgl. das Datenbankkonzept AIDA, Abbildung 16 S. 87.



AZ	Name	Bearbeiter	Streitwert
21/93	Meyer ./. Meyer	Murks	6000,00
22/93	Quer ./. Besser	Streng	100,23

PNr	AZ	Beteiligung
1	21/93	Antragsteller
2	21/93	Antragsgegner
3	22/93	Kläger
2	22/93	Kläger
4	22/93	Beklagter
5	22/93	Nebenkläger

PNr	Name	Vorname	Straße	Ort
1	Meyer	Peter	Musterring 2	8888 Adorf
2	Meyer	Ulla	Musterring 2	8888 Adorf
3	Ziegler	Karl	Neustraße 12	9999 Bestadt
4	Querer	Hans	Blumengasse 4	7777 Cestadt
5	Besser	Inge	Demo Ring 22	2222 Dehausen
6	Klug	Werner	Adorfer Allee	8889 Effen

Abbildung 11: Verknüpfung dreier Tabellen einer schematischen Aktenverwaltung

Die Graphik zeigt die drei beschriebenen Tabellen sowie ihre (typischen) Verknüpfungen. Die Tabellen **Beteiligung** und **Verfahren** werden über das Aktenzeichen des jeweiligen Verfahrens verknüpft. Da ein Verfahren mehrere Beteiligte haben kann, besteht zwischen der Verfahrenstabelle und der Beteiligungstabelle eine 1 : n Beziehung[231]. Ebenfalls eine 1 : n Beziehung liegt bei der Verknüpfung der Tabellen **Personen** und **Beteiligung** über die jeweilige Personalnummer vor. Es können nämlich mehrere Personen aus der Personentabelle in unterschiedlichen Formen an einem Verfahren beteiligt sein. Die gesamte Konstruktion ergibt die **n : m Beziehung**[232] zwischen den **Personen** und **Verfahren**, da einerseits in einem Verfahren mehrere Beteiligte existieren, andererseits eine Person an mehreren Verfahren beteiligt sein kann.

Während die Komponenten zum Aufbau von Tabellen und zur Pflege an dieser Stelle von untergeordneter Bedeutung sind, ist der für diese Arbeit interessanteste Bereich von SQL die Abfragefunktionalität. Ziel einer Abfrage ist im Grunde genommen die Erstellung einer neuen (Ziel–)Tabelle aus den zu durchsuchenden (Quell–)Tabellen, die sowohl eine limitierte Anzahl von Feldern der Quelltabellen als auch nur eine selektierte Zahl von Datensätzen enthält. Im minimalen Fall wird dabei lediglich ein Feld ausgewählt und die Suchanfrage trifft auch nur auf einen Datensatz zu. In diesem Fall wird also ein eindeutiger Wert zurückgeliefert. Das andere Extrem ist die Erstellung einer Gesamttabelle, die alle Daten der Quelltabellen mit entsprechend redundanten Informationen enthält.

Die Auswahl der Daten erfolgt in SQL mit Hilfe des Befehls *SELECT*. Weitere Befehle dienen dem Einfügen (*INSERT*), Löschen (*DELETE*) oder Ändern (*UPDATE*) von Daten. Hiermit werden zunächst die gewünschten Felder und die Art der Weiterverarbeitung definiert. Über die folgende Klausel *FROM* werden die betroffenen Tabellen bestimmt und über die wiederum anschließende Klausel *WHERE* wird der Filter angegeben, der die Datensätze selektiert. Die Angabe der Verknüpfung der Tabellen erfolgt ebenfalls in der *WHERE*–Klausel und kann über die Klausel *JOIN* noch weiter spezifiziert werden. Die konkrete Syntax ist oftmals abhängig von dem verwendeten SQL-Dialekt. Das folgende Beispiel zeigt, wie aus der beschriebenen Datenbasis Name und Adresse des Klägers aus dem Verfahren *Quer./. Besser* ermittelt wird:

```
SELECT Name, Vorname, Strasse, Ort
FROM Personen, Beteiligung, Verfahren
WHERE Verfahren.AZ = Beteiligung.AZ
AND Beteiligung.PNr = Personen.PNr
AND Beteiligung.Beteiligung = "Kläger"
```

Der Ausdruck `SELECT` leitet die Abfrage ein, `FROM` definiert die betroffenen Tabellen. `WHERE` enthält die Suchbedingung, wobei die ersten Zeilen dieses Abschnitts die

[231] Eins–Zu–Viele–Beziehung.

[232] Viele–Zu–Viele–Beziehung.

Verknüpfung der Tabellen beschreiben und die letzte Zeile den eigentlichen Filter angibt. Das Ergebnis ist in diesem Fall eine Tabelle, mit zwei Datensätzen:

```
"Ziegler","Karl","Neustraße 12","9999 Bestadt"
"Querer","Hans","Blumengasse 4","7777 Cestadt"
```

Der Zugriff erfolgt je nach Implementierung unterschiedlich, entweder durch dezidierten Zugriff auf die einzelnen Datensätze oder durch Übermittlung der gesamten Tabelle. Das Austauschformat beruht auf reinem Text. Es bleibt der Anwendung überlassen, den Text in den benötigten Variablentypus zu übersetzen. Dabei werden die unterschiedlichen Variablentypen in einer einheitlichen Syntax ausgegeben Das Verfahren soll an dem etwas komplexeren aber sehr praxisrelevanten Problem der **Verwandtschaft** vertieft werden. Hierzu wird eine Datenbank mit den folgenden Tabellen eingeführt:

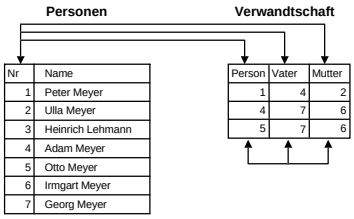


Abbildung 12: Verwandtschaftsdatenbank

Die erste Tabelle *Personen* enthält lediglich die Angabe über die einzelnen Personen. Die zweite Tabelle *Verwandtschaft* enthält Angaben darüber, wer Vater und wer Mutter einer Person ist. Dabei enthalten alle drei Tabellenfelder Sekundärschlüssel mit dem Primärschlüssel *Personalnummern* aus Tabelle 1 für die notwendigen Referenzen[233]. Will man nun den Namen vom Großvater von **Peter Meyer** wissen, so zwingt SQL einen zu einem mehrstufigen Vorgehen:

```
SELECT Verwandtschaft.Vater
WHERE Verwandtschaft.Person = Nr.Personen
AND Personen.Name = "Peter Meyer"

SELECT Personen.Name
WHERE Verwandtschaft.Vater = Personen.Nr
AND Verwandtschaft.Person = X
```

X ist dabei das Ergebnis der ersten Selektion, in diesem Falle also *4*. Zunächst wird also die Personalnummer des Vaters von Peter Meyer gesucht. Danach wird der Name desjenigen gesucht, der Vater dieses Vaters (X) ist. Dieses schrittweise Vorgehen wird als **prozedural** bezeichnet, da die Anwendung die Ablaufschritte, in denen sich ein Programm an ein gesuchtes Datum herantastet, einzeln vorgeben muß. Daran ändert auch die Tatsache nichts, daß über die Einbringung von Variablen mehrere Schritte in einer Abfrage formuliert werden können. Schematisch ist die Suche nach dem Großvater **G** von **E** einfach darstellbar:

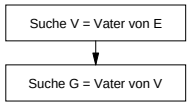


Abbildung 13: Prozedurale Suche nach dem Großvater einer Person

[233] Denkbar ist sogar in bestimmten Fällen daß eine Tabelle Sekundärschlüssel auf eigene Datensätze derselben. Im vorliegenden Fall wäre dann eine zweite Tabelle verzichtbar, wenn jeder Datensatz Schlüsselfelder für Vater und Mutter enthielt. Aus technischen Gründen verzichtet man oft auf dieses Vorgehen. In diesem Fall bestünde die Gefahr einer erzwungenermaßen endlosen Tabelle, wenn man den Eintrag von Vater und Mutter zur Pflicht macht.

Bei komplexeren Anfragen etwa nach den Onkeln einer Person **N** sind entsprechend mehr Schritte nötig:

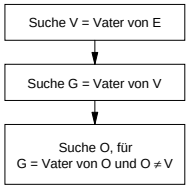


Abbildung 14: Prozedurale Suche nach den Onkeln einer Person

Bei der Suche nach einem Onkel einer anderen Person **M** wird diese Prozedur jeweils wiederholt. Eine prozedural orientierte Abfragesprache bietet also keine effektiven Werkzeuge zur Konservierung von komplexen Beziehungen zwischen Daten über die reine Organisation in den angelegten Tabellen hinaus. Benötigt man im Beispiel die Frage nach komplexeren Verwandtschaftsbeziehungen regelmäßig, so wäre eine Vereinfachung der Abfrage nur durch entsprechend informationsreichere Tabellen zu bewerkstelligen. Diese würden jedoch insofern redundante Informationen aufnehmen müssen, als sich die enthaltenen Beziehungen wie gezeigt auch algorithmisch ermitteln lassen.

Der prozedurale Zugriff auf relationale Datenbanken erscheint insbesondere für per se einfache Applikationen sehr kompliziert. Wird z.B. an einer bestimmten Stelle in einem Schriftsatzformular der Name des Vaters des Klägers aus der aktuellen Akte benötigt, so wäre im Fall eines prozeduralen Zugriffs etwa mittels SQL-Abfragen ein mehrere Schritte umfassender Vorgang notwendig. Dieser läßt sich in einem Textbaustein eines Textverarbeitungsprogramms kaum unterbringen. Das Textverarbeitungsprogramm muß seinen Informationsbedarf an einer Stelle möglichst mit einer einzigen Anfrage formulieren können. Hinzu kommt, daß eine prozedurale Problembeschreibung gerade für den juristischen Autor wesentlich ungewohnter erscheint, als die oben dargestellte Informationsbeschreibung *Name des Vaters des Klägers*.

b) Sprachen der fünften Generation

Als Sprachen der 5. Generation werden üblicherweise **deskriptive Sprachen** bezeichnet. Sie bilden den Gegenpol zu den prozeduralen Sprachen. Sie **beschreiben** die gesuchte Information in einem Ausdruck und verlangen vom Benutzer bzw. dem aufrufenden Prozeß keine Umsetzung seines Problems in eine Kette von Einzelanfragen. Für 5GLs gibt es noch keinen eigenen Standard. An dieser Stelle kann lediglich eine Zusammenfassung einiger Forschungsansätze gegeben werden[234].

Bei deskriptiven Sprachen wird das gesuchte Datum aus einem Pool von **Fakten** mit Hilfe von zusätzlichen **Regeln deduziert**. Das Verfahren der Deduktion wurde bereits bei der juristischen Logik besprochen. Das eigentlich besondere an einer deduktiven Datenbank ist die Verknüpfung von Deduktionsmechanismen und einer klassischen Datenhaltung. So werden klassische Sprachen vor allem aus dem Bereich der künstlichen Intelligenz wie **Prolog** oder **LISP** dazu verwendet, auf herkömmliche relationale Datenbestände zuzugreifen. Das folgende Schema zeigt eine derartige Architektur[235]:

---

[234] Einen guten Überblick vermittelt *Karagiannis*, a.a.O. ; weiter hierzu *Bayer*, a.a.O.; *Lüttringhaus*, a.a.O.; *Preiß*, a.a.O., *Clemens*/*Griefahn*/*Hinze*, a.a.O.; *Maier, A.* a.a.O.

[235] *Karagiannis*, S. 10.

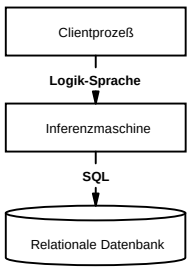


Abbildung 15: Schema eines deduktiven Zugriffs auf eine relationale Datenbank.

Betrachtet man noch einmal die oben beschriebene Datenbank der Verwandtschaftsbeziehungen, so läßt sich die Verwandtschaftstabelle auch als eine Anzahl von **Fakten** i.S. der Prädikatenlogik beschreiben:

$$Vater(A, P) \wedge Vater(O, A) \wedge Vater(G, O)$$

Prädikatenlogische Fakten werden also nach diesem Modell in einer klassischen relationalen Datenbank abgelegt. Dies hat insbesondere den Vorteil, daß im Bereich klassischer Datenbanken der Umgang mit großen Datenmengen wesentlich effektiver zu gestalten ist, wohingegen Interpreter der Logiksprachen bei großen Datenmengen meist sehr unhandlich werden.

Die Endanwendung operiert jedoch abgehoben von der Datenbankorganisation rein auf der Ebene der logischen Sprache. Sie übermittelt für die Erschließung der Fakten zunächst Regeln und letztendlich eine Anfrage. So wären beispielsweise folgende Regeln denkbar:

$$Vater(v, k) \wedge Vater(v, b) \wedge (b \neq k) \rightarrow Bruder(b, k)$$
$$Vater(v, k) \wedge Bruder(v, o) \rightarrow Onkel(o, k)$$

Sie beschreiben allgemein die Verwandtschaftsarten *Bruder* und *Onkel*. Die konkrete Frage nach dem Onkel einer Person **P** läßt sich mit der Relation $Onkel(x, P)$ formulieren. Das System kann anhand der dargelegten Fakten und Regeln mit $x = O$ antworten. Hierzu versucht das System zunächst die Voraussetzungen derjenigen Regel anhand der vorhandenen Fakten zu ermitteln, die die gewünschte Frage unmittelbar beantworten kann. Dies wäre im vorliegenden Fall die zweite Frage. So ließe sich etwa die Voraussetzung $Vater(x, P)$ durch folgende SQL-Abfrage ermitteln:

```
SELECT Vater FROM Verwandtschaft WHERE Person = "P";
```

Die Frage $Bruder(v, A)$ ist hingegen nur durch die Hinzuziehung der ersten Regel lösbar. Hierzu sind nochmals entsprechende Anfragen an die Datenbank zu richten:

```
SELECT Vater FROM Verwandtschaft WHERE Person = "A"
SELECT Person FROM Verwandtschaft WHERE Vater = "E" AND Person <> "A"
```

Die Realisierung einer deskriptiven Abfragesprache erfolgt also durch die Einbringung einer abstrakteren Schicht, die die Logik-basierten Abfragen auswertet und in klassische Abfragen übersetzt. Hierzu wird ein Inferenzmechanismus eingesetzt, der die vorgegebenen Regeln etwa mit Hilfe des beschriebenen **Backtrackings** auswertet und die jeweiligen Fragen für die akut benötigten Fakten formuliert. Im Gegensatz zu einer an eine Reihenfolge gebundenen Abfragesprache ist es für die deduktive Sprache unerheblich, zu welchem Zeitpunkt die Regeln an das System übermittelt werden. Sie können im Prinzip auch noch nach der eigentlichen Anfrage nachgeschoben werden, wenn die Anfrage aufgrund der bekannten Fakten und Regeln nicht zum gewünschten Erfolg führt.

Sobald die notwendigen Regeln einmal dem System übermittelt wurden, ist für die eigentliche Anfrage nach den gewünschten Daten im Prinzip ein einziger Ausdruck notwendig. Hieran ändert sich auch nichts, wenn in einer weiteren Anfrage etwa nach dem Onkel einer anderen Person gefragt würde. Wiederum würde das System anhand der bekannten Regeln und Fakten ohne weitere Vorgabe der Einzelschritte das gewünschte Ergebnis zu beschaffen suchen.

Mit Hilfe dieser Technik besteht nun die Möglichkeit, die notwendige Information sehr einfach und näher an einer verständlichen Sprache zu beschreiben. Sowohl die Formulierung der Regeln, als auch diejenige der Anfragen erfolgt auf einer Ebene, die jedenfalls für den juristischen Programmierer und Endanwender gewohnter ist, als die Erarbeitung von sequentiellen Suchstrategien. Die eigentliche Recherche nach einem Datum kann durch einen einzelnen Ausdruck angestoßen werden. Sie kann somit etwa als Platzhalter etwa in einem Formular untergebracht werden. Das Datenbanksystem muß lediglich rechtzeitig mit Regeln bezüglich des verwendeten Ausdrucks versorgt werden.

Aus der starken Abstraktion der Abfragesprache von der eigentlichen Datenablage ergeben sich jedoch diverse Gefahren. Insbesondere erscheint das System dem Anwender genauso wie dem Programmierer der Anwendungen oftmals als undurchsichtig. Es ermittelt aus z.T. beliebig akquirierten Regeln Lösungen, deren Weg für den Benutzer nicht immer nachvollziehbar ist. Vergleicht man hingegen die Handlungsanweisungen einer prozeduralen Sprache etwa für die Ermittlung des Onkels einer Person (*suche zunächst den Vater, dann dessen Vater und zuletzt dessen weitere Söhne*) so ist diese an sich sehr einfach zu verstehen und auch zu formulieren. Eine auf Regeln basierte Sprache bildet sich ihre Abfragesequenz eigenständig und kommt gerade bei komplexeren Regelwerken nicht immer auf dem direkten Weg zum Ziel.

Eine abschließende Beurteilung der verschiedenen Ansätze soll deshalb an dieser Stelle nicht abgegeben werden. Es bleibt jedoch festzustellen, daß eine einfache Anfrage im Stil der 5GLs für den Endnutzer dann komfortabel ist, wenn das System mit den notwendigen Regeln ausgestattet ist, diese Anfrage eigenständig aus den Fakten zu deduzieren.

Die vorstehenden Erläuterungen beschränkten sich auf einige wenige Themen des Datenaustauschs, die jedoch die Grundströmungen im Bereich des Retrievals von Informationen umrissen haben. Als Tendenz wird zum einen eine Abstraktion von Datenverarbeitung und Datenhaltung deutlich. Zum anderen steigert sich das Abstraktionsniveau der eingesetzten Schnittstellen in Form von Datenbanksprachen. Dabei bauen die Sprachen der sogenannten 5. Generation auf Grundlagen der Prädikatenlogik auf.

4. Realisierungen im Bereich juristischer Programme

Nach einer Betrachtung allgemeiner Techniken der EDV sollen hier noch die Realisierungen im Bereich juristischer Programme umrissen werden.

a) Allgemeines

Die Branche der juristischen Software glänzt wie viele andere mittelständische Branchenlösungen nicht gerade durch schnelle Innovationszyklen. Vielmehr sind noch mit Einführung des Betriebssystems Windows 95 und der damit verbundenen Aufgabe des älteren Systems DOS wesentliche Programme nicht unter einer graphischen Benutzeroberfläche, insbesondere nicht unter Windows verfügbar. Dies beruht wohl zum einen auf dem Umstand, daß der Computer in den Kanzleien oft nur

an den Arbeitsplätzen des Sekretariats verfügbar ist[236]. An diesen Plätzen ist die Innovationsbereitschaft nicht sonderlich hoch, da sie regelmäßig mit hohem Schulungsaufwand verbunden ist. Zum anderen neigen die Benutzer nicht ohne gute Gründe dazu, einmal angeschaffte und operativ laufende System möglichst lange beizubehalten. Letztlich werden besonders Anwendungen mit juristisch anspruchsvollen Lösungen oftmals von Juristen entwickelt, die sich mehr mit der Implementierung der Inhalte als mit der technischen Ausgestaltung etwa der Oberfläche beschäftigen[237].

Grundsätzlich sind jedoch Programme unter Windows weitaus kooperativer als Systeme unter DOS. Dies mag einerseits an den im Betriebssystem vorgesehenen Kommunikationsmöglichkeiten liegen[238]. Andererseits hat sich aufgrund der Multitasking–Fähigkeiten von Windows ein ganz neues Bewußtsein für die Möglichkeiten integrierter Arbeitsumgebungen entwickelt[239]. Zuletzt ermöglicht die einfache Bedienung der Programme zunehmend auch die Einbindung der Entscheidungsträger in die EDV–Organisation einer Kanzlei[240]. Die Bedürfnisse dieser Anwender sind jedoch vielfältiger als die des Sekretariats. Das Bedürfnis, mehrere Anwendungen zu integrieren, steigt entsprechend.

Für die technische Seite der Integration hat die Wahl des Betriebssystems erhebliche Folgen. Während die Kommunikation von DOS-Programmen sich auf den Austausch von Dateien beschränkt, besitzen Windows–Programme diverse Möglichkeiten der Kommunikation. Hierzu gehören die **Zwischenablage**, der **Dynamische Datenaustausch** (**DDE**) sowie die gemeinsame Nutzung von **Programmbibliotheken** (**DLL**)[241], die jedenfalls unter Windows erheblich einfacher und wesentlich populärer ist als unter DOS. Diese Formen der Kommunikation sind nicht programmspezifisch und können unter Windows vorausgesetzt werden.

Ein weiterer Gesichtspunkt für die Integration ist die Bereitschaft und das Interesse der Anbieter, integrative Bestandteile in ihre Software zu implementieren. Einerseits wird der höhere Aufwand gescheut, dessen Nutzen jedenfalls nicht ausreichend augenfällig ist. Andererseits erhöht die Möglichkeit, Programmelemente unterschiedlicher Anbieter zu integrieren, die Gefahr, daß Kunden ein Produkt schneller wechseln. Dies mag zur Neigung mancher Hersteller beitragen, bewußt proprietäre Systeme anzubieten.

b) Anwaltsprogramme

Anwaltsprogramme sind regelmäßig Programmpakete, die aus mehreren speziell für die anwaltliche Tätigkeit zugeschnittenen Programm–Modulen bestehen. Sie erfüllen meist folgende Aufgaben[242]:

– Akten– und Stammdatenverwaltung
– Mahnwesen
– Zwangsvollstreckung
– Gebührenabrechnung

---

[236] *v. Raden*, in **Informationstechnik..**, S. 56: Die größten Erwartungen an den Computer liegen im Bereich der Routinearbeiten und der Textverarbeitung.

[237] So etwa das Programm von *Gutdeutsch* zum Familienrecht.

[238] S. hierzu S. 106 ff.

[239] *Gerblinger*, CoR 2/91 S. 9 ff.; *Juristar*, 8.2.2.

[240] Zu den Kriterien s. Kienbaum-Gutachten 2.5.

[241] S. hierzu S. 106 ff.

[242] *Wolf*, Rechtsanwaltshandbuch 93/94, K III RdNr. 35 ff. Vgl. auch FN. 67.

– Buchhaltung
– Kanzleiinterne Datenbank
– Terminsverwaltung
– Kommunikation
– Juristische Berechnungen
– Textverarbeitung

Sie unterscheiden sich damit im Kernbereich, der Haltung von Kundenstammdaten und der entsprechenden Buchhaltung nicht elementar von Softwarepaketen, die für andere Branchen entwickelt wurden. In einigen Fällen wird für diese Funktionen auch auf einheitliche Module zurückgegriffen[243]. Typisches Merkmal eines guten Anwaltspakets ist die globale Verfügbarkeit zumindest aller Stammdaten[244]. Realisiert wird dies im Regelfall über eine gemeinsame relationale Datenbank.

Offen zugänglich und beispielhaft ist das Datenmodell *AIDA* des Produktes *PHANTASY*[245].

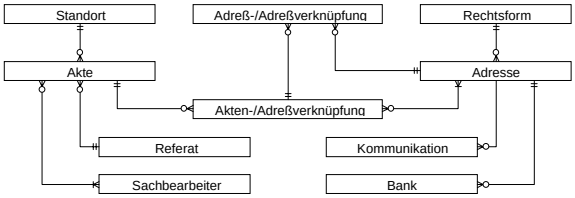


Abbildung 16: Entity–Relationship–Modell des Datenkonzepts AIDA[246].

Die zentrale Datenbasis ermöglicht allen Modulen einen Zugriff auf die benötigten Stammdaten. Setzt man technisch normierte Schnittstellen wie ODBC[247] ein, so können auch Programme auf die Daten zugreifen, die nicht zum eigentlichen Programmpaket gehören. AIDA ist dabei als Versuch zu werten, über die technische Normierung hinaus auch eine semantische Normierung der eingesetzten Tabellen und deren Felder zu erreichen.

Es ist mit dem Entwickler des Systems davon auszugehen, daß auch die übrigen Anwaltsprogramme eine ähnliche Datenhaltung besitzen. Vielfach wird diese jedoch aus Gründen der Kundenbindung nicht offengelegt. Auch die hier gezeigte Darstellung ist längst nicht ausreichend. Vielmehr muß für einen gemeinsamen Zugriff die Struktur der einzelnen Tabellen bekannt sein. Es müssen also zumindest Namen und Typen der Datenfelder offengelegt werden. Hier liegt auch das wesentliche Problem

---

[243] Ein Produkt ist etwa **orgAnice** (orgAnice/RA ist die Version für den Rechtsanwalt), dessen Grundmodule für Adressenverwaltung und Buchhaltung durch branchenspezifische Module an unterschiedliche Bedürfnisse angepaßt wurden (CoR 1994, 194).

[244] Vgl. *Birkigt, Waltl*, CoR 5/91, S. 19; *Möller, Weinknecht*, jur-pc 89, S. 111 ff.

[245] Schöpfer der Idee ist *Andreae*, S. hierzu CoR 4/93, S. 3. Vgl. auch *Nilgens*, jur-pc 1992, S. 1883, 1885.

[246] Quelle CoR 4/93, S. 3 (kein Faximile).

[247] Open Database Connectivity: Standard der Firma Microsoft für den Zugriff auf relationale Datenbanken. Es erlaubt den Zugriff auf unterschiedlichste relationale Datenquellen mit einheitlichen Funktionen. Als Datenbanksprache wird ein SQL–Dialekt eingesetzt. ODBC genießt eine hohe Verbreitung, da es von allen Microsoft–Produkten und vielen Fremdherstellern unterstützt wird. Vgl. auch S. 77.

der beschriebenen Datenhaltung. Die Struktur der Tabellen entscheidet nämlich letztlich über Dichte und Informationsgehalt der gespeicherten Daten[248].

Die Systeme sind für eine Haltung eines begrenzten Umfangs von Stammdaten gut geeignet. Ihre eigentliche Stärke liegt in der Datensicherheit der Strenge des zugrundeliegenden Datenbanksystems. Weiterhin sind die Daten aller Fälle durchsuchbar und können systematisch abgearbeitet werden. Dies ist besonders bei Tätigkeiten von Vorteil, die in gewissem Turnus bei einer Vielzahl von Fällen durchzuführen sind, wie etwa Mahnverfahren und andere Terminsachen. Nicht abgedeckt werden hierdurch jedoch Fakten komplexer individueller Sachverhalte. Dies wirkt sich in den angebotenen Paketen oft bei der Textverarbeitung aus. Hier müssen bei einigen Paketen bestimmte Daten für mehrere Schriftsätze mehrfach eingegeben werden, da die entsprechenden Systeme nicht in der Lage sind, alle Variablen eines Textbausteines zur späteren Wiederverwendung abzulegen.

Aus technischer Sicht ergeben sich jedenfalls keine neuen Anhaltspunkte für die Gestaltung der hier zu entwickelnden Schnittstelle, da die bereits beschriebenen Mechanismen im Einsatz sind.

c) Berechnungsprogramme

Juristische Berechnungsprogramme etwa zum Bereich *Unterhalt* zeichnen sich, sofern sie nicht aus einem global konzipierten Anwaltssystem ausgekoppelt sind[249] durch ein hohes Maß an Individualität aus. Zurückzuführen ist dies wohl auf die Pionierzeit der juristischen Programmierung, in der insbesondere die juristischen Know-how–Träger selbst die entsprechenden Programme entwickelten[250]. Die Programme legen die eigenen Daten regelmäßig in einer Form ab, die in der verwendeten Programmiersprache einfach zu realisieren ist. Auf Gesichtspunkte der Integration mit anderen Systemen achten Sie in der Regel nicht. Die beiden folgenden populären Beispiele zeigen diese Tendenz:

Das **Familienrechtsprogramm** von *Gutdeutsch* wurde ursprünglich für einen programmierbaren Taschenrechner in BASIC entwickelt[251] und später auf C portiert. Es speichert die Daten in einem sogenannten *Protokoll*. Hier werden die Eingaben genau in der Reihenfolge und Form mitgeschrieben, in der sie vom Programm beim Anwender abgefragt werden. Das Protokoll ist also ohne das zugehörige Programm nicht brauchbar. Eine Übernahme der Daten in andere Programm ist nahezu unmöglich. Als einzige Schnittstelle zu Fremdanwendungen fungiert ein ausformuliertes Textprotokoll, das es erlaubt, die Eingaben und Ergebnisse in eine Textverarbeitung zu übernehmen.

Die diversen Programme von *Hoffmann* wurden in Clipper[252] programmiert. Die Eingaben und Ergebnisse werden hier im d-base–Format abgelegt. Die Ausgestaltung der Datensätze ist allerdings recht willkürlich. Es werden oft die Eingaben und Ergebnisse einer Berechnung in einem Datensatz abgelegt. Dabei wird für jeden

---

[248] Bei einem Steuer–Programm konnte ich eine Benennung der Daten durch **Kennzahlen** beobachten. Dies erfolgte aufgrund der Vielzahl und der hohen Änderungsrate der Einzelposten. Der Zugang zu den Daten ist bei dieser Lösung ohne einen entsprechenden Schlüssel nicht mehr möglich.

[249] So etwa die ADVO–LINE des Otto Schmidt Verlages (CoR 1995, 74).

[250] Vgl. *Wolf*, CoR 3/94, 176.

[251] Vgl. *Gutdeutsch* in **Informationstechnik..**, S. 25 ff.

[252] *Clipper* ist ein Compiler auf der Grundlage der d-base–Programmiersprache. Letztere ist bei mehreren Autoren sehr beliebt. Sie besitzt einen sehr allgemeinen Funktionsumfang und ermöglicht eine einfache Ablage der Daten. *Hoffmann*, **PC-Praxis**, S. 130.

Speichervorgang eine Datei mit genau einem Datensatz und dem entsprechenden d-base–Kopf erzeugt. Dieses Speicherformat erlaubt prinzipiell auch einen Zugriff anderer Programme. Das physikalische Datenformat entspricht einem de–facto–Standard. Die einzelnen Datenelemente sind durch die Angaben im Dateikopf benannt. Die Programme sind deshalb allerdings noch nicht **kommunikativ**, d.h. auf einen Datenaustausch angelegt. Die Möglichkeit, die d-base–Datei auszulesen, ist eher ein Zufallsprodukt aus der Wahl der Programmiersprache. Auch die einzelnen Programme des Autors besitzen keine Verbindung untereinander.

Letztlich weisen viele der juristisch interessanten Programme nur sehr marginale Strukturen für einen strukturierten Datenaustausch auf. Aufschluß über eine Gestaltung einer Schnittstelle ist hier nicht zu erwarten. Wesentlich ist jedoch, daß die Schnittstelle nicht allzu aufwendig sein darf, da sonst mit keiner großen Akzeptanz bei den Entwicklern der Endanwendungen zu rechnen ist.

d) Standardprogramme

Einige Autoren, die grundsätzlich eine *Integration* als notwendig erachten, sehen diese in integrierten Softwarepaketen wie *Microsoft Works* oder *Framework III*[253]. Andere vertrauen auf die einigende Wirkung etwa von *Microsoft Windows*[254].

Sowohl Paketlösungen als auch *Windows* oder auch das Betriebssystem des Macintosh stellen gute Plattformen zur Realisierung einer integrierten Lösung dar. Dabei handelt es sich aber lediglich um eine technische Option, Daten auszutauschen. Die eigentliche inhaltliche Implementierung ist hierbei jedoch nicht geleistet. So können in Paketlösungen Daten aus der integrierten Datenbank jeweils in die Textverarbeitung direkt übernommen werden. Bei Windows wird derselbe Effekt über die Standardschnittstelle DDE erzielt. Die Ausgestaltung der Datenbank und die Plazierung der notwendigen Elemente in dem Schriftsatz muß aber selbstverständlich von dem Anwender übernommen werden.

Die Integration mittels dieser Werkzeuge ist deshalb für den Anwender sehr belastend. Will er sich eine individuelle Umgebung zusammenstellen, so obliegt ihm der inhaltliche Aufwand der Integration. Will er sich hingegen aus Fremdangeboten bedienen, so hängt das Maß an Integration von der Normierung der inhaltlichen Schnittstellen zwischen den diversen Anbietern ab.

Die marktüblichen Programme für juristische Aufgaben ermöglichen nur dann integriertes Arbeiten, wenn sie aufeinander abgestimmt sind. Im Bereich der technischen Realisierung ist lediglich der Vorstoß von *Andreae* mit seinem Konzept *AIDA* als Versuch zu werten, eine Integration mehrerer Produkte zu ermöglichen. Im übrigen gehen hiervon keine Impulse für die weitere Konzeption des zu entwickelnden Systems aus.

C. Entwurf eines Konzeptes

1. Einleitung

Bisher wurden Datenaustausch bzw. Datenbeschreibung aus juristischer sowie informationstechnischer Sicht beschrieben. Als Ergebnis läßt sich sicherlich resümieren, daß sich Fakten wohl am elegantesten mit den Mitteln der Prädikatenlogik beschreiben lassen. Im folgenden soll die Entwicklung eines eigenen Ansatzes dargestellt werden, der sich insbesondere diese Erkenntnis zunutze macht.

---

[253] *Walter*, jur-pc 91, S. 897 ff.; *Endrös* CoR 6/89, S. 22 ff.

[254] *Kraft*, jur-pc 91, S. 971 ff.; *Gerblinger*, CoR 2/91 S. 9 ff.

Die dem Modell zugrundeliegende und hier zu entwickelnde Schnittstelle wird in den folgenden Ausführungen als *System* bezeichnet. Wird von einer Anwendung ein Faktum erfragt, so wird dieser Vorgang als *Anfrage (an das System)*, die Rückmeldung als *Antwort* oder *Rückgabe* bezeichnet. Eine Anwendung besitzt neben der Anfrage auch die Möglichkeit, von sich aus ungefragt Fakten an das System zu leiten. Dieser Vorgang heißt zukünftig *Übermittlung (an das System)*.

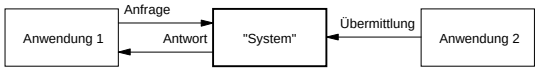
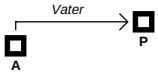


Abbildung 17: Abstrakte Architektur der hier zu beschreibenden Schnittstelle (System).

2. Prädikatenlogisches Modell

Grundlage des Ansatzes ist ein prädikatenlogisches Modell der Fakten eines Sachverhaltes. Der Ansatz geht zunächst von der Existenz von *Objekten* (in der Prädikatenlogik *Individuen*) und *Beziehungen* zwischen diesen Objekten (in der Prädikatenlogik *Relationen* oder *Funktionen*) aus. So sind *A* und *P* beispielsweise existierende Objekte, zwischen denen die Beziehung Vater(A, P) besteht. Die Graphik zeigt diesen Sachverhalt. Objekte werden dabei als Rechtecke, Beziehungen als Pfeile gekennzeichnet.



Es können beliebig viele Objekte durch eine Beziehung verbunden sein. Eine Eigenschaft liegt vor, wenn genau ein Objekt Gegenstand einer Beziehung ist: $Mann(A)$.

3. Präzisierungen

a) Terminologie

Ein **Objekt** kann alles sein, was Eigenschaften haben kann oder mit anderen Objekten in Beziehung stehen kann: Eine real existierende Person ist genauso ein Objekt wie eine Zahl oder eine beliebige Kette von Zeichen. Eine **Objektvariable** ist ein Platzhalter, der beliebige Objekte einer definierten Menge von Objekten repräsentiert.

Ein Ausdruck, der durch die Verwendung von Objektvariablen[255] eine Vielzahl von möglichen Beziehungen beschreibt, wird im folgenden als **Relation** bezeichnet. Eine Relation ist etwa der Ausdruck $Kind(k, v, m)$. Eine Relation kann auch in funktionaler Schreibweise dargestellt werden. So ist auch der Ausdruck $k = Kind(v, m)$ ebenfalls eine Relation. Eine Funktion liegt lediglich dann vor, wenn der Ausdruck alle Objekte *v* und *m* eindeutig auf jeweils ein Objekt *k* abbilden würde. $Vater(v, k)$ ist eine solche Funktion, wenn es definitionsgemäß für jedes Kind *k* nur einen Vater *v* gäbe:

$$\forall x \forall y \forall z((x=Vater(y)) \wedge (z=Vater(y)) \rightarrow (z = x))$$

Von einer **Beziehung** wird dann gesprochen, wenn anstelle der Objektvariablen **konkrete Objekte**[256] angegeben werden. Der mit einer Beziehung gemeinte Ausdruck beschreibt allerdings einen rein **hypothetischen** Sachverhalt. Er beinhaltet noch keinerlei Festlegung dahingehend, ob die beschriebene Beziehung **tatsächlich** besteht oder nicht. Er ist anders als eine Aussage auch keiner logischen Bewertung zugänglich.

---

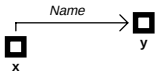
[255] Objektvariablen werden regelmäßig mit Kleinbuchstaben geschrieben.

[256] Bezeichner konkreter Objekte werden groß geschrieben.

Eine Beziehung die dahingehend konkretisiert ist, daß behauptet wird, sie existiere oder existiere nicht, ist eine **Aussage**. Eine Aussage kann wahr oder falsch sein, je nachdem ob die Behauptung, daß eine Beziehung existiere, der Realität entspricht oder nicht.

b) Funktionale Syntax

Betrachtet man die oben abgebildete Beziehung, so verbindet die Relation *Vater* zwei Personen *A* und *P* miteinander. Benötigt nun eine einfache Anwendung (Verfassen eines Schriftsatzes) die Information, wer der Vater von *P* ist, so bedarf diese einer Präzisierung dahingehend, daß regelmäßig nicht das abstrakte Objekt *A*, *sondern* dessen **Name** gesucht sein wird. Die bisher verwendeten *Namen* dienen lediglich als Bezeichner[257] der Objekte, nicht jedoch als deren Namen in dem Sinne, wie sie in einem Schriftsatz zu verwenden sind. Hierzu dient eine eigene Beziehung $Name(x, y)$ zwischen einer Person und Ihrem Namen. *x* ist hierbei Individuenvariable für den Namen von *y*.



Bei einer Anfrage nach dem Namen des Vaters von *P* bedarf es also zweier Schritte:

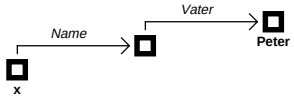
$$Name(x, y) \wedge Vater(y, P).$$

Die Darstellung ist aufwendig. Zudem stellt sie nicht eindeutig dar, wonach letztendlich gesucht wird, *x* oder *y*. Eine verschachtelte Beschreibung behebt dieses Problem:

$$Name(x, Vater(y, P)).$$

Diese Darstellung weist einen Schwachpunkt dadurch auf, daß die Beziehung $Vater(y, P)$ an die Stelle eines Individuums in der Beziehung $Name(x, y)$ tritt. Dies mag zu der Annahme führen, hier werde die Prädikatenlogik zweiter Stufe eingeführt. In der Prädikatenlogik zweiter Stufe werden Prädikate über Prädikate gebildet. D.h. an die Stelle einer konkreten Relation tritt eine Relationsvariable. Ihr Abbildungsverhalten steht in diesem Falle noch nicht fest. Sie könnte die Relation *Mutter*, *Vater* oder auch jede andere Relation beschreiben.

Eine solche Konstellation ist mit dem Ausdruck jedoch nicht gemeint. Vielmehr wird von einer konkreten Abbildung, also von einer Relationskonstanten ausgegangen. Es wird lediglich eine Funktionsdarstellung verwendet, die *P* auf dessen Vater abbildet. Die genaue Formulierung müßte also $x=Vater(P)$ heißen. Der Gesamtausdruck hieße dann $Name(x, Vater(Peter))$. Die Graphik veranschaulicht diesen Ausdruck nochmals:



Will man nun Verschachtelungen beliebiger Tiefe zulassen, so etwa die Frage nach dem **Namen des Vaters des Klägers in einem Verfahren V**, so wird man folgende Forderung formulieren müssen:

---

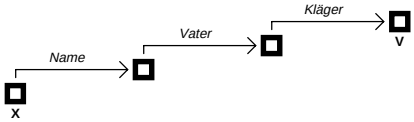
[257] Meist wird hierfür die englische Bezeichnung *Identification* (Identifikatoren, kurz: ID) verwendet.

Regel 1

> Beziehungen zwischen Objekten sind regelmäßig so zu beschreiben, daß sie in funktionaler Schreibart $y = F(x)$ formulierbar sind.

Im Regelfall ist dieses Attribut gänzlich verzichtbar, da es vom Prädikat selbst repräsentiert wird.

Die Darstellung gilt natürlich auch für die Beziehung *Name* im genannten Beispiel, so daß die Formulierung einer letzten Präzisierung bedarf: $x=Name(Vater(Peter))$. Diese Prämisse läßt auch tiefere Verschachtelungen zu wie die Frage nach dem *Namen des Vaters des Klägers in einem Verfahren V*: $x=Name(Vater(Kläger(V)))$.

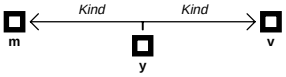


Eigenschaften werden bei funktionaler Darstellung als Konstanten dargestellt. Sie haben keine Parameter, bilden also auch keine Variablen auf eine weitere Variable wie $y=Mensch()$

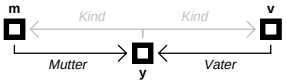
c) Präzisierung der Parameter

Regel 1 führt zu einer sehr kompakten Darstellung von Beschreibungen von einzelnen Fakten, da sie eine beliebige Verschachtelung von Einzelbeziehungen ermöglicht. Eine derartige Verschachtelung ist in der juristischen Praxis durchaus denkbar. So wird etwa der Name des Vaters des Mandanten in Schriftsätzen zum Unterhalt oder in Erbstreitigkeiten benötigt. Ähnliches gilt für das Nettoeinkommen des Vaters des Klägers. Viele andere z.T. sicher noch weitaus komplexere Beschreibungen sind bei komplexen Vertragsgestaltungen denkbar.

Die funktionale Darstellung ergibt dann Probleme, wenn nicht das Individuum einer Beziehung gesucht ist, welches durch die Funktion beschrieben wird, sondern eines, das durch einen Parameter der Funktion repräsentiert ist. Sucht man beispielsweise nicht den Vater von *P, sondern* das Kind von *A* wird dieses Problem klar. Der Ausdruck $A=Vater(x)$ muß zunächst ähnlich einer mathematischen Formel nach *x* aufgelöst werden. Das Problem wird besonders dort deutlich, wenn die Frage Teil eines komplexeren Ausdrucks ist. Zu beschreiben ist beispielsweise der Name des Kindes von *A*. Zu beheben ist dieses Problem dadurch, daß für jeden Parameter einer Funktion eine auflösende Funktion definiert wird. In diesem Fall könnte diese Funktion $y=Kind(v)$ lauten. Diese Auflösung leidet jedoch offensichtlich an einem Mangel: *y* ist de facto das Kind zweier Eltern, einer **Mutter** und eines **Vaters**. Die Funktion muß entsprechend erweitert werden: $y=Kind(v, m)$ Die Erweiterung läßt sich graphisch wie folgt darstellen:



Die Relation *Kind* korrespondiert entsprechend mit zwei Relationen *Vater* und *Mutter* bzw. aus der Relation *Kind* lassen sich die beiden anderen $v=Vater(y)$ und $m=Mutter(y)$ ableiten. Hieraus ergibt sich ein Abbild für die gesamten Eltern-Kind Beziehungen.



Letzteres Beispiel zeigt, daß die Position eines Parameters einer Funktion aufgrund ihrer reversen (auflösenden) Funktion eine besondere Bedeutung erfährt. Die einzusetzenden Objekte können nicht beliebig ausgetauscht werden. So ist der erste Pa-

rameter der Funktion *Kind* per Definition durch die Funktion *Vater* aufzulösen, Parameter 2 durch die Funktion *Mutter*.

Regel 2

> Zu jeder Variablen *x* einer Relation existiert eine weitere (reverse) Relation *f*, die keine, eine oder mehrere Variablen mindestens auch auf *x* abbildet.

Die Regel ist weit gefaßt. Sie zwingt nicht dazu, daß eine Variable der reversen Relation wiederum durch die Ursprungsrelation abgebildet wird, wie dies etwa bei *Kind* und *Vater*/*Mutter* der Fall ist. Sie ermöglicht es sogar, daß lediglich eine quasi reverse Eigenschaft vorliegt, wie etwa die Eigenschaft *Mensch* oder die Grundeigenschaft *Objekt*, die die Elementenmenge auf sich selbst abbildet. Im Beispiel $y=Geburtstag(x)$ gibt es keine sinnvolle Funktion die den Geburtstag *y* nach *x* auflöst. Statt dessen wird die Eigenschaft *Mensch* oder *Lebewesen* hier einzusetzen sein.

Die Regel führt also zu einer Beschränkung der in eine Variable einzusetzenden Elemente. Es kann nicht jedes Element der Elementenmenge in eine Variable eingesetzt werden, sondern nur diejenigen Elemente, die durch die reverse Relation abgebildet werden können[258]. In den folgenden Darstellungen von Relationen werden die Variablen mit den Namen der reversen Relationen bezeichnet. Zur Unterscheidung von den Relationen selbst, werden sie weiterhin klein geschrieben.

d) Vollständigkeit der Parameter

Häufig ergibt sich bei der Bildung von Relationen (Prädikaten) die Frage nach der Definition der Variablen. Betrachtet werden soll das Beispiel *Ehe*: Sie ließe sich beispielsweise durch $y=Ehe(ehemann, ehefrau)$ darstellen. Die Relation *Ehe* bildet also die Variablen *ehemann* und *ehefrau* auf ein abstraktes Objekt *y* ab, die Ehe. Die gewählten Parameter sind zunächst naheliegend, scheinen jedoch bei näherer Betrachtung nicht vollständig. So ist die Abbildung nicht eindeutig, da dieselben Personen mehrmals verheiratet sein können. Die Parameter reichen also für die Identifizierung genau einer Ehe nicht aus. Vielmehr würde man Parameter wie den Zeitpunkt der Eheschließung, die Registernummer oder ähnliche Anhaltspunkte benötigen, um eine Eindeutigkeit herzustellen.

Die hier gewählte Lösung geht von einem wesentlich pragmatischeren Ansatz aus. Als Parameter sollen danach nur die wesentlichen juristischen Beteiligten verwendet werden[259]. Für eine präzise Identifikation wird eine Ordnungsziffer herangezogen[260]. Bildet also eine Relation bestimmte Objekte auf mehrere Objekte ab, so werden diese quasi numeriert und durch eine Ordnungszahl unifiziert. Aus der Relation wird so eine Funktion. Ein Objekt wird eindeutig gekennzeichnet[261]. Die Sortierreihenfolge der Numerierung kann vorgewählt werden. Sie wird regelmäßig die Historie berücksichtigen. Das **erste** Kind ist also das älteste, etc.

Welche Parameter als wesentlich gelten, wird aus dem gesetzlichen Grundtatbestand geschlossen. So ergibt sich aus § 1353 BGB für den Grundsatz der Ehe, daß sie zwischen einem **Ehemann** und einer **Ehefrau** geschlossen wird. Sind zwei Personen (im Beispiel *Peter* und *Tina*) zum zweiten Male verheiratet, kann dies als

---

[258] Vgl. zur Bildung von *Sorten* FN. 45.

[259] Vgl. S. 45.

[260] Vgl. S. 57.

[261] Vgl. S. 44.

$y = Ehe(Peter, Paula,2)$ dargestellt werden. Diese Funktion bildet also Peter und Tina auf ihre **zweite Ehe** ab. Der Ordnungsparameter ist aus Gründen der Übersicht und Praktikabilität optional. Wird er weggelassen, so bildet die Funktion die Objekte immer auf das letzte Objekt ab, das die Relation erfüllt. Es wird also die größte Ordnungsziffer eingesetzt.

Regel 3

> Als Parameter einer Funktion werden lediglich die aus der rechtlichen Definition benötigten wesentlichen Objekte verwendet. Zur Unifizierung einer Funktion wird eine optionale Ordnungsziffer eingeführt.

Alle weiteren denkbaren Parameter werden nicht in die Hauptfunktion eingebracht. Sie werden vielmehr als zusätzliche Funktionen dargestellt, die in diesem Fall die Ehe auf die entsprechenden Objekte abbilden. So wäre beispielsweise die Registernummer einer Ehe als $Registernummer(ehe)$ darzustellen. Eine Besonderheit bildet der Parameter Zeit. Hierauf wird später[262] noch genauer eingegangen.

e) Werte von Objekten

Das bisher dargestellte Instrumentarium erlaubt es, auf einfache, effektive Art, Objekte anhand von Funktionen zu identifizieren. Eine Funktion bildet mehrere Objekte auf ein neues Objekt ab. Wie bereits dargestellt wird ein Objekt durch einen Bezeichner, der regelmäßig eine Art kurzer Name sein wird, beschrieben. Dieser Name ist jedoch nicht zwingend identisch mit dem Vor- oder Nachnamen etwa einer Person. Insbesondere bei abstrakten Objekten wie etwa dem Geburtstag, ist der Bezeichner (etwa *Gebutrtstag von Peter*) wenig aussagekräftig, wenn nicht sogar trivial. Vielmehr wird die konkrete Datumsangabe (etwa *23.03.72*) benötigt. Diese Angaben, der Name einer Person, das Datum eines Ereignisses oder der Betrag eines Preises werden im folgenden als **Wert** bezeichnet. Ziel der Übertragung von Fakten ist hiernach konkret die Übermittlung eines Wertes, der regelmäßig einem Objekt zugeordnet werden kann. Hieraus folgt

Regel 4

> Objekte besitzen neben ihrem Bezeichner einen oder mehrere Werte.

Die Werte können von unterschiedlichem Typs sein. So folgt ein Datum anderen Regeln als ein Betrag einer Forderung und wiederum anderen Regeln als ein einfacher Name[263]. Beträge können beispielsweise addiert und subtrahiert werden. Dies ist bei einem Datum nur eingeschränkt, bei einem Namen jedoch gar nicht möglich. In der Datenverarbeitung haben sich deshalb bestimmte Grundtypen von Werten herausgebildet, die an dieser Stelle übernommen werden:

- Zeichenkette
- Numerischer Wert
- Datum-Zeitwert
- Logischer Wert

Eine *Zeichenkette* ist jede beliebige Abfolge von Zeichen, wie Namen, Markenbezeichnungen etc. Es ist der allgemeinste Wertetyp. Er besitzt keine definierten Weiterverarbeitungsmöglichkeiten wie Addition, Subtraktion etc. Er dient der reinen Anzeige.

---

[262] S. S. 97.

[263] Vgl. Begriffsformen, S. 32.

Numerische Werte werden gemeinhin unterteilt in Ganzzahlen und Fließkommawerte. Diese Unterscheidung hat meist nur technische Gründe und ist deshalb in einer abstrakten Konzeptbeschreibung verzichtbar[264]. Numerische Werte können Gegenstand der klassischen mathematischen Operationen sein. Sie repräsentieren meist Geldwerte, Stückzahlen, Maßangaben aber auch die Intensität von Merkmalen etc.

Datum-Zeitwerte wurden in den bisherigen Beschreibungen weithin unberücksichtigt gelassen. Tatsächlich werden sie ähnlich wie numerische Werte behandelt und deshalb von ihnen teilweise nicht unterschieden. Typischerweise wird ein Datum-Zeitwert in seiner Repräsentation als numerischer Wert in die Zeitspanne zwischen einem definierten festen Datum (etwa der 1.1.1900, 0.00 Uhr) und dem abgebildeten Datum umgerechnet. Auch bei der Repräsentation als numerischer Wert gelten für Datumswerte eigene Rechenregeln. Sie können nicht **miteinander** addiert, wohl aber zur Berechnung der Zeitdifferenz voneinander subtrahiert werden. Das Ergebnis ist jedoch dann eine Zeitspanne in Form eines einfachen numerischen Wertes und kein Datum. Auch kann zu einem Datum eine Zeitspanne (8 Tage, 4 Monate etc.) addiert bzw. von ihm subtrahiert werden. Wie bei numerischen Werten können auch bei Daten sinnvolle Vergleiche wie größer (später) und kleiner (früher) angestellt werden. Jede Form der Multiplikation hingegen ist sinnlos.

Logische Werte sind zunächst *wahr* oder *falsch*. Als solche können sie Gegenstand klassischer boolescher Operatoren wie Konjunktion, Disjunktion, Exclusion, Implikation etc. sein. Der logische Wert eines Objekts beschreibt dessen reale Existenz. Ein Objekt E für das gilt $E = Ehe(Peter, Paula, 2)$ besitzt genau dann den Wert wahr, wenn *Peter* und *Tina* mindestens zweimal verheiratet waren. Nur dann gibt es dieses Objekt tatsächlich. Andernfalls ist E ein hypothetisches Objekt, daß gar nicht existiert.

Neben den Werten *wahr* und *falsch* erlaubt das System einen (bewußt) undefinierten Wert *unklar*. Die sich hieraus ergebende *dreiwertige Logik* läßt sich eindeutig aus der zweiwertigen Logik ableiten. Dabei ist das Ergebnis eines logischen Ausdrucks immer dann bestimmt, wenn man für den unklaren Wert sowohl **wahr** als auch **falsch** einsetzen kann, ohne daß sich das Ergebnis des logischen Ausdrucks ändert. Andernfalls ist der Ausdruck selbst ebenfalls *unklar*.

Zu unterscheiden ist der **bewußt unklare** Wert von der immer in Betracht zu ziehenden Möglichkeit, daß ein Wert bei der Anfrage an das System (noch) **unbekannt** ist. Der bewußt undefinierte Wert setzt voraus, daß ein Benutzer oder ein Algorithmus den Wert bereits entscheiden sollte und ihn für unklar gehalten hat. Der unbekannte Wert hingegen stand innerhalb eines Falls noch nicht zur Entscheidung an. Rechenoperationen werden diese Varianten kaum unterschiedlich behandeln. In beiden Fällen ist der Wert unklar. Für Ablaufsteuerungen mag die Unterscheidung jedoch wichtig sein. Wurde der Wert bereits zur Entscheidung vorgelegt, so mag es innerhalb eines Programmablaufes sinnvoll sein, zunächst zu versuchen einen anderen Wert zu entscheiden und so die Entscheidung über den unklaren Wert obsolet werden zu lassen. Ist beispielsweise in dem logischen Ausdruck $y \Leftrightarrow a \vee b$ der Wert von a (bewußt) unklar, so wird ein Prüfungsmechanismus, der den **Urteils-**

[264] Tatsächlich wird oftmals noch zwischen unterschiedlichen Wertebereichen und Genauigkeiten differenziert. Die Verwendung der Variablentypen hängt vom tatsächlichen Bedarf einerseits und den technischen Gegebenheiten wie Speicherplatz oder Prozessorleistung andererseits ab.

stil[265] abbildet, zunächst bei *b* fortfahren, da sich, falls *b* wahr ist, eine nähere Prüfung von *a* erübrigt. Ein Mechanismus mit **Gutachtentechnik**[266] hingegen wird ohnehin beide Werte abklären müssen.

Das System bietet keinen eigenen Wertetyp für eine mehr als dreiwertige Logik. Sollen analoge Unsicherheitswerte übermittelt werden, muß statt logischer Werte auf die numerische Darstellung ausgewichen werden.

f) Wahrheit von Aussagen

Vom logischen Wert eines Objekts ($\exists x(x = A)$) ist die Frage zu unterscheiden, ob eine Beziehung zwischen bestimmten Objekten besteht ($B=F(A)$). In dem Ausdruck $Anton = Kind(Peter, Tina)$ beispielsweise können *Anton*, *Peter* und *Tina* existierende Individuen sein. Die logischen Objektwerte wären entsprechend wahr. Dennoch kann die Aussage als ganzes falsch sein, wenn *Anton* nicht das Kind von *Peter* und *Tina* ist.

Regel 5

Konkreten Aussagen können Wahrheitswerte zugeordnet werden.

Bedeutung gewinnt die Differenzierung aufgrund der später zu behandelnden Unsicherheit einzelner Fakten. Zieht eine Norm beispielsweise aus dem Bestehen einer Ehe zwischen zwei Personen eine Folge *y* ($\forall m \forall f \forall e \forall y((x = Ehe(m, f)) \rightarrow y)$), so sind für eine konkrete Folge *Y* zwei Merkmale gleichzeitig gefordert: die Existenz eines Objekts *E* und dessen Beziehung *Ehe* zu den Objekten *M* und *F*. Behauptet eine Partei das Bestehen einer solchen Ehe $E = Ehe(M, F)$, so kann sich ein Bestreiten der Gegenpartei gegen die Existenz der Ehe $\neg\exists x(x = E)$ oder gegen die Tatsache richten, daß es sich bei der angeblichen Ehe wirklich um eine Ehe handelt $\neg(E = Ehe(M.F))$. Die mögliche Differenzierung muß durch den Sprachumfang des zu entwickelnden Systems gewährleistet werden.

g) Funktionen und Typen

Die Möglichkeit, Objekten unterschiedliche Formen von Werten zuzuweisen, macht einen Zugangsmechanismus zu diesen Werten notwendig. Die Funktion $y = Geburtstag(mensch)$ bildet nach der bisherigen Definition die Variable *mensch* auf *y* ab, wobei *y* ein Objekt aus der Elementenmenge repräsentiert. Konkret wird z.B. der Geburtstag von *P* auf *T* abgebildet, wobei *T* der Geburtstag von *P*, nicht jedoch der Wert (z.B. *23.03.82*) ist. Hingegen würde zunächst bei unbefangener Betrachtung wohl eine Abbildung auf den Wert erwartet werden. Ebenso liegt es bei einer einfachen Funktion wie $y = Name(mensch)$. Nach der bisherigen Konvention repräsentiert *y* nicht die Zeichenkette (z.B. *Paul*), sondern ein abstraktes Objekt, das den Namen darstellt.

Anders mag es im folgenden Fall liegen: $Geburtstag(A) = Todestag(B)$. Um den Wahrheitsgehalt dieser Aussage beurteilen zu können, ist zunächst festzustellen, ob die Gleichheit der Objekte oder die Gleichheit der Datumswerte gemeint ist. Besonders im Streitfall bestehen hier Variationsmöglichkeiten: So kann unstreitig sein, daß B bei der Geburt von A gestorben ist. Dann würde es sich bei den Daten um dieselben Objekte handeln. Streitig wäre lediglich der Datumswert. Es kann aber auch gerade diese Frage, starb die Mutter *A* bei der Geburt ihres Sohnes *B*, im Streit

---

[265] Hier wird die rechtliche Prüfung beendet, falls auch nur eine Tatsache den geprüften Anspruch zum scheitern bringt. Vgl. *Sattlelmacher*/*Sirp*, S. 258.

[266] Hier werden alle Merkmale auch dann geprüft, wenn das Scheitern des Anspruchs bereits feststeht. Vgl. *Sattlelmacher*/*Sirp*, S. 258.

stehen, ungeachtet dessen, an welchem Datum sich dies ereignete. Beide Ereignisse können aber auch gänzlich unabhängig voneinander sein. Eine Identität der Datumswerte wäre somit rein zufällig. Eindeutiger ist hier etwa die Aussage $Geburtstag(A) < Todestag(B)$, da sie sich nur auf die Datumswerte beziehen kann.

Wiederum anders liegt die Sache bei verschachtelten Funktionen wie $Geburtstag(Vater(Kläger(V)))$. Die Variablen von Funktionen stellen immer nur Elemente der Menge aller Objekte dar. Eine Funktion, die innerhalb eines verschachtelten Ausdrucks auftritt, muß deshalb als Ergebnis immer ein Objekt liefern.

In Regel 4 wurde die Möglichkeit eröffnet, daß ein Objekt mehrere Werte unterschiedlichen Typs besitzen kann. Erwartet man, daß es eine Funktion $f(x)$ gibt, die das Objekt *x* auf seinen Wert abbildet, so muß definiert werden, welcher Wert hiermit gemeint ist. Hieraus ergibt sich die Notwendigkeit, mehrere Funktionen bereitzustellen, die Objekte auf deren Werte abbilden. Dies wird durch **vordefinierte Funktionen** ermöglicht:

Regel 6

> Für jeden Wertetypus existiert eine Funktion, die ein Objekt auf dessen Wert mit diesem Typus abbildet.

Diese Funktionen werden in Zukunft mit $Zeichen(x)$, $Zahl(x)$, $Datum(x)$ sowie $Existiert(x)$ bezeichnet.

Soll nun ein Objekt auf den Wert seines Geburtsdatums abgebildet werden, müßte dies durch den Ausdruck $y = Datum(Geburtstag(x))$ erfolgen. Für die technische Umsetzung wird noch eine Vereinfachung vorzusehen sein, bei der bereits die Funktion $Geburtstag(mensch)$ das Objekt *mensch* auf den Datumswert seines Geburtstages abbildet.

h) Neue Definition von Fakten

Das vorgestellte Modell geht von einem prädikatenlogischen Ansatz aus. Hierbei gibt es Objekte und Beziehungen der Objekte untereinander. Diese Beziehungen werden als Funktionen dargestellt, die kein, ein oder mehrere Objekte auf ein neues Objekt abbilden. Objekte besitzen Werte, die unterschiedliche Typen haben können. Durch vordefinierte Funktionen werden Objekte auf ihre Werte abgebildet. Mit diesem Regelwerk läßt sich auch der Gegenstand des Datenaustauschs, das Faktum, neu definieren:

⇒ *Fakten sind die Werte und Beziehungen von Objekten. Dabei ist es unerheblich, ob die Objekte tatsächlich existieren. Vielmehr ist der Existenzwert eines hypothetischen Objekts ebenfalls ein Faktum.*

4. Juristische Faktoren

Bisher wurde davon ausgegangen, daß eine Funktion mehrere Objekte auf genau ein weiteres Objekt abbilde. Im Fall einer faktischen Mehrdeutigkeit wurden Ordnungsziffern eingeführt. Des weiteren wurde angenommen, daß ein Objekt (genau) einen Wert desselben Typs habe, der regelmäßig das Ziel des angestrebten Austauschvorganges darstelle. Deshalb wurde die Frage etwa nach dem Geburtstag der Mutter des 1. Klägers im Verfahren V als Funktion $x = Datum(Geburtstag(Mutter(Kläger(V, 1))))$ formuliert. Das so entwickelte Regelwerk ignoriert dabei drei in der juristischen Praxis erhebliche Faktoren:

1. Werte und Beziehungen können sich mit der Zeit ändern[267].

[267] Vgl. S. 34.

2. Unterschiedliche Personen können unterschiedliche Standpunkte zu einzelnen Werten oder Beziehungen einnehmen[268].
3. Die Standpunkte einer Person zu einzelnen Werten oder Beziehungen können sich im Laufe der Zeit ändern[269].

Kurz gesagt παντα ρει, alles ist im Fluß und nichts ist verläßlich. Die folgenden Punkte versuchen diese oft in der juristischen Praxis relevanten Probleme aufzugreifen und zu lösen.

a) Zeitliche Änderung von Fakten

Die zeitliche Veränderung von Werten ist der einfachste Unsicherheitsfaktor und noch wenig spezifisch für juristische Belange. So sinkt der Wert einer normalen Sache mit der Zeit, derjenige eines Hauses steigt oft. Ehen werden geschlossen und wieder geschieden. Schulden werden beglichen, Kaufsachen zerstört. Arbeits- oder Einkommensverhältnisse sehen sich regelmäßigen Schwankungen ausgesetzt.

Bei der Frage nach einem Faktum ist es also von erheblicher Bedeutung, zu welchem Zeitpunkt dieses Faktum vorgelegen haben soll. Der Zeitpunkt ist wesentliches Element des Faktums. Hierzu wird zweckmäßigerweise jeder Funktion ein Parameter beigegeben, der den Zeitpunkt angibt, für den der gewünschte Wert gelten soll[270].

Regel 7

> Es sind mehrere Angaben über Fakten möglich. Jede Angabe ist zeitlich determinierbar.

In einem Großteil der Fälle wird der Zeitfaktor gar keine Rolle spielen. So ändern sich Werte wie der Name, das Geburtsdatum oder die Adresse einer Person innerhalb des eingegrenzten Zeitraums eines Falls oftmals nicht. Aus diesem Grund ist es sinnvoll, den zusätzlichen Parameter Zeit **optional** zu gestalten. Für den Fall der Auslassung des Parameters müssen Regeln verfügbar sein, nach denen der fehlende Faktor ergänzt wird. Eine einfache und sicher pragmatische Regel ist es, immer den aktuellsten Wert zu verwenden.

Wenn auch Fakten einer fließenden Veränderung ausgesetzt sind, so werden sie in der Praxis oft nicht nur punktuelle Gültigkeit haben, sondern innerhalb eines gewissen Zeitraumes konstant sein. So ändert sich der Kaufpreis nicht stetig. Eine Schuld besteht über eine gewisse Zeit. Ein Vertrag hat eine Geltungsdauer etc. Wird nun ein Faktum innerhalb eines Vorgangs ermittelt und soll dieser Wert an das System weitergegeben werden, so sollte das System nicht nur punktuelle Angaben erhalten, sondern über den gesamten Gültigkeitszeitraum informiert werden. Andernfalls kann es auch nur Anfragen beantworten, die genau den bekannten Zeitpunkt betreffen. Bei der Übermittlung eines Wertes sind also in zwei Parametern **Anfang und Ende des Gültigkeitszeitraumes** zu übermitteln. Ebenso wie bei der Anfrage nach einem Faktum muß auch bei der Übermittlung eines Faktums die Angabe des Zeitraumes optional sein. Fehlt eine Angabe über den Beginn des Zeitraumes, so wird davon ausgegangen, daß dieser vor dem für den Fall relevanten Zeitraum liegt, fehlt die Angabe des Endes, so liegt es nach dem relevanten Zeitraum. Fehlen beide Daten, so hat das Faktum für den gesamten Zeitraum Gültigkeit.

---

[268] Vgl. S. 35.

[269] Vgl. S. 36.

[270] Vgl. S. 34.

b) Streit über Fakten

Ein wohl typisch juristisches Problem ist die Möglichkeit, daß Werte oder Beziehungen von Objekten im Streit stehen. Dieser Streit rechtfertigt oft erst den Bedarf für eine gerichtliche Klärung eines Sachverhaltes. Beteiligte an einem juristischen Verfahren können grundsätzlich alles streitig stellen. Etwa kann bestritten sein, daß eine Ehe zustande gekommen ist. Andererseits kann strittig sein, wann die Ehe geschlossen wurde. Es ist auch vorstellbar, daß eine Ehe zweimal geschlossen wurde und strittig ist, welche der Eheschließungen gültig ist. Es kann das Geburtsdatum einer Person strittig sein, ja sogar deren Existenz mag angezweifelt werden. Eine besondere Variante ist auch, daß zwei Fakten bestritten werden, deren Bestreiten in einem logischen Widerspruch stehen. So kann die Existenz einer Ehe bestritten werden und gleichzeitig (für den Fall des Bestehens) das Datum der Eheschließung. Aus diesen unterschiedlichen Sachlagen ergibt sich

Regel 8

> Es sind mehrere Aussagen über Fakten möglich. Zu jeder dieser Aussagen muß die Quelle feststellbar sein.

Es ist also sowohl bei der Anfrage nach Fakten als auch bei deren Übermittlung notwendig, daß die Quelle des Faktums angegeben wird. Dies kann etwa über die optionale Hinzufügung eines weiteren Parameters geschehen. So kann beispielsweise eine Relation im juristischen Sinne, d.h. eine Gegenüberstellung unterschiedlicher Parteivorträge, gerade darauf aufbauen, die unterschiedlichen Fakten der unterschiedlichen Quellen zu erfragen. Falls keine Quellenangabe erfolgt, muß eine Standardquelle verwendet werden. Bei der Übermittlung signalisiert dies, daß das Faktum unstreitig ist. Bei der Anfrage wird wieder auf die aktuellste Information zurückgegriffen.

c) Zeitliche Änderung von Aussagen über Fakten

Ein weiteres typisch juristisches Problem ergibt sich aus einer Kombination der beiden vorgenannten Besonderheiten: Jede Aussage über einen Wert oder eine Beziehung kann sich im Laufe der Zeit ändern. Einer Partei fällt beispielsweise im Laufe eines Prozesses ein neuer Gesichtspunkt ein. Aufgrund von neuen Beweisen stellen die Parteien bestimmte Tatsachen unstreitig etc.

Regel 9

> Es sind mehrere Aussagen über Fakten auch aus derselben Quelle möglich. Zu jeder dieser Aussagen ist neben der Quelle auch der Zeitpunkt der Aussage feststellbar.

Werden also Fakten an das System übermittelt, so müssen neben dem Geltungszeitraum des Faktums selbst und der Bezugsquelle auch der Geltungszeitraum der Aussage übermittelt werden. Dies wird sich in den meisten Fällen auf die Angabe des Zeitpunkts beschränken, an dem die Aussage behauptet wurde, da der Zeitpunkt, an dem die Aussage geändert wird, regelmäßig in der Zukunft liegt. Hieraus ergibt sich die Notwendigkeit, daß eine geänderte Aussage explizit eine Annullierung der alten Aussage beinhaltet. Dies ist ein Ausfluß der theoretischen Möglichkeit, auch sich in den Voraussetzungen widersprechende Behauptungen aufstellen zu dürfen.

Angenommen die Partei *Peter* hat bis zum Auftreten der Zeugin *Z* am 21.3.94 behauptet, diese nie geheiratet zu haben. Danach stellt *Peter* die Frage nach der Heirat unstreitig, behauptet jedoch, geschieden worden zu sein. Es reicht nun nicht aus, die ursprüngliche Aussage $\neg\exists x\big(x = Ehe\big(P,\ Z\big)\big)$ durch die Aussage $S = Scheidung\big(E\big)$

zu ändern. Vielmehr muß gleichzeitig mit der Aussage, daß nun eine Ehe angenommen wird, $E = Ehe(P, Z)$, aufgestellt werden.

d) Beweismittel

Die allgemeine Möglichkeit, jedes Faktum zu bestreiten führt zu der Notwendigkeit, jede Behauptung auch beweisen zu können. Dabei reicht oft nicht nur die Angabe eines Beweismittels aus, sondern es müssen mehrere Beweise angegeben werden können[271].

Die Besonderheit bei Beweismitteln im Gegensatz zu anderen Faktoren einer Aussage etwa der Quelle oder dem Geltungszeitraum liegt darin, daß das Beweismittel selbst wiederum Gegenstand der juristischen Behandlung ist. So werden Beweismittel juristisch klassifiziert: Zeugen (§ 373 ff. ZPO), Sachverständige (§ 402 ff. ZPO), Augenschein (§ 371 f. ZPO), Urkunden (§ 415 ff. ZPO) und Parteivernahme (§ 445 ff. ZPO). Beweismittel können zulässig oder unzulässig sein. Sie können sich als wirkungslos herausstellen, wenn etwa der Zeuge oder Sachverständige etwas anderes sagt, als erwartet wurde etc.

Dabei ist es nicht sinnvoll, bei der Betrachtung der Beweismittel auf die derzeitige prozeßrechtliche Lage abzustellen. Dies wäre der Fall, wenn das System die oben genannten Beweismittel quasi immanent vorsehen würde. Vielmehr befindet sich aber diese genauso im Fluß, wie die übrigen rechtlichen Belange. Es können neue Beweismittel möglich oder alte verboten werden. Entsprechend muß die Bezeichnung als Beweismittel letztlich einen ähnlichen Status aufweisen, wie alle übrigen Aussagen. Sie stellt allerdings nicht eine Relation $Beweis(B, X)$ zwischen dem Beweismittel B und einem anderen Objekt X dar, sondern sie setzt eine Aussage, also einen konkreten Wert oder eine Beziehung, in Beziehung zu einem Beweisobjekt. Der Ausdruck $Urkundsbeweis(U, Ehe(E, M, F))$ könnte dahingehend interpretiert werden, daß U eine Urkunde zum Beweis der Existenz einer Ehe zwischen M und F darstellt. Dies bedarf allerdings bei der konkreten Implementierung einer syntaktischen Schärfung. Nach der bisherigen Syntax beweist im vorliegenden Ausdruck U das Objekt E, nicht jedoch dessen Existenzwert oder die Aussage über die genannte Beziehung. Deshalb bedarf es einer eigenen

Regel 10

> Zu jeder Aussage über ein Faktum ist die Angabe von einem oder mehreren Beweisen möglich. Die Angabe eines Beweises ist selbst eine rechtlich relevante Aussage.

Der anfragende Vorgang erhält so also die Möglichkeit, zu jedem Faktum einen oder mehrere Beweise abzufragen. Jeder Vorgang kann zudem einem Wert einen neuen Beweis zuordnen oder einen Beweisantritt aus rechtlichen oder tatsächlichen Gründen in Frage stellen. Dies eröffnet nicht nur eine lückenlose Versorgung eines Vorgangs etwa zur Schriftsatzgenerierung mit relevanten Fakten, es erlaubt auch den automatischen Nachweis von Beweismitteln.

e) Zusammenfassung der juristischen Regeln

Aus der Gesamtbetrachtung juristischer Besonderheiten ergibt sich: Es müssen zu einem Objekt beliebig viele Werte verfügbar sein, zu denen Angaben über den Zeitraum der Gültigkeit des Wertes (Regel 7), die Quelle (Regel 8) sowie den Zeitpunkt der Aussage (der Quelle), über den Wert (Regel 9) und über die Beweismittel für jede Aussage (Regel 10) abgelegt werden. Für die Angabe von Beziehungen von Objekten unterein-

[271] Vgl. S. 71.

ander gilt im Prinzip ähnliches. Es müssen mehrere Wahrheitswerte für unterschiedliche Zeiträume auch aus unterschiedlichen Quellen für das Bestehen oder Nichtbestehen einer Beziehung möglich sein. Hieraus folgt, daß auch die Angabe, eine Beziehung bestehe nicht (Bestreiten einer Beziehung), als Information positiv mit allen Zusatzinformationen verfügbar sein muß.

5. Architektur

Beim abstrakten Entwurf einer Architektur der Schnittstelle lassen sich die Darstellungen zu den Datenbankarchitekturen[272] verwerten. Die dort beschriebenen Architekturmodelle können auf einer abstrakteren Ebene auf folgende Fragen zurückgeführt werden:

1. Soll ein Austausch der Daten zweier Prozesse direkt oder über eine zentrale Station erfolgen?
2. Sollen die an einem Datenaustausch beteiligten Prozesse synchron oder asynchron arbeiten?

a) Kommunikationswege

In einer Architektur, bei der zwei Prozesse ihre Fakten direkt miteinander austauschen, tritt ein Prozeß P1, der Fakten benötigt, die ein Prozeß P2 ermitteln kann, mit letzterem direkt in Kontakt. Dabei fragt er bei P2 nach den Fakten in der oben beschriebenen Weise. Hierzu muß es ein Regelwerk geben, nach dem P1 erkennt, bei welchem Prozeß er die benötigten Fakten am günstigsten erhalten kann. Ein Problem kann sich besonders dann stellen, wenn dasselbe Faktum von mehreren Prozessen ermittelt werden könnte. P1 muß dann bei allen Prozessen anfragen, ob das gesuchte Faktum verfügbar oder von dem Prozeß errechenbar ist. Wenn er von unterschiedlichen Prozessen unterschiedliche Werte zurückerhält, so muß er selbst entscheiden, welcher der Werte für ihn relevant ist.

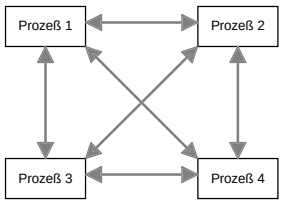


Abbildung 18: Kommunikationswege bei direkter Kommunikation von Prozessen untereinander.

Da prinzipiell jeder Prozeß mit jedem in Verbindung treten kann, ergeben sich bei *n* Prozessen

$$\sum_{x=1}^{n-1} x = \frac{n \cdot (n - 1)}{2}$$

potentielle Verbindungen. Bei der Hinzunahme eines *n+1*ten Prozesses werden entsprechend *n* neue Verbindungen möglich. Die Anzahl der Verbindungen steigt also quadratisch mit der Anzahl der Prozesse. Dieser Systemaufbau wird mit steigender Anzahl der beteiligten Prozessen immer unhandlicher und, ganz unabhängig von der Art der technischen Realisierung, mit Sicherheit immer langsamer.

Haben hingegen alle Prozesse lediglich einen zentralen Ansprechpartner S, so entfällt die Komplexität der möglichen Verbindungen. Bei Hinzunehmen eines weite-

[272] S. 76.

ren Prozesses wird eine weitere Verbindung benötigt. Der Anstieg ist in diesem Fall lediglich linear.

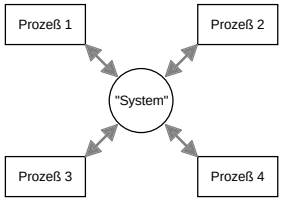


Abbildung 19: Kommunikationswege bei der Kommunikation von Prozessen über ein zentrales System.

Bei einer Anfrage entscheidet der zentrale Ansprechpartner (das System) darüber, bei welchem der beteiligten Prozesse er das Datum zu ermitteln versucht. Bei einer Kollision von Fakten entscheidet wiederum das System, welches der Fakten es an den fragenden Prozeß weiterleitet.

Das hier zu entwickelnde Datenaustauschsystem wird entsprechend von einer zentralistischen Architektur ausgehen.

b) Synchronisation

Bei einer synchronen Datenübertragung müssen ähnlich wie beim Telefon alle an der Datenübertragung beteiligten Prozesse gleichzeitig aktiv sein. Verwendet man eine zentralistische Architektur so fragt ein Prozeß P1 bei dem zentralen System nach einem Faktum. Dieses aktiviert den Prozeß P2 und fordert ihn auf, das Faktum dem System zu übermitteln. Das System leitet das Faktum weiter an P1 und deaktiviert evtl. P2. Benötigt ein Prozeß P3 dasselbe Faktum, so wiederholt sich das Prozedere von neuem.

Eine asynchrone Datenübertragung arbeitet wie die Briefpost mit einem Zwischenspeicher. Dieser ermöglicht es, Daten an ein zentrales System auch ungefragt zu übertragen. Das System legt die Daten ab und leitet sie auf Anforderung an jeden Prozeß weiter. Der Datentransfer erfolgt nur dann quasi synchron, wenn die gesuchten Fakten dem System S noch unbekannt sind. S startet dann seinerseits einen neuen Prozeß, der für die Ermittlung der gewünschten Fakten verantwortlich ist. Dieser Prozeß übermittelt die Fakten an das zentrale System, welches sie sofort an den anfragenden Prozeß weiterleitet. Um den Vorgang auch hier möglichst effektiv zu gestalten, werden auch diejenigen Daten, die erst auf eine Anfrage ermittelt wurden, vom System zwischengespeichert.

Die asynchrone Kommunikation ist insbesondere in Verbindung mit einer zentralistischen Architektur von Vorteil, da sie einmal durchlaufene Vorgänge konserviert. Dies verringert zwangsläufig die Laufzeiten bei einer Anfrage, da ein weiterer Prozeß nur dann angestoßen werden muß, wenn ein Faktum dem System noch nicht bekannt ist. Da die Daten nicht jedesmal neu bei den verantwortlichen Prozessen abgefragt werden, bleiben sie einerseits konsistent, andererseits besteht die Gefahr, daß sie nicht auf dem neuesten Stand sind. Jeder Vorgang hat somit bei diesem Verfahren die Verantwortung dafür, daß der Datenpool des Systems auf dem aktuellen Stand ist.

Die Architektur des hier entwickelten Modells geht von der Existenz eines zentralen Ansprechpartners für alle Prozesse aus. Die Kommunikation erfolgt regelmäßig asynchron. Das zentrale System speichert alle eingehenden Fakten. Eine Anfrage beantwortet es prioritär mit den gespeicherten Daten. Falls keine Daten vorhanden sind, versucht das System diese durch einen weiteren Prozeß zu ermitteln.

Teil III. Realisierung

Der folgende Teil beschreibt die konkrete Umsetzung des entwickelten Modells in eine lauffähige Prototypenversion. Der Prototyp dient zum Test des Verfahrens anhand einiger einfacher Beispiele. Hierzu war das Schnittstellensystem selbst sowie einige Anwendungen zu entwickeln, die auf die benannte Schnittstelle zugreifen.

A. Terminologie

Um bei der praktischen Umsetzung auf eine gewohnte Terminologie zurückgreifen zu können, werden in diesem Abschnitt Prozesse je nach Ausgestaltung auch als *Programme* oder als *Anwendungen* oder *Module* bezeichnet.

Ein **Programm** ist ein selbstablaufender Code zur Lösung einer meist komplexen Aufgabe. Ein Programm ist auch dann selbstablaufend, wenn es auf die Funktionen des zugrundeliegenden Betriebssystems zugreift.

Die Begriffe **Anwendung** und Programm werden teilweise synonym verwendet. Im folgenden ist eine Anwendung meist komplexer als ein Programm. Dies kann durch eine konkrete Anpassung eines Programms an eine individuelle Aufgabe oder durch das planvolle Zusammenarbeiten mehrerer Programme zur Aufgabenlösung erfolgen. Ein typisches Beispiel für eine Anwendung ist ein Kalkulationsblatt etwa zur Zinsberechnung. Es setzt auf ein Tabellenkalkulationsprogramm auf und konkretisiert dessen Aufgaben im Sinne der entsprechenden Speziallösung. Auch ein ganzes System zur Organisation einer Kanzlei kann selbst dann als Anwendung bezeichnet werden, wenn es aus mehreren auch selbständig ablauffähigen Elementen besteht.

Ein **Modul** ist eine abgegrenzter Teil eines Programmes, der zur Lösung einer definierten Teilaufgabe dient. Module sind meist nicht selbst ablauffähig, sondern werden in ein oder mehrere Programme eingebettet. Ein typisches Beispiel wäre ein Modul zur Brutto-Netto-Umrechnung des Einkommens. Es kann in Unterhaltsprogrammen genau so eingesetzt werden wie in dezidierten Einkommensteuerprogrammen.

Der Unterschied zwischen Programm, Anwendung und Modul ist besonders aufgrund der hohen Integration moderner Betriebssysteme und programmübergreifender Programmiersprachen fließend. Allgemein kann vom Modul über das Programm zur Anwendung eine Steigerung der Komplexität angenommen werden. Zentrales Merkmal eines Moduls ist seine Einbettung in einen komplexeren Zusammenhang, wohingegen eine Anwendung meist mehrere Module vereint, um eine vielschichtige Aufgabe zu lösen.

B. Auswahl der Arbeitsumgebung

Der hier entwickelte Prototyp dient der Demonstration der Anwendbarkeit des beschriebenen Konzeptes und der Konkretisierung einiger Detailfragen. Er liegt jedoch noch fern ab von einem auslieferbaren Produkt. Bei der Wahl einer Arbeitsumgebung zur Realisierung eines solchen Prototypen überwiegen pragmatische Kriterien. So wurde weder eine wissenschaftliche Marktanalyse durchgeführt, um die Verbreitung des Betriebssystems und der verfügbaren Anwendungen zu erforschen, noch wurden alle Betriebssysteme auf ihre optimale Eignung für das geplante Vorhaben geprüft. Statt dessen wurden die faktisch verfügbaren Betriebssysteme anhand von Ausschlußkriterien beleuchtet, die sich aus den Anforderungen insbesondere an den angestrebten Arbeitsstil und an die Systemarchitektur ergaben.

Bei der Auswahl der Programmierumgebung wurde vornehmlich die Frage der Aufgabenangemessenheit gewertet. Insgesamt wurde davon ausgegangen, daß bei

einer juristischen Arbeit eine Minimierung softwaretechnischer Probleme und eine Optimierung der Lesbarkeit des Codes anzustreben ist.

1. Betriebssystem

Die Entscheidung über ein Betriebssystem scheint vielfach eher als Glaubenssache angesehen zu werden, als mit Argumenten belegbar zu sein. Aufgrund der praktischen Ausrichtung des Themas kamen nur marktübliche Betriebssysteme in Frage. Dies sind vor allem DOS oder Windows[273] von Microsoft, OS/2 von IBM, System 7 von Apple sowie die unterschiedlichen UNIX-Betriebssysteme. Ältere Betriebssysteme wie CPM oder GEM oder wissenschaftliche Systeme wie EUMEL scheiden aufgrund ihrer nachlassenden Verbreitung und ihres veralteten Funktionsumfanges von vornherein aus.

Die mit dem konzipierten System angestrebte Arbeitsweise geht von einer hohen Integration inhomogener Programme aus. Es wird die Möglichkeit erwartet, daß mehrere Programme quasi gleichzeitig arbeiten können[274]. Fehlt etwa bei der Verfassung eines Schriftsatzes in der Textverarbeitung eine Angabe, so muß es möglich sein, eine weitere Anwendung aufzurufen, um das fehlende Datum zu ermitteln und an die Textverarbeitung weiterzuleiten. Dieses Vorgehen erfordert vom Betriebssystem zwingend zwei Kriterien:

1. Das Betriebssystem muß das (quasi) gleichzeitige Ablaufen mehrerer Anwendungen ermöglichen.
2. Das Betriebssystem muß Normen für den Austausch von Daten zur Verfügung stellen.

Beide Forderungen werden von dem wohl verbreitetsten Betriebssystem **DOS** nur unzureichend erfüllt[275]. Zwar lassen sich von einer Anwendung aus andere Programme temporär starten. Ein Multitasking-Betrieb, der das Wechseln zwischen mehreren Anwendungen ermöglicht, ist dann jedoch nicht explizit vorgesehen. Insbesondere bietet DOS jedoch keine ausreichenden Datentransferoptionen. Eine Entwicklung solcher Optionen ist denkbar. Sie sprengt jedoch den Rahmen einer juristischen Arbeit. Die übrigen Betriebssysteme erfüllen diese Kriterien besser. Sie unterscheiden sich in den geforderten Punkten so unwesentlich, daß ein Abweichen von dem wohl marktführenden und dem Autor gewohnten Betriebssystem MS-Windows nicht geboten schien: MS Windows erlaubt den gleichzeitigen Ablauf mehrerer Programme mit der Möglichkeit, frei zwischen allen Programmen zu wechseln. Es stellt zudem mit DLLs, DDE und OLE mehrere sehr moderne Möglichkeiten des Datenaustauschs zwischen laufenden Applikationen zur Verfügung.

Die übrigen erwähnten Betriebssysteme bieten im Bereich des Multitaskings z.T. modernere Realisationen[276]. Im Bereich der Integration von Anwendungen wiederum, bietet Windows sehr moderne Mechanismen, die in den übrigen Betriebssystemen teilweise noch nicht so ausgereift sind. Auch die hohe Verbreitung von

---

[273] Die oft gestellte Frage, ob Windows ein Betriebssystem ist, ist mit der Einführung von Windows 95 als Nachfolger von Windows 3.1 weitgehend verstummt.

[274] Vgl. S. 101 ff.

[275] *Hoffmann*, jur-pc 91, 901 ff. realisiert dies wie viele andere im wesentlichen durch Austausch mittels Zwischendateien. Einen interessanten Integrationsansatz unter DOS liefert RAMANDATA, s. *Schedel*, jur-pc 93, 2212 ff. Hier können Berechnungsergebnisse über Felder in Texte eingebunden werden.

[276] Die Entwicklung des Systems erfolgte noch unter Windows 3.1 mit kooperativem Multitasking. Die Programme sind auch unter Windows 95 lauffähig, allerdings aufgrund der unveränderten 16-BIT-Programmarchitektur weiterhin nicht mit präemptivem Multitasking.

MS-Windows und die Vielfalt der implementierten Anwendungen sowohl allgemeiner[277] als auch spezifisch juristischer Art spricht für die Wahl dieses Betriebssystems. Zudem sind neben den gängigen Standardprogrammen einige sehr leistungsfähige und komfortable Programmierwerkzeuge verfügbar.

2. Programmierwerkzeug

Einem ähnlich pragmatischen Ansatz wie bei der Auswahl des Betriebssystems folgte die Wahl der verwendeten Programmierwerkzeuge. Im Bereich der Compiler stehen für Windows ein Pascal-Compiler[278] und mehrere C bzw. C++ Compiler zur Verfügung. Obwohl die Standardprogrammiersprache unter Windows sicherlich C ist, wurde für die im Rahmen der Arbeit zu compilierenden Programme oder Module Pascal eingesetzt. Dies erfolgte zunächst aufgrund der besseren Kenntnis des Autors in diesem Bereich. Die Verwendung von Pascal ist unbedenklich, da Pascal gegenüber C auch unter Windows keinerlei funktionale Einschränkungen besitzt. Vielmehr ist die Programmierung aufgrund der objektorientierten Erweiterungen einfacher als in C[279]. Zudem ist der Code beim Abdruck im Anhang in dieser Arbeit für den Laien besser lesbar, da Pascaldie Syntax durchgehend Klartextausdrücke verwendet. Entsprechend wird Pascal oftmals als Lehrsprache eingesetzt[280].

Insbesondere für die in der späteren Entwicklungsphase erstellten Anwendungsbeispiele wurde Visual BASIC eingesetzt. Diese Programmiersprache bietet im Gegensatz zur Urform von BASIC alle Strukturmerkmale einer modernen Compilersprache. Die Syntax ist der Pascal–Syntax sehr ähnlich, so daß auch diese Programmbeispiele leicht lesbar sind. Gegenüber den verfügbaren Compilersprachen arbeitet BASIC auf einem fast völlig betriebssystemunabhängigen Abstraktionsniveau. Insbesondere die Gestaltung der Programmoberfläche und der Zugriff auf die Elemente der Benutzerschnittstelle wird erheblich vereinfacht. Unregelmäßigkeiten und typische Probleme der Windowsprogrammierung sind in BASIC oftmals sehr elegant gelöst. Dennoch bietet BASIC optional den Zugriff auf die volle Windows-Funktionalität. Es ist deshalb ein nahezu ideales Werkzeug für die Entwicklung und Präsentation von Prototypen. Mit der Veröffentlichung von Windows 95 wurden die Programmbeispiele auf die 32-Bit Version von Visual BASIC 4.0 portiert. Windows 95 bietet eine objektorientierte Benutzeroberfläche. Zudem kommt dem hierarchischen Baum als Zugangsweg zu Informationen eine wesentlich größere Bedeutung zu als in den älteren Versionen. Beide Ansätze werden durch das Betriebssystem und die Entwicklungsumgebung optimal unterstützt. Dieses Bedienungsmodell, Koppelung von Informationen an Objekte und Organisation der Objekte in hierarchischen Bäumen, kommt dem hier gewählten Ansatz für den Zugang und die Behandlung von Fakten eines Falles sehr nahe. Die Portierung der Anwendung führt entsprechend zu einer höheren Integration der Anwendungen in die gewohnte Arbeitstechnik des Anwenders.

Trotz des prädikatenlogischen Grundansatzes fanden logische Sprachen wie etwa Prolog oder LISP[281] keine Anwendung. Dies lag zum einen an der schlechten Ver-

---

[277] Allen voran MS-Word für Windows als Standardtextverarbeitung (*Hoffmann*, PC-Praxis, S. 161).

[278] Inzwischen wird der Compiler der Firma *Borland* unter dem Namen *Delphi* angeboten. Hinter der stark visuellen Programmierumgebung verbirgt sich weiterhin die für objektorientierte Programmierung erweiterte Sprache *Turbo–Pascal*.

[279] C++ war zu Beginn der Entwicklungen noch nicht allgemein verbreitet.

[280] Vgl. *Bund*, S. 208.

[281] Einen kurzen Überblick über die Sprachen bietet *Bund*, S. 201 ff.

fügbarkeit der Werkzeuge und zum anderen daran, daß sie keinen echten Nutzen erwarten ließen. Die wahren Stärken dieser Systeme liegen in der Abarbeitung von logischen Regeln. Das zu entwickelnde System arbeitet jedoch nicht regelbasiert. Vielmehr soll es lediglich einen kleinen Teilbereich, die Verwaltung der Fakten, abdecken. Für diesen Bereich sind aber gerade Logiksprachen gar nicht optimiert.

Das System wurde unter MS-Windows 3.1 und später Windows 95 in den Sprachen Pascal und Visual–BASIC entwickelt. Hierfür sprachen im wesentlichen pragmatische Gründe. Für den Leser am wichtigsten dürfte der gut lesbare Beispielscode sein.

C. Realisation von systematischen Einzelfragen

Der folgende Abschnitt beschäftigt sich mit Einzelfragen, die bei der Umsetzung des beschriebenen Konzeptes in ein konkretes Programm zu lösen sind. Dabei wird auch hier soweit dies möglich ist auf die besonderen Erfordernisse des gewählten Betriebssystems verzichtet.

1. Architektur

Unter der Architektur einer Anwendung wird das Konzept verstanden, welches die Aufgliederung der Anwendung in einzelne Module und deren Zusammenspiel bezeichnet. Im folgenden wird eine Architektur entwickelt, die sich zwar an den Möglichkeiten der Arbeitsumgebung MS-Windows orientiert, ähnlich aber auch in anderen Umgebungen realisierbar ist.

a) Einleitung

Aufgrund der Spezifikation des Modellentwurfs wird davon ausgegangen, daß alle Prozesse mit einer einheitlichen, zentralen Instanz korrespondieren. Die Kommunikation zwischen zwei Prozessen über diese Instanz erfolgt asynchron, d.h. es ist nicht notwendig, daß beide Prozesse gleichzeitig ablaufen[282]. Das zentrale Kommunikationssystem stellt statt dessen zur Datenspeicherung einen Puffer zur Verfügung. Das Schaubild zeigt das Prinzip dieser Architektur:

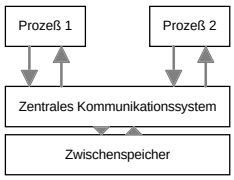


Abbildung 20: Schematische Darstellung des beschriebenen asynchronen, zentralen Kommunikationssystems.

Das System ähnelt damit stark der in der Datenbanktechnik viel diskutierten Client-Server-Architektur[283]. Für die Realisierung der einzelnen Kommunikationsvorgänge stehen mehrere Wege offen, die im Folgenden kurz skizziert werden sollen.

b) Kommunikationsmöglichkeiten von MS-Windows[284]

Dieser Exkurs beschreibt die konkreten Möglichkeiten des Austauschs von Daten, wie sie unter MS-Windows realisiert sind.

---

[282] Vgl. S. 101.

[283] S. S. 76.

[284] *Blankenburg, van Luit, Weihermüller*, Interprozeßkommunikation und EDV-Einsatz am Juristenarbeitsplatz, im Tagungsband zum 2. Deutschen EDV-Gerichtstag, S. 46 ff.

aa) Direkter Dateizugriff

Die einfachste Form der Definition einer zentralen, asynchronen Schnittstelle ist die Festlegung eines einheitlichen Dateiformates. Ein derartiges Dateiformat muß die geforderten Mechanismen zur Beschreibung eines Datums abbilden. Eine Anwendung die ein Datum benötigt, muß den Dateinamen der aktuellen Datei kennen. Sie öffnet die Datei und greift mit programminternen Mechanismen auf die dort gelagerten Daten zu. Das Programm liest die benötigten Fakten und schreibt die ermittelten Fakten in die Datei zurück. Am Ende der Zugriffe, spätestens aber wenn eine andere Anwendung vom Benutzer den Fokus erhält[285], schließt das Programm die Dateien und ermöglicht so anderen Programmen den Zugriff.

Dieses Verfahren ist passiv. Es gibt also kein zentrales Modul oder Programm, das im Fall des Fehlens von Daten aktiv werden könnte. Die Möglichkeit, unter diesen Umständen eine andere Anwendung aufzurufen, um die fehlenden Daten zu ermitteln, bleibt dem Programm selbst überlassen. Ebenso ist das Programm für den korrekten Umgang mit dem Datenformat verantwortlich.

Eine Beschränkung der Implementation auf die Beschreibung eines Dateiformates erfüllt die aufgestellten Ansprüche nicht. Es wird keine automatisierte Umsetzung einer abstrakt juristischen Problemstellung in eine konkrete programmtechnische Umsetzung erreicht. Vielmehr ist jedes Programm selbst dafür verantwortlich, das juristische Problem erst in einen Zugriffsalgorithmus auf das Datenformat umzusetzen. Desweiteren fehlt es an einer zentralen Koordination der Kommunikation mehrerer Anwendungen. Jede Anwendung bleibt bei der Erfassung der notwendigen Fakten letztlich auf sich gestellt.

bb) Dynamische Funktionsbibliotheken (DLLs)

Sogenannte DLLs[286] sind eine Windows-spezifische Form von Funktionsbibliotheken, die jedes Programm zur Laufzeit einbinden kann. Auf diese Weise kann der Leistungsumfang der meisten Windowsprogramme dynamisch um die in einer DLL verfügbare Funktionalität erweitert werden. Dies gilt für Standardprogramme genauso wie für Applikationen, die spezifisch auf die Verwendung einer bestimmten DLL zugeschnitten wurden. (Auch in allen anderen Betriebssystemen können Programmodule zur Laufzeit eines Programms eingebunden werden. Die Standardisierung ist dabei jedoch unterschiedlich stark ausgeprägt, so daß die Möglichkeiten auch unterschiedlich genutzt werden.)

DLLs bieten wiederum zwei unterschiedliche Möglichkeiten, Daten zwischen Programmen auszutauschen:

aaa) Gemeinsam genutzter Datenbereich im Arbeitsspeicher

Bei Windows 3.1 DLLs sind alle globalen Daten für alle gleichzeitig operierenden Anwendungen im Zugriff. Verwenden also zwei Anwendungen gleichzeitig dieselbe DLL, so werden - anders als bei mehrfach ablaufenden Programmen - die Daten nicht für jedes Programm getrennt gespeichert. Vielmehr arbeiten beide Programme mit denselben Daten und können somit die Daten auch für die andere Anwendung verändern.

In den neuen 32-Bit Versionen von Windows ist diese Funktion verändert worden. Hier werden Daten für mehrere auf eine DLL zugreifende Programme standardmäßig getrennt verwaltet. Da aber explizit als *teilbar* deklarierte Daten weiterhin

---

[285] Alle Eingaben, die ein Benutzer macht, beziehen sich auf die Anwendung mit dem *Fokus*.

[286] Dynamic Link Library.

gemeinsam von mehreren Programmen verwendet werden können, hat sich die Funktionalität entsprechend auch bei höheren Betriebssystemvarianten nicht grundlegend verändert. Sie bleibt aufgrund der meist garantierten Abwärtskompatibilität auch generell erhalten.

bbb) Funktionen zum abstrakten Dateizugriff

Wie bereits bei der Beschreibung des ODBC-Modells erwähnt[287], können gemeinsam nutzbare Bibliotheken dazu verwendet werden, abstrakte Funktionen zum Zugriff auf zentrale Datenbanken verfügbar zu machen. In einem solchen Fall spricht man von einer Anwendungs-Programmier-Schnittstelle, kurz API[288].

Ein Anwendungsprogramm verwendet die Funktionen einer API wie eigene Funktionen. Es kann zum Datenaustausch über diese Funktionen auf eine gemeinsam genutzte Datei zugreifen. Der Unterschied zu dem oben beschriebenen direkten Dateizugriff besteht in der Abstraktion der verfügbaren Funktionen. Die Anwendung muß bei dem Dateizugriff keinerlei Kenntnisse von dem physikalischen Aufbau der Datei haben. Da die Funktionen immer erst zur Laufzeit des Programms hinzugezogen werden, ist es sogar möglich, durch einen separaten Austausch der API das Dateiformat zu ändern, ohne daß sich für das eigentliche Programm eine Veränderung ergibt. Als Beispiel sollen zwei fiktive Funktionen gezeigt werden, die einen Datumswert in der vom Modell beschriebenen Weise anfordern bzw. übermitteln:

```
Declare Function GDatumsWert Lib "beispiel.dll"_
  (ByVal szAnfrage as String) As String
Declare Sub SDatumsWert Lib "beispiel.dll"_
  (ByVal szAnfrage as String, ByVal szWert as string)
```

Diese Passage bindet die Funktionen `GDatumsWert` und `SDatumsWert` aus der Bibliothek `BEISPIEL.DLL` in ein Visual- oder Word-BASIC-Programm ein. Das Präfix *G* steht dabei für *gib* (engl. *get*) und das Präfix *S* für *setze* (engl. *set*). Mit der Einbindung stehen die Funktionen in diesem BASIC-Programm zur Verfügung, falls die DLL auf dem verwendeten Computer verfügbar ist. Die folgende Zeile versucht nun das Geburtsdatum von Peter mit Hilfe der Funktion `GDatumsWert` zu erfragen:

```
szDatum$ = GDatumsWert ( "Geburtstag(Peter)" )
```

Die eigentliche Anfrage wird dieser fiktiven Funktion also als Zeichenkette übergeben. Damit wird eine hohe Flexibilität erreicht, da diese Zeichenkette frei von den syntaktischen Beschränkungen der verwendeten Programmiersprache ist. Deshalb kann eine Funktion auch so definiert werden, daß sie die bereits in der Spezifikation des Modells beschriebenen komplex verschachtelten Anfragen ermöglicht. Mit dieser Realisierung kann die Funktion dann etwa auch den Geburtstag des Kindes des Klägers erfragen wie das folgende Beispiel darstellt:

```
szDatum$ = GDatumsWert ( "Geburtstag(Kind(Kläger()))" )
```

Will man nun einen Wert an das System übermitteln, so verwendet man die hypothetische Funktion SDatumsWert wie folgt:

```
SDatumsWert ( "Geburtstag(Peter)", "10.10.1963" )
```

Hier wird also von der funktionalen Darstellung der Relationen abgewichen. Der Wert, der dem System übermittelt werden soll, wird als zweiter Parameter übergeben.

---

[287] S. S. 77.

[288] Applications Programmers Interface.

Im übrigen demonstriert das Beispiel, daß über eine Funktion einer Funktionsbibliothek ein Zugriff auf zentrale Daten, unabhängig von deren interner Organisation möglich ist. Die konkrete Umsetzung der Anfrage in einen Zugriff auf den Arbeits- oder Massenspeicher erledigt die Bibliothek. Die Anfrage ist fast identisch zu der abstrakt formulierten Beschreibung von Fakten im Konzeptentwurf[289]. Lediglich geringfügige Abweichungen ergeben sich aus der Syntax der zugrundeliegenden Programmiersprache.

Ein DLL–basiertes Systemist theoretisch auch in der Lage, beim Fehlen von Daten andere Anwendungen zu starten und bis zu der Übermittlung der fehlenden Daten zu warten. Hingegen gestaltetet sich eine Einflußnahme des Systems auf den Ablauf eines anderen Programms als schwierig. Damit eine Funktionsbibliothek einem anderen Programm Arbeitsanweisungen oder Daten quasi ungefragt übermitteln kann, muß das entsprechende Programm Vorkehrungen für eine derartige Fernsteuerung treffen. Dies kann etwa durch eine definierte programminterne Funktion geschehen, die anderen Programmen durch einen sogenannten Export zugänglich gemacht wird. Ein anderes Programm oder ein Modul kann diese definierte Funktion, sie wird auch Einstiegspunkt[290] genannt, ähnlich wie die Funktionen einer DLL als eigene einsetzen.

Eine derartige Funktion müßte jedoch in jedem Programm implementiert sein. Zwar ist es denkbar, daß Programme, die für den Umgang mit dem hier entwickelten System entworfen werden, einen einheitlichen Einstiegspunkt besitzen. Dies gilt jedoch nicht für Standardprogramme wie Tabellenkalkulationen, die von unabhängigen wenig fachspezifischen Softwarehäusern entwickelt werden. Die Fernsteuerung eines Programms mag jedoch dann von Interesse sein, wenn das zentrale System aufgrund eines Mangels an Daten ein anderes Programm startet und ihm gleichzeitig mitteilen möchte, welche Daten es zu ermitteln gilt. Da dieser Mechanismus ein zumindest interessanter Teil des beschriebenen Konzeptes darstellt, sollte hierauf nicht grundlos verzichtet werden.

Die Verwendung einer Funktionsbibliothek besitzt jedoch noch einen zweiten wohl erheblicheren Nachteil: Derartige Funktionen können lediglich in einen dynamischen Ablauf eines Programms implementiert werden. Ein solches Programm kann zwar auch ein Makro eines Standardprogramms sein. Dennoch ist eine statische Implementierung etwa in einem Formular oder einem Text oder Textbaustein einer Textverarbeitung nicht möglich. Es wäre somit ausgeschlossen, in einem Text ein Textfeld so zu gestalten, daß ohne Aufwand die beschriebenen Daten hier eingesetzt würden. Vielmehr müßte zum Einsetzen der Daten regelmäßig ein Programm gestartet werden, das die Daten aktiv in bestimmte Zellen einsetzt. Dieser Anwendungsfall etwa der Textbausteine kann jedoch nicht außer Acht gelassen werden. So mag es für den Benutzer von besonderem Interesse sein, daß er beim Aufruf eines Musterbriefes sofort in den variablen Positionen die für einen Fall korrekten Werte eingetragen erhält[291]. Die Variablen enthalten dann quasi verdeckt die vollständige Anfrage an das System, um das Faktum übermittelt zu bekommen.

---

[289] S. S. 89 ff.

[290] engl.: Entry Point Function.

[291] Ähnlich schon *Bauer*/*Lichtner*, S.108.

cc) Dynamischer Datenaustausch

Eine Alternative zur Verwendung von DLLs ist das Datenaustauschprotokoll DDE[292]. Die beiden vorab beschriebenen Mankos der Entwicklung einer API werden von dem unter Windows definierten Standard DDE behoben. DDE dient per se dem Austausch von Daten zwischen Anwendungen über eine betriebssystemeigene Kommunikationsleitung[293]. Es ist sozusagen die Windows-Hauspost für Anwendungen. Ähnliche Systeme sind auch in einigen anderen Betriebssystemen vorgesehen. Sie werden mit dem Oberbegriff *Interprozeßkommunikation*[294] bezeichnet. Das DDE-Protokoll wird hier in den Grundzügen beschrieben.

aaa) DDE-Protokoll

Windows ist ein Botschaften-gesteuertes Betriebssystem. Mit anderen Worten, die Kommunikation des Betriebssystems mit den laufenden Anwendungen erfolgt durch das Zusenden von Botschaften[295]. So erhält jede Anwendung eine Botschaft, wenn der Benutzer Eingaben macht, das Fenster verändert etc. Windows erlaubt es aber auch, daß sich Programme gegenseitig Botschaften zusenden oder daß ein Programm eine Botschaft global an alle anderen aktiven Programme schicken kann. Dabei können Standardbotschaften verwendet werden, die etwa ein Ereignis vortäuschen, es können aber auch Botschaften verschickt werden, die nur die beteiligten Programme verstehen (private messages).

DDE macht sich diese Möglichkeit zunutze, indem es eine bestimmte Folge von Botschaften definiert, die ein Programm absenden muß, um mit einem anderen Programm Daten auszutauschen. Da das Protokoll fester Bestandteil von Windows ist, sind dies Betriebssystemmeldungen. Das andere Programm muß auf die empfangenen Botschaften in einer definierten Form antworten. Eine derartige Definition einer Abfolge von gegenseitigen Aktivitäten wird als **Protokoll** bezeichnet.

Für die Beschreibung der Daten, die ausgetauscht werden, verwendet DDE die Bezeichner *Anwendung*[296], *Thema*[297] und *Element*[298]. **Anwendung** bezeichnet das Programm, mit dem ein anderes Programm Daten austauschen möchte. **Thema** ist ein Oberbegriff für das Thema des Datenaustauschs. Hier wird neben dem definierten Standardthema *System* regelmäßig ein Dateiname einer Anwendungsdatei verwendet. Es kann aber auch eine Konversation über ein beliebig definiertes Thema geführt werden. Das **Element** beschreibt den konkreten Wert, der übertragen werden soll. Bei einer Tabellenkalkulation sind dies etwa der Name oder die Koordinaten einer oder mehrerer Zellen. Auf Anfrage übermittelt die Tabellenkalkulation die Werte dieser Zellen.

Das anfragende Programm (der Client) schickt zur Einleitung der Konversation eine Botschaft an alle anderen Programme. Sie enthält die Bezeichnung der gesuchten Anwendung und des Themas. Ist die gesuchte Anwendung aktiv und betriebsbereit, bestätigt sie die Aufnahme der Konversation. Erhält der Client keine positive Antwort, versucht er die gesuchte Anwendung von sich aus zu starten und die Konver-

---

[292] DDE = Dynamic Data Exchange (Dynamischer Datenaustausch).

[293] *Gerblinger*, *Manfred* a.a.O.

[294] S. FN. 284.

[295] engl.: messages.

[296] engl.: Application.

[297] engl.: Topic.

[298] engl.: Item.

sation noch einmal einzuleiten. Schlägt dies wiederum fehl, bricht er seine Bemühungen ab und meldet dies dem Benutzer. Im Erfolgsfall beginnt die Konversation.

Das Programm, das Daten auf Anfrage übermittelt, wird als *Server* bezeichnet. Das folgende Schaubild zeigt eine einfache Anfrage eines Programms an das Kalkulationsprogramm EXEL mit der Bitte, die *Gesamtsumme* im Kalkulationsblatt *RECHNUNG.XLS* zu übermitteln. *Gesamtsumme* ist in diesem Fall also ein benannter Datenbereich des Kalkulationsblattes.

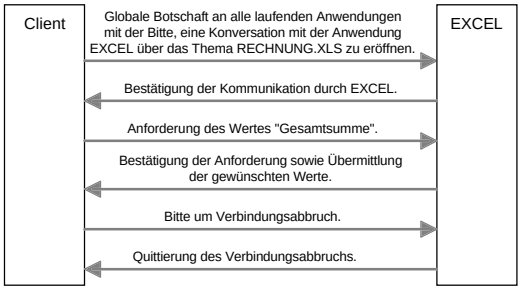


Abbildung 21: Schematischer Ablauf einer DDE-Anfrage nach einem Wert.

Die Übermittlung der Daten von einem Programm zum anderen erfolgt dadurch, daß diese in einen vom Betriebssystem zur Verfügung gestellten Speicherbereich geschrieben werden. Der Speicherbereich ist beiden Programmen zugänglich (global shared). Bei der Beantwortung der Anfrage erhält das fragende Programm in der Antwort die Adresse des Speicherbereichs. Das Programm kann sich die Daten dort abholen.

Das Protokoll enthält neben Verhaltensregeln bei geglückten Verbindungen auch Regeln für das Verhalten beim Fehlschlagen von Verbindungen oder bei nicht zu beantwortenden Anfragen. Zusätzlich zu der Möglichkeit, von einem Programm Daten einmalig zu erfragen, bestehen folgende weitere Optionen:

– Ein Programm kann eine Dauerverbindung herstellen, bei der der Server die Daten immer ungefragt übermittelt, falls diese sich geändert haben.
– Der Client kann ungefragt Daten an den Server übermitteln.
– Der Client kann beim Server Makros auslösen.

Mit Hilfe dieser Technik können nicht nur die Daten zweier Anwendungen auf demselben Stand gehalten werden, der Client kann den Server auch weitgehend fernsteuern.

bbb) DDE in einer konkreten Anwendung

Die vorhergehende Beschreibung zeigt die internen Vorgänge bei der Datenübertragung via DDE. In den meisten Fällen wird weder der Endanwender noch der Anwendungsprogrammierer mit diesen sehr komplexen Interna belastet. Lediglich bei der Implementation von DDE mit einer betriebssystemnahen Compilersprache wird eine Kenntnis des Protokolls erforderlich.

Der Anwender stellt eine DDE-Verbindung dadurch her, daß er den gewünschten Wert in der Serveranwendung kopiert und in der Clientanwendung mit der Menüanweisung *Verknüpfung einfügen* einfügt. Der Wert wird ab diesem Zeitpunkt in der Clientanwendung aktualisiert, wenn er in der Serveranwendung geändert wird. In der Textverarbeitung MS-Word beispielsweise erfolgt die Darstellung einer DDE-Verknüpfung mit Hilfe eines Feldes, das überall im Text plaziert werden kann. Dabei erstellt der Befehl *Verknüpfung einfügen* dieses Feld automatisch. Die Realisation des oben beschriebenen Beispiels sähe als Word-Feld folgendermaßen aus:

{DDEAUTO EXCEL RECHNUNG.XLS Gesamtsumme }

Nach dem Feldbezeichner *DDEAUTO*, der eine dauerhafte Verbindung aufbaut, folgt die Angabe der Anwendung, des Themas sowie des gesuchten Elements. Dieses Verfahren erlaubt nun auch die statische Plazierung juristischer Anfragen an einer spezifischen Position eines Textes, eines Formulars oder eines Kalkulationsblattes. So ließe sich die Anfrage nach dem Geburtsdatum des Kindes des Klägers als Feld in einer Textverarbeitung wie folgt formulieren:

{DDE FAKTEN "System"Geburtstag(Kind(Kläger(),)"}

Die Anwendung in diesem Fall heißt *FAKTEN*. Das Thema lautet wie im Standard *System*, unabhängig davon, welcher Fall gerade in Bearbeitung ist. Die Beschreibung des Elements entspricht der in der Spezifikation geforderten Faktenbeschreibung. Die Zelle zeigt in der Textverarbeitung nun immer den entsprechenden Wert für die Personen des als aktuell bezeichneten Falls an.

Für die Verwendung von DDE innerhalb von Programmabläufen lassen sich Funktionen entwickeln, die prinzipiell genauso arbeiten, wie die beschriebenen Funktionen der API. Hinzu kommt jedoch eine Normung für die Übermittlung von Daten an Applikationen und die Fernsteuerung über Makros. DDE ist insoweit flexibler als eine API und bedarf keiner individuellen Anpassung der Einzelanwendung. Diese müssen lediglich das Windows-Standardprotokoll unterstützen.

Beim Einsatz von DDE ist jedoch die Komplexität der Einbindung in betriebssystemnah programmierten Anwendungen in Betracht zu ziehen. Das sehr aufwendige Protokoll neigt zu Fehleranfälligkeit bei ungenau entwickelten Anwendungen. Diese ergibt sich vornehmlich daraus, daß das Protokoll asynchron arbeitet. Die Problematik soll hier jedoch nicht weiter vertieft werden. Sie führt in der Praxis dazu, daß DDE nicht in allen Windows–Anwendungen verfügbar ist. Vielmehr wird es in der Regel nur von komplexeren Programmender höheren Preisklasse in vollem Umfang unterstützt.

dd) OLE und OLE–2

Die Einbindung graphischer Elemente in sogenannte Verbunddokumente war die Ausgangsbasis für die Entwicklung von OLE[299]. Es ging also darum, beispielsweise eine Geschäftsgraphik aus einer Tabellenkalkulation in einen in der Textverarbeitung verfaßten Bericht einzubinden. Ähnlich wie bei DDE behält eine solche Graphik prinzipiell eine Verbindung zu dem Programm, mit dem sie ursprünglich erstellt wurde. Dabei können sich die Ausgangsdaten in einer eigenen Datei etwa des Kalkulationsprogramms (Object Linking) oder unmittelbar innerhalb der Daten des Hauptdokuments (Object Embedding) befinden. Ein Doppelklick des Anwenders

[299] Object Linking and Embedding.

auf die Graphik genügt, und das ursprüngliche Bearbeitungsprogramm wird gestartet, um die Graphik zu ändern.

Bei OLE werden Graphiken aber auch andere oftmals multimediale Elemente wie Klang und Video als Objekte innerhalb eines Dokuments eingebunden. D.h. es werden die originalen Daten des Ursprungsprogramms abgelegt. Handelt es sich nicht um eine originäre Graphik, so werden die Objekte durch ein Symbol angezeigt. Objekte besitzen in OLE Methoden, um angezeigt, abgespielt oder verändert zu werden. Fehlt eine Methode für die Anzeige am Bildschirm, so enthält das Objekt mindestens ein Abbild der Originaldaten in einem Windows-Standardformat wie BMP[300], WMF[301] oder RTF[302].

Das Programm, das ein Objekt aufnimmt, wird als Client bezeichnet. Es besitzt ein standardisiertes Verhalten, um die Methoden auszulösen und dem Benutzer verfügbar zu machen. So dient ein Doppelklick primär dem Abspielen des Objekts, falls dieses ein Video oder Klangobjekt ist. Anderenfalls ermöglicht der Doppelklick eine Bearbeitung des Objekts. Die Methode zur Anzeige wird aufgerufen, sofern das Objekt am Bildschirm oder auf dem Drucker dargestellt werden soll.

Die Kommunikation zwischen dem Clientprogramm und dem OLE-Serverprogramm erfolgt bei OLE(1) intern über DDE. Es wird jedoch eine API angeboten, die eine abstrakte, von DDE unabhängige Programmierung erlaubt.

OLE2 erweitert den Funktionsumfang erheblich. Die Objekte bieten einen nahezu beliebigen Umfang an Methoden. Das Serverprogramm kann sich im Client derartig einnisten, daß die Menüs und andere Kontrollelemente des Clients nun die Funktionen des Servers auslösen. Der Benutzer kann keinen Unterschied mehr zwischen Client und Server feststellen. Über das Konzept OLE–Automation kann sich eine Client–Anwendung nahezu jeder Funktion des Servers bedienen.

Das OLE Konzept paßt nicht für das hier geplante System. Es überläßt im wesentlichen der Datenquelle auch deren Repräsentation in der Clientanwendung. Dies ist hier jedoch gar nicht gewollt. Die hier ausgetauschten Daten sollen vielmehr unmittelbar weiterverarbeitet werden. Dies kann gerade auch durch individuelle Darstellungsformen des Clients geschehen. Weiterhin geht das System davon aus, daß die jeweiligen Daten im Regelfall nur von dem Datenlieferanten verarbeitet werden. Dies würde beispielsweise bedeuten, daß die Daten für eine Unterhaltsberechnung weiterhin nur von dem Unterhaltsberechnungsprogramm bearbeitet werden. Eine Weiterverarbeitung oder Änderung der Zwischenergebnisse durch andere Programme ist zunächst in diesem Konzept nicht vorgesehen. In einem Schriftsatz etwa wäre für die Präsentation der Ergebnisse und der Begründung das Berechnungsprogramm verantwortlich.

OLE basiert indes auf einer Architektur der Kommunikation, die bereits im Vorfeld als zu komplex bezeichnet wurde[303]. Hier müssen nämlich alle Anwendungen direkt miteinander verkehren. Eine zentrale Datenhaltung ist in dem Konzept nicht vorgesehen. Das DDE-Protokoll ist hier wesentlich flexibler. Zwar ist auch hier der Grundgedanke eine direkte Kommunikation der Datenlieferanten untereinander. Es

---

[300] Windows Bitmap: Bild mit Punkt für Punkt Speicherung.

[301] Windows Meta File: Graphisches Speicherformat, das eine Reihe von Aufrufen der Windows-Graphikschnittstelle GDI abspeichert. Das Format ermöglicht die Ablage von Vektor- und Bitmapgraphiken und kann sehr einfach angezeigt und gedruckt werden.

[302] Rich Text Format: Format zur Darstellung formatierter Texte.

[303] s.S. 101 f.

ist jedoch genau so denkbar, daß alle Programme die Daten an eine zentrale Stelle übermitteln und von dort beziehen. Aufgrund des Einsatzes allgemein genormter Datenformate hat jedes Programm die Möglichkeit, Daten weiterzubearbeiten und zu manipulieren.

Von den drei Möglichkeiten der Kommunikation, API mit DLLs, DDE und OLE erscheint DDE derzeit die flexibelste Variante[304]. Sie arbeitet abstrahiert von systemnaher Programmierung und bietet dennoch ein sehr hohes Maß an Flexibilität. Gegen den Aufbau einer API spricht die Notwendigkeit, daß alle zugreifenden Anwendungsprogramme darauf angepaßt werden müssen. Gegen den Einsatz des an sich moderneren OLE spricht letztlich die zugrundeliegende dezentrale Architektur, die davon ausgeht, daß bestimmte Daten jeweils nur von einer Anwendung manipuliert werden.

c) Ausgestaltung der Architektur

Nach der Festlegung der Systemplattform und der Vorstellung der Austauschmechanismen wird nun der konkrete Aufbau des Systems dargestellt. Der folgende Abschnitt beschreibt die Rollen der einzelnen Komponenten, legt jedoch die sprachliche Ausgestaltung der Verständigung noch nicht fest.

aa) Selbständiges Programm mit DDE-Schnittstelle

Die Realisierung der konkreten Datenaustauschschnittstelle erfolgte anhand eines selbstablaufenden Programms mit einer DDE-Schnittstelle zu den teilnehmenden Anwendungen. Das Programm trägt den Arbeitstitel *Atlas*. Aufgrund einiger Vorarbeiten mit einem einfacheren aussagenlogisch ausgerichteten System erhielt es die Versionsnummer *2*. In den folgenden Ausführungen wird das Programm kurz als *Atlas* bezeichnet.

Das Programm kann vom Benutzer beim Hochfahren des Betriebssystems oder von jedem Teilnehmerprogramm bei Bedarf gestartet werden. Atlas erhält von den teilnehmenden Programmen via DDE alle juristisch relevanten Fakten übermittelt und liefert sie auf Anfrage an alle Teilnehmer weiter (zentralistischer asynchroner Datenfuß). Neben seiner Funktion als Ansprechpunkt für die Teilnehmerprogramme enthält es Optionen zur Datenpflege. Der Benutzer hat die Möglichkeit, sowohl die Datenbasis als auch die Fakten eines Falls sowie weitere interne Informationen direkt in der Anwendung zu betrachten und teilweise auch zu modifizieren.

bb) API zur Datenbankmaschine

Für die Zwischenspeicherung der Daten sorgt eine **Datenbankmaschine**. Sie kann je nach dem zu erwartenden Anfall von Daten und je nach verfügbarer Hard- und Software beliebig ausgetauscht werden. Während die Kommunikation von Atlas mit den teilnehmenden Anwendungen über das Windows-DDE-Protokoll abläuft, besitzt das Programm intern eine definierte Programmschnittstelle (API) zu der verwendeten Datenbank. Diese Schnittstelle beschreibt alle Funktionen, die eine DLL bereitstellen muß, um als Datenbankmaschine für Atlas zu fungieren. Eine solche DLL, die eine spezifisch für ein Programm zugeschnittene API zur Verfügung stellt, wird auch als Treiber bezeichnet.

So ist ein Treiber vorstellbar, der speziell für kleine Datenmengen alle Daten im Arbeitsspeicher des Computers behält. Ein solcher Treiber wäre besonders schnell im Datenzugriff. Es kann aber auch ein Treiber entwickelt werden, der die abstrakten Befehle von Atlas auf die Ebene der ODBC-Kommandos übersetzt. Die Daten

---

[304] So auch GERBLINGER, a.a.O.: *DDE ist sonach wichtige Basis für die Realisierung zukünftiger Anwendersoftware*.

können dann auf beliebigen von ODBC unterstützten SQL-Datenbanken abgelegt werden.

cc) API für Funktionserweiterungen

Das Serverprogramm Atlas besitzt lediglich einen rudimentären Funktionsumfang. Damit dieser Funktionsumfang flexibel erweitert werden kann, wird eine direkte Programmschnittstelle für **Erweiterungsmodule** angeboten.

Ein Beispiel für eine derartige Funktionserweiterung mag die Funktion *Alter*(*person*, *zeitpunkt*) sein, die das Alter einer Person zu einem bestimmten Zeitpunkt ermittelt. Da sich dieses Alter regelmäßig aus dem Geburtsdatum einer Person unzweifelhaft und eindeutig errechnen läßt, ist es nicht notwendig, jede dieser abgefragten Altersangaben in der Datenbank zu speichern. Vielmehr kann bei einer entsprechenden Anfrage eines Programms das Alter sofort aus dem Geburtsdatum errechnet werden.[305] Da sich das Alter zudem im regelmäßigen Fluß befindet, erscheint es hier auch umständlich, bei jeder Anfrage des Alters zu einem anderen Zeitpunkt eine separate Anwendung zur Altersberechnung zu starten und die Daten per DDE hin und her zu transferieren. Für derartig einfache und vor allem unstreitige Berechnungsverfahren ist das DDE-Protokoll zu träge und unbequem. Komplexe Berechnungsprogramme sind hier nicht notwendig. Deshalb können einfach gelagerte 1:1 Konvertierungen direkt in Atlas als Funktionserweiterung eingebunden werden und müssen nicht auf dem für komplexe Anwendungen üblichen Weg aufgerufen werden.

Weiterhin besitzt Atlas von sich aus keine Funktionalität, beim Fehlen von Fakten entsprechende Anwendungen zu starten, die diese Fakten eventuell ermitteln können. Diese Möglichkeit ist jedoch eine besonders interessante Möglichkeit der Integration des Arbeitsplatzes und wurde bereits mehrfach erwähnt. Auch für die Komplettierung des Funktionsumfangs in dieser Richtung dient die Schnittstelle. Sie erlaubt es, daß diese fehlende Funktionalität durch ein eigenes Modul angeboten wird. Im folgenden werden noch weitere Funktionen beschrieben, die Bereiche der Konzeption abdecken, de facto jedoch im Prototyp noch nicht realisiert sind. Auch diese Funktionen lassen sich regelmäßig ohne Eingriff in das eigentliche Programm über die Erweiterungsschnittstelle anbauen.

Der Vorteil dieses sehr modularen Aufbaus liegt in seiner hohen Anpassungsfähigkeit. So können Programme eigene, individuelle Methoden entwickeln, wie sie beim Fehlen von bestimmten Fakten gestartet und gesteuert werden. Durch modulare Erweiterbarkeit bleibt das System vom Konzept her zukunftssicher und kann weitgehend auf spätere Entwicklungen angepaßt werden.

[305] Vgl. S. 58.

dd) Abschließende Übersicht

Die folgende Graphik stellt die Systemarchitektur von Atlas im Überblick dar:

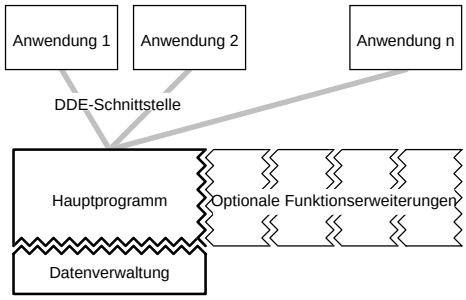


Abbildung 22: Schematische Darstellung der Atlas Architektur.

Die Graphik vernachlässigt dabei der Übersicht wegen, daß auch zwischen Funktionserweiterungen und Anwendungen Beziehungen unterschiedlichster Art bestehen können. So können die Funktionserweiterungen andere Anwendungen starten, sie können aber auch über DDE oder sogar über eine Programmierschnittstelle mit ihnen in Verbindung treten und sie fernsteuern. Weiterhin können Funktionserweiterungen mit spezifischen Programmen kürzere Wege des Datenaustauschs ermöglichen. Die Funktionserweiterung kann somit im äußersten Fall ein integrierter Bestandteil einer Anwendung sein.

Da der Übergang zwischen Funktionserweiterung und eigenständiger Anwendung somit fließend ist, sind die Schnittstellen, abgesehen von ihrer technischen Realisierung, sehr ähnlich konstruiert.

2. Datenmodell

Neben der Systemarchitektur stellt die Definition des internen Datenmodells eine wesentliche Systementscheidung dar. Sie hat zwar für die Anwendung des Programms keine unmittelbaren Auswirkungen, bestimmt aber mittelbar die Mächtigkeit des Programms und dessen Flexibilität im Hinblick auf zukünftige Erweiterungen. Nicht selten sind die einer Anwendung zugrundeliegenden Datenmodelle mächtiger als der Funktionsumfang der Schnittstellen.

Die konkrete Datenablage wird in der Atlas–Systemarchitektur von einem dynamisch angebundenen Datenverwaltungsmodul übernommen. Der folgende Abschnitt beschreibt ausschließlich das Datenmodell, wie es dem Hauptprogramm zugrunde liegt. Die Beschreibung dient dabei dem Verständnis der inneren Abläufe. Sie versetzt den Leser nicht in die Lage, Erweiterungen für den konkreten Prototypen zu schreiben. Hierzu dienen die mitgelieferten Programmierhilfen sowie die Beschreibung im Anhang. Als Beschreibungssprache in diesem wie in allen anderen Abschnitten soll Visual–BASIC dienen, wenn auch die konkrete Realisierung in Pascal erfolgte. Auf diese Weise kann sich der mit Programmiersprachen weniger vertraute Leser auf die Gewöhnung an eine Syntax beschränken.

a) Relationen

Atlas geht davon aus, daß Fakten als **Eigenschaften** von **Objekten** und deren **Beziehungen** untereinander abgelegt werden. Konkrete Beziehungen werden dabei aufgrund abstrakter **Relationen** beschrieben. Atlas speichert dieseRelationen in einer Datenbank, die als *globale Datenbasis* bezeichnet wird. Die Datenbasis kann vom Benutzer nicht ausgewählt werden. Es wird also von der Existenz genau einer Datenbasis auf einem Computer ausgegangen, die alle wesentlichen Beziehungen, von Personen unabhängig, von der Rechtsmaterie abbilden kann. Die Definition einer Relation in dieser Datenbasis enthält Angaben über den Bezeichner und den

Standardwertetyp. Sie ermöglicht den Zugriff auf die Parameterleiste und auf eine Liste von Zusatzmodulen, die unmittelbar an eine Relation gekoppelt werden können.

Die Kommunikation mit dem Datenverwaltungsmodul erfolgt über Funktionen zum Lesen, Verändern und Anfügen von Relationen. Zur Beschreibung einer Relation verwendet Atlas eine Struktur etwa folgender Art:

```
Type tRelation
  sBezeichner As String * 255 '                                    Name der Relation
  lFlags As Long '     Zusatzangaben
End Type
```

sBezeichner enthält den Bezeichner, also den Namen einer Relation. Der Bezeichner kann eine Zeichenkette von bis zu 255 Zeichen Länge sein. Eine Beschränkung der Zeichen ergibt sich aus der Speicherform nicht, wohl aber aus syntaktischen Bedürfnissen, die später dargestellt werden.

lFlags enthält eine Liste von Eigenschaften der konkreten Relation. Von *Flags* spricht man regelmäßig, wenn die einzelnen Bits einer Zahl definierten Eigenschaften zugewiesen werden. Eine Zahl vom Typ Long wird mit 32 Bits dargestellt, besitzt also in binärer Darstellung 32 Stellen und könnte so 32 verschiedene Eigenschaften unabhängig voneinander, d.h. mit 2^{32} Kombinationen darstellen. Auf die Aufgabe dieser Flag wird erst bei der Beschreibung der konkreten Syntax[306] näher eingegangen.

Die Parameter einer Relation werden definitionsgemäß durch reverse Relationen dargestellt[307]. Die Beschreibung der einzelnen Parameter kann also wiederum mit Variablen des Typs *tRelation* erfolgen. Der Zugriff auf die Parameter einer bestimmten Relation erfolgt über eigene Funktionen des Datenverwaltungsmoduls. Sie erlauben es, Parameter einer angegebenen Relation zu lesen, zu ändern, zu entfernen und hinzuzufügen. Die konkrete Verknüpfung der Parameter mit der Relation wird von dem Modul realisiert. Eine Funktion zum Auslesen eines Parameters kann etwa folgendermaßen lauten:

```
GibParameter (_
  RRelation As tRelation,_ '                    Angabe der Relation, deren Parameter gesucht wird
  RParameter As tRelation,_ ' „Puffer", in den die Daten des Parameters kopiert werden
  iIndex As integer)_ '  Angabe der Stelle des gesuchten Parameters
  As Integer    '  Rückgabewert hat Integer und meldet Erfolg der Operation
```

Diese Funktion *GibParameter* übergibt den durch *iIndex* bezeichneten Parameter aus der Parameterleiste der Relation *RRelation* an die Variable *RParameter*. Der Rückgabewert dieser Funktion meldet, ob die Funktion erfolgreich ausgeführt wurde oder ein Fehler aufgetreten ist. Die Funktion macht deutlich, daß, wie in der Spezifikation, davon ausgegangen wird, daß jeder Parameter selbst durch eine Relation repräsentiert ist. Entsprechend muß für die Beschreibung der Parameter lediglich auf diese Relation verwiesen werden. Hieraus ergibt sich ein Datenmodell, das im folgenden Abbild anhand eines Beispiels skizziert wird:

---

[306] S. S. 141 ff.

[307] Regel 2 s. S. 93.

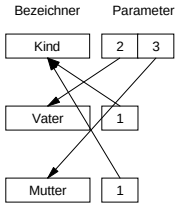


Abbildung 23: Schematische Darstellung des Datenmodells zur Beschreibung von Relationen in Atlas anhand der Relation $k = Kind(vater, mutter)$.
Auf die Darstellung der Flagliste wurde verzichtet.

Die technische Konkretisierung ist unabhängig vom abstrakten Modell. So ist beispielsweise eine relationale Verknüpfung zweier Datenbanktabellen denkbar. Während eine Tabelle die Informationen zu den einzelnen Relationen enthält, stellt eine zweite Tabelle die Beziehungen zwischen der Relation und ihren Parametern her. Da die Bezeichner der Relationen eindeutig sind, können sie auch als Schlüsselfeld dienen. Eine derartige Realisation kann etwa folgendermaßen aussehen:

Relationen

Bezeichner	Wertetyp
Kind	Bool
Vater	Bool
Mutter	Bool

Parameterliste

Relation	Parameter	Index
Kind	Vater	1
Kind	Mutter	2
Vater	Kind	1
Mutter	Kind	1

Abbildung 24: Vereinfachtes Schema einer relationalen Datenbank zur Abbildung des Atlas Datenmodells der globalen Datenbasis.

Neben der Parameterleiste besitzt jede Relation eine Liste von Erweiterungsmodulen. Diese Module können aufgerufen werden, falls ein Faktum nicht verfügbar ist, das über die Relation abgefragt wird. So kann beispielsweise für die Relation $y = Alter(x, t)$ ein Modul in diese Liste eingetragen werden, das das Alter berechnet oder etwa ein weiteres Programm zur Altersberechnung startet. Auch die Elemente dieser Liste werden über eine Funktion angesprochen. Die präzise Schnittstelle zu Erweiterungsmodulen wird später beschrieben[308].

b) Objekte

Auch Objekte werden von dem Datenverwaltungsmodul verwaltet. Im Unterschied zu Relationen beziehen sich Objekte auf den individuellen Fall. Sie sind Bestandteil des konkreten Sachverhalts. Atlas geht davon aus, daß Objekte für jeden Fall in einen separaten Datenpool, d.h. in der Regel in eine eigene Datei geschrieben werden. Zur Kommunikation mit dem Datenverwaltungsmodul existieren Funktionen zum Bilden neuer Objekte sowie zum lesenden Zugriff auf vorhandene Objekte. Für die Beschreibung von Objekten verwendet Atlas eine Datenstruktur etwa folgenden Typs:

```
Type tObjekt
  sBezeichner as String * 256
  lFlags as Long
End Type
```

`sBezeichner` enthält eine Zeichenkette zur Identifizierung des Objekts. Für ihn gelten dieselben Regeln wie für die Bezeichner von Relationen. `lFlags` dient der Aufnahme von Zusatzinformationen. So stellt das Flag

```
Const OBJ_VIRTUELL = &H00000001H
```

[308] S. S. 171.

fest, daß das Objekt implizit vom System instantiiert wurde und nicht explizit von einer teilnehmenden Anwendung. Die Variable bietet Raum für weitere 31 Flags oder insgesamt 2^{32} Kombinationen.

c) Fakten

Wie Objekte sind auch Fakten Elemente des konkreten Sachverhalts. Sie werden also zusammen mit den Werten an einer einheitlichen Stelle abgelegt. Als Fakten wurden sowohl Werte von Objekten als auch Beziehungen zwischen Objekten bezeichnet, jedenfalls dann, wenn über die Frage der Existenz einer Beziehung entschieden wird. Systematisch betrachtet handelt es sich in beiden Fällen um logische Aussagen. Im Fall einer Beziehung ist dies sofort einleuchtend. Behauptet jemand $M = Kind(V, M)$, so macht er damit eine Aussage, die einer logischen Bewertung zugänglich ist.

Mit Werten von Objekten verhält es sich jedoch nicht anders. Alle Werte, respektive Zahlen-, Datums- und logische Werte sowie Zeichenketten, sind Elemente entsprechender Elementenmengen. Sie wurden deshalb bereits bei der Definition des Begriffs Objekt[309] ebenfalls hierin eingeschlossen. Die Zuweisung eines *Wertes* zu einem *Objekt* entspricht de facto der Behauptung einer Beziehung. Wesentlicher Unterschied ist, daß die Relationen $y = Zahlwert(x)$, $y = Datumswert(x)$, $y = Existenzwert(x)$ oder $y = Zeichenkettenwert(x)$ vom System bereitgestellt werden und die Ausgabe in einer definierten Form erfolgt. Das System behandelt Beziehungen und Werte teilweise unterschiedlich und teilweise auch identisch.

Eine wesentliche Gemeinsamkeit von Beziehungen und Werten ergibt sich aus der Erkenntnis, daß es sich in beiden Fällen um Aussagen handelt. Sie unterliegen den bereits erwähnten faktischen Schwankungen[310]. Sie können nicht nur danach variieren, wer die Aussage macht, sondern auch danach, wann die Aussage gemacht wird. Weiterhin unterliegt der Inhalt einer Aussage zeitlichen Veränderungen. Somit gibt es für jede Aussage einen zeitlichen Gültigkeitsbereich. Deshalb wurde bereits in der Spezifikation gefordert, diesen elementaren Parametern dadurch Rechnung zu tragen, daß Informationen darüber mitgeführt werden, aus welcher Quelle eine Aussage stammt, zu welchem Zeitraum sie gemacht wurde, und für welchen Zeitraum sie gültig sein soll[311]. Zu diesem Zweck bedient sich Atlas einer Datenstruktur etwa der folgenden Art:

```
Type tInfoAussage '  Typdefinition für alle Formen von Aussagen
  lDatumEingabe As Long '                          Eingabedatum in serieller Form
  oDatumBeginn As tObjekt '                  Ereignis, mit dem die Gültigkeit der Aussage beginnt
  oDatumEnde As tObjekt '                     Ereignis, mit dem die Gültigkeit der Aussage endet
  oQuelle As tObjekt '  Quelle der Aussage
  sAnwendung As String'                 Name der Anwendung, die das Faktum übermittelt hat312
End Type
```

`lDatumEingabe` enthält eine numerische Angabe des Zeitpunktes, an dem die Aussage gemacht wurde. Der numerische Ausdruck läßt sich eineindeutig in eine für den Benutzer verständliche Zeichenkettenfolge umrechnen. Vorteil der numerischen Ablage von Daten ist die Tatsache, daß sich so klassische algebraische Operationen auch auf Daten anwenden lassen. Insbesondere reicht ein klassischer Größenver-

[309] S. S. 90.

[310] S. S. 97.

[311] Vgl. zusammenfassend s. S. 100.

[312] Im Prototyp nicht realisiert.

gleich aus, um zwischen früheren und späteren Datumsangaben unterscheiden zu können.

`oDatumBeginn` und `oDatumEnde` enthalten den zeitlichen Bereich, für den die Aussage gelten soll. Diese Informationen sind allerdings nicht als konkrete Datumsangaben, sondern als Referenzen auf Objekte realisiert. Dieser Ansatz geht davon aus, daß der Gültigkeitszeitraum definitionsgemäß ein regelmäßiger Bestandteil einer Relation ist. Die hierfür in einer Relation notwendigen Parameter *von* und *bis* werden aufgrund der tatsächlichen Unbeständigkeit aller Fakten bei jeder Aussage implizit vorausgesetzt. Die generelle Verfügbarkeit der Parameter durch Atlas führt lediglich zur Verkürzung einer ausdrücklichen Relation wie $Ehe(E, M, F, H, S)$ (mit $Eheschließung(H, E)$ und $Scheidung(S, E)$) um die letzten Parameter auf die Relation $Ehe(E, M, F)$. Entsprechend stellt Atlas diese impliziten Parameter auch in Form von Objekten zur Verfügung, wie sie bei expliziter Formulierung der Relation eingebracht würden. Praktisch bietet dieses Verfahren die Möglichkeit, zwischen den Ereignissen, die die Gültigkeit einer Aussage beschränken, und den sich daraus ergebenden Datumswerten zu unterscheiden. Dies ist insbesondere dann wichtig, wenn über unterschiedliche Punkte Streit besteht. So können *V* und dessen Tochter *T* darüber streiten, ob die Ehe mit der Ehefrau und Mutter *M* durch die Scheidung oder durch den späteren Tod der Mutter beendet wurde. Sie können aber auch über diesen Punkt einig sein und nur den Todeszeitpunkt anders determinieren. Im ersten Fall betrifft der Streit die den Gültigkeitszeitraum der Beziehung eingrenzenden Ereignisse, im zweiten Fall deren Datumswert.

Für die Angabe des Zeitpunktes der Aussage wurde auf eine Referenz zu einem Ereignis insbesondere deshalb verzichtet, weil die meist unstreitige aber überaus ereignisreiche Prozeßgeschichte nicht die materiellen Rechtsfragen überlagern sollte. Dennoch ist es denkbar, daß der Zeitpunkt einer Aussage auch im rechtlichen Sinne Einfluß auf den Ausgang eines Verfahrens hat. Dies geschieht etwa dann, wenn Tatsachen verspätet vorgebracht werden (§96 ZPO). In einem solchen Falle kann die Aussage selbst, mithin der Aussagezeitpunkt zum Verfahrensobjekt werden. Beide werden zum Gegenstand von juristischen Anwendungen und müssen wie jedes andere Objekt des Falls instantiiert werden.

Wie die Ereignisse des Gültigkeitszeitraums so ist `oQuelle` eine Referenz auf ein Objekt, nämlich den Urheber der repräsentierten Aussage. Dies mag verwundern, da die Quelle einer Aussage kaum eine andere Behandlung erwarten läßt, als der Zeitpunkt der Aussage. In beiden Fällen geht es primär um Fragen der Prozeßgeschichte. Die abweichende Realisierung erfolgt hier denn auch aus rein praktischen Erwägungen. Zum einen werden Aussagen regelmäßig von einer abgrenzbaren Zahl von Personen gemacht, die meist auch an dem behandelten Sachverhalt beteiligt sind. Zum anderen werden von diesen Personen in anderen Anwendungen oftmals Angaben, wie Name und Adresse benötigt. Auf diese Angaben kann durch den Verweis auf eine am Verfahren beteiligte Person leicht zugegriffen werden. Würde man hingegen nicht auf ein Objekt verweisen, ergebe sich das Problem, wie die Quelle statt dessen konkret zu beschreiben wäre.

d) Werte

Der Wert eines Objekts ist per Definition ein Faktum im Sinne dieser Arbeit. Die Zuordnung eines Wertes zu einem Objekt erfolgt mittels vom System vorgegebenen Relationen $y = Zahlwert(x)$, $y = Datumswert(x)$, $y = Existenzwert(x)$ oder $y = Zeichenkette(x)$. Die Zuordnung kann im konkreten Fall wahr oder falsch sein. Sie ist mithin eine Aussage im Sinne der Logik. Das System läßt beliebige solcher Wertzuweisungen zu. Es macht keinen Unterschied, ob die zugewiesenen Werte in Teilbereichen oder vollständig identisch sind. Mit jeder dieser Aussagen wird der

Zeitpunkt, die Quelle sowie ein Zeitbereich, in dem die Aussage überhaupt gültig sein soll, abgelegt.

Werte werden wie Objekte und Relationen über das angebundene Datenverwaltungsmodul verwaltet. Atlas geht davon aus, daß sie in einem fallbezogenen Datenpool, also beispielsweise einer fallbezogenen Datei abgelegt werden. Für den Zugang zu einem Wert geht das System davon aus, daß alle Werte eines Objekts als eine Liste verwaltet werden, die dem Objekt zugeordnet ist. Der Zugriff erfolgt unter Angabe des Objekts und einem Index des gesuchten Wertes in dessen Werteliste. Fehlt eine Indexangabe wird der zuletzt eingegebene Wert, d.h. der Wert mit dem höchsten Index, verwendet. Das Zentralprogramm greift auf die Daten über Funktionen zum Anlegen neuer und Abrufen alter Werte zu. Zum Austausch der Informationen über eine Aussage bezüglich eines Wertes wird eine Datenstruktur etwa folgender Art eingesetzt:

```
Type tWert '          Typdefinition für Werte von Objekten
  InfoAussage As tInfoAussage '                    Angaben über Geltungszeitraum, Quelle etc.
  iTyp As Integer '   Typ des Wertes (Zahl, Bool, Datum, Zeichenkette)
  vWert As Variant '  Angabe des Wertes in der typgerechten Form
End Type
```

InfoAussage enthält die Angaben über den Zeitpunkt der Aussage, die Quelle sowie den Gültigkeitsbereich[313]. *iTyp* gibt den Wertetyp an. Der Parameter kann einen der folgenden Werte annehmen:

```
' Konstanten zur Definition des Standard-Wertetyps einer Relation
Const TTY_TEXT = 1 ' Zeichenkette (String)
Const TTY_REAL = 3 ' Fließkommazahl (Double)
Const TTY_DATE = 4 ' Datumswert (Long)
Const TTY_XBOOL= 5 '                                  logischer Wert (Integer)
```

vWert enthält den eigentlichen Wert. Der Typ *Variant* kann unterschiedliche Wertetypen repräsentieren.

- Textwerte können beliebige Zeichenketten von bis zu 255 Zeichen des Systemzeichensatzes (ANSI) sein.
- Realzahlen werden intern durch 64 Bit dargestellt und können Fließkommawerte im Bereich von 1.79769313486232E308 bis -4.94065645841247E-324 im negativen und von 4.94065645841247E-324 bis 1.79769313486232E308 im positiven Bereich sein.
- Datumswerte werden als Ganzzahlen im Bereich von 0 bis 2^{32} (4.294.967.296) abgelegt. Die Ablage erlaubt theoretisch eine Speicherung jeder Minute innerhalb von ca. 8000 Jahren.
- Wahrheitswerte werden durch Ganzzahlen dargestellt. Folgende Werte sind zulässig:

```
' Deklaration der möglichen Werte der dreiwertigen Logik
Const XBO_WAHR = 1 '                                  wahr
Const XBO_FALSCH = 2 '                                falsch
Const XBO_UNKLAR = 3 '                                (bewußt) unklar
```

e) Beziehungen

Eine Beziehung ist eine Verbindung eines oder mehrerer Objekte auf der Grundlage einer Relation. Der Zugriff von Atlas auf eine konkrete Beziehung erfolgt über die entsprechende Relation. Wie bei Objekten geht Atlas auch bei Beziehungen davon aus, daß diese vom Datenverwaltungsmodul in eigenständigen fallbezogenen Datenpools abgelegt werden. Dieser Datenpool enthält für jede Relation eine Liste von

[313] Vgl. S. 97 ff.

konkreten Beziehungen, die auf der Relation beruhen. Wird der Beziehung eine Information darüber beigefügt, ob sie tatsächlich besteht, nicht besteht oder ihr Bestehen unklar ist, so kann der Ausdruck als Ganzes einer logischen Bewertung unterzogen werden. Es handelt sich dann um eine Aussage.

Atlas kann beliebige Beziehungen und beliebige Existenzwerte von Beziehungen verwalten. Das Programm kommuniziert mit dem Datenverwaltungsmodul über Funktionen, die die Anlage neuer Beziehungen und das Lesen alter Beziehungen ermöglicht. Zur Beschreibung einer Beziehung wird dabei eine Datenstruktur eingesetzt, die nachfolgend vereinfacht dargestellt wird:

```
' Diese Struktur dient der Beschreibung konkreter Beziehungen
Type tBeziehung '    Typ zur Beschreibung einer Beziehung
  InfoAussage As tInfoAussage '                    Angaben über Geltungszeitraum, Quelle etc.
  iExist As Integer '   Angabe über die tatsächliche Existenz der Beziehung
  Objekt(64) As tObjekt '                  Angabe von bis zu 64 an der Beziehung beteiligten Objekten
End Type
```

`InfoAussage` enthält die Angaben über den Zeitpunkt der Aussage, die Quelle sowie den Gültigkeitsbereich[314]. `iExist` nimmt einen der folgenden Wahrheitswerte an:

```
' Deklaration der möglichen Werte der dreiwertigen Logik
Const XBO_DUMMY = 0 '                                  Wert ist nicht gesetzt
Const XBO_TRUE = 1 '                                   wahr
Const XBO_FALSE = 2 '                                  falsch
Const XBO_UNCERTAIN = 3 '                              (bewußt) unklar
```

Damit wird angegeben, ob die Beziehung existiert, nicht existiert oder ob ihre Existenz unklar ist. Weiterhin kann die Aussage gänzlich offen gelassen werden. Eine Beschreibung einer Beziehung, bei der `iExist` nicht den *Nichtwert* `XBO_DUMMY` hat, kann als Aussage angesehen werden. Atlas geht also davon aus, daß für jede Beziehung die Informationen zu der konkreteren Aussage mitgeliefert werden. Dabei ergibt sich dann eine Redundanz, wenn etwa für die Beziehung $E = Ehe(M, F)$ beide Aussagen, sie existiert und sie existiert nicht, aus unterschiedlichen Quellen übermittelt werden. Hier bleibt es dem Datenverwaltungsmodul überlassen, durch eine Trennung von Beziehung und Existenzwert für eine redundanzfreie Datenhaltung zu sorgen.

Die Liste `Objekt` kann bis zu 64 Objekte aufnehmen. Sie bilden die Parameter der konkreten Beziehung. `Objekt(0)` repräsentiert den Hauptparameter und muß immer belegt sein. Die Anzahl der weiteren Objekte ergibt sich aus der Anzahl der Parameter der zugrundeliegenden Relation.

[314] Vgl. S. 97 ff.

f) Abschließender Überblick

Abbildung 25 soll noch einmal einen Überblick über die im Datenmodell von Atlas vorgesehenen Verknüpfungen verschaffen.

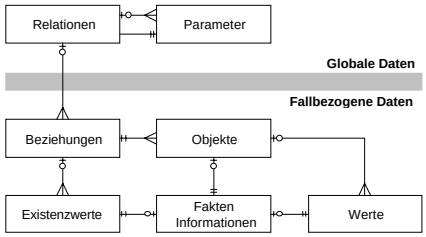


Abbildung 25: Schematisiertes Datenmodell, wie es dem funktionellen Zugang von Atlas auf das Datenverwaltungsmodul zugrundeliegt.

Das Wechselspiel zwischen Relationen und Parametern ergibt sich daraus, daß Parameter immer durch einen Verweis auf Relationen definiert werden. Das Wechselspiel von (Existenz-)Werten und Objekten ergibt sich aus den Informationen über Geltungszeitraum und Informationsquelle einer Aussage. Sie werden als Verweise auf Objekte realisiert.

3. Datenpflege

Der vorangegangene Abschnitt hat sich mit dem Datenmodell beschäftigt, wie es Atlas zugrundeliegt. Es ist für die Definition der abstrakten Zufgriffsmechanismen auf das Datenverwaltungsmodul sowie für die hierfür verwendeten Datenstrukturen notwendig. Der nun folgende Abschnitt beschäftigt sich mit der Frage, wie, von wem und aus welchem Anlaß Daten verändert werden. Dabei ist zu differenzieren zwischen Zugriffen auf die global definierten Relationen und den Zugriffen auf die Fakten eines einzelnen Falls.

a) Globale Daten

Ein wesentliches Problem der praktischen Realisierung ergibt sich aus der Frage, wie universelle Relationen vom System am Arbeitsplatz des Endanwenders verwaltet werden. Selbstverständlich ist, daß sie nicht fest implementierter Bestandteil des Programms Atlas sind, sondern an jedem Arbeitsplatz frei erweitert werden können. Andernfalls läge es in der Verantwortung des Systementwicklers, alle rechtlich denkbaren Relationen zu sammeln und zu implementieren. Insbesondere müßte er immer sofort auf Rechtsänderungen und die möglichen Konsequenzen eingehen. Alle Anwendungen wären somit auf die Kompetenz, Qualität und Geschwindigkeit des für die Pflege des Computerprogramms zuständigen Organs angewiesen. Dies ist nicht der Sinn des angestrebten Systems. Vielmehr ist der Autor einer juristischen Endanwendung weitaus tiefer mit der juristischen Materie befaßt als der Entwickler einer technischen Schnittstelle. Somit muß es auch dem Anwendungsentwickler obliegen, die benötigten Relationen zu beschreiben und in das System zu implementieren. Das System selbst wird sozusagen zum Zeitpunkt der Veröffentlichung leer, d.h. ohne jegliche Relationen ausgeliefert. So ist selbst die Relation $y = Kind(vater, mutter)$ nicht von Anfang an Teil des Systems. Fraglich ist nun, wie diese Funktionalität in das System implementiert wird. Hier ergeben sich zwei Möglichkeiten: eine **implizite** oder eine **explizite** Implementierung.

Eine implizite Implementierung richtet eine Relation im System quasi durch ihre Benutzung ein. Eine Anfrage $y = Name(Kind(Peter, Tina))$ installiert im System somit die Relation $y = Kind(v, m)$ sowie $y = Name(x)$. Syntaktische Voraussetzung hierzu ist, daß das System Relationsbezeichner (*Kind* und *Name*) von Objektbezeichnern (*Peter* und *Tina*) unterscheiden kann. Praktisch weiterhin notwendig für

die Anlage einer Relation ist, daß diese nicht schon im System existiert. Hierbei ergibt sich dann ein besonderes Problem, wenn die im System bestehende Beschreibung von der aktuellen Verwendung abweicht. Dies kann etwa dann geschehen, wenn die Anzahl oder gar Reihenfolge der Parameter, die unterschiedliche Anwendungen annehmen, voneinander abweichen. Die vom Benutzer verwendete Anwendung oder Atlas müssen dann während des Ablaufs eine Fehlermeldung ausgeben, die den Benutzer an der Ablage seiner Fakten hindert. Im Fall einer Abweichung der Parameterfolge wird die Erkennung des Fehlers kaum möglich sein.

Eine explizite Installation der Relationen geschieht typischerweise im Setup eines Anwendungsprogramms oder bei dessen erstem Aufruf. Im Regelfall kennt jedes Anwendungsprogramm alle von ihm benötigten bzw. bedienten Relationen. Es teilt diese innerhalb der Programminstallation dem System mit. Eventuelle Kollisionen mit bereits definierten Beschreibungen können zu diesem Zeitpunkt behoben werden. Insbesondere ermöglicht dieses Verfahren auch anderen Anwendungen den sofortigen Zugriff auf die neuen Relationen. Dies ist speziell für solche Anwendungen von Interesse, die einen Zugriff auf alle im System vorhandenen Relationen ausnutzen, ungeachtet woher diese stammen. Dies sind z.B. Programme, die den Sachverhalt visualisieren und dabei auf alle Kriterien zugreifen sollen, die am Arbeitsplatz des Anwenders von Interesse sein können. Ein solches Programm wäre etwa die später beschriebene elektronische Fallskizze[315].

Atlas erlaubt aufgrund der Unsicherheiten, die zum Zeitpunkt der Anwendung noch auftreten können, keine implizite Installation von Relationen, sondern erwartet eine explizite Installation. Die so installierten Relationen bleiben dem System solange erhalten, bis sie explizit geändert oder gelöscht werden. Hierzu dient die Möglichkeit des Benutzers, direkt im Dialog mit Atlas neue Relationen zu definieren oder vorhandene zu modifizieren. Von ihr sollte allerdings nur in äußersten Ausnahmesituationen vom Endbenutzer Gebrauch gemacht werden.

b) Objekte und Werte

Bei der Verwaltung von Objekten und deren Werten liegt die Sache anders als bei der Verwaltung der globalen Datenbasis. Objekte und Werte sind fallbezogen. Sie stützen sich nicht auf einen vorgegebenen Normenapparat, quasi ein Wirklichkeitsmodell. Sie füllen diesen Apparat aus. Die denkbaren Gestaltungsmöglichkeiten sind nahezu unendlich.

Objekte werden deshalb selbstverständlich während des Programmablaufes generiert. Man spricht hier auch von **Instantiierung**. Gleichwohl ist in diesem Fall eine explizite Form der Anlage und eine implizite Form möglich und auch gestattet. Eine explizite Instantiierung liegt vor, wenn der Wert eines bis dahin nicht benannten Objekts wie etwa die Existenz oder die Nichtexistenz oder auch deren Zweifelhaftigkeit dem System explizit mitgeteilt wird. Dies erfolgt etwa mit der Übermittlung folgender Beziehung $Existiert(Peter) = wahr$. *Existiert* ist eine systeminterne Funktion, die auf den Existenzwert des Objekts zugreift. Hier wird das Objekt *Peter* eingerichtet. Dabei wird ihm der Existenzwert *wahr* zugewiesen.

Implizit erfolgt eine Instantiierung beispielsweise mit der Übermittlung der Beziehung $Geburtstag(Kind(Peter, Tina)) = 10.10.63$. Zunächst einmal werden hier die Objekte *Peter* und *Tina* instantiiert, falls diese Objekte dem System noch nicht bekannt sind. Ihnen wird jedoch kein Wert zugewiesen. Weiterhin geht die Beziehung von der Existenz eines Objekts aus, das zu *Peter* und *Tina* in der Beziehung *Kind*

[315] S. S. 190.

steht. Falls ein solches Objekt noch nicht existiert, muß es das System anlegen. Da das Objekt keinen Bezeichner hat wie *Peter* oder *Tina*, muß das System einen geeigneten Bezeichner bilden. Es verwendet hierzu den Bezeichner der Beziehung sowie der beiden bezogenen Objekte und generiert hieraus den Bezeichner *Kind Peter-Tina*. Derartige Bezeichner werden programmintern als *virtuell* bezeichnet.

Der Ausdruck $Geburtstag(x) = y$ stellt laut Konzeption eine verkürzte Form des Ausdrucks $Datum(Geburtstag(x)) = y$ dar[316]. Dieser Ausdruck geht von der Existenz eines weiteren Objekts aus, nämlich des Geburtstages. Atlas instantiiert auch dieses Objekt mit dem Bezeichner *Geburtstag Kind Peter-Tina*. Es weist ihm den Datumswert *10.10.1963* zu. Atlas geht davon aus, daß die konkreten Objekte und Werte an individuellen Speicherplätzen abgelegt werden. Diese Daten werden als *fallbezogene Daten* bezeichnet. Eine Quersuche über die Daten mehrerer Fälle hinweg ist nicht vorgesehen. Das Datenmodell erlaubt allerdings prinzipiell auch die Ablage aller Fakten in einer Datenbank. Die Verwaltung der Daten liegt bei dem konkreten Datenverwaltungsmodul. Dieses hat unter Umgehung des Konzeptes von Einzeldateien auch die Möglichkeit, alle Fakten in einer Datenbasis abzulegen und entsprechend durchsuchbar zu machen.

c) Exkurs: Definition von Standards

Die Verlagerung der Angabe der Relationen auf den Zeitpunkt des Setups ermöglicht eine frühzeitige Erkennung von Fehlern, die durch unterschiedliche Definition einer Relation auftreten können. Das eigentliche Problem, derartige Definitionsunterschiede im Vorfeld zu vermeiden, ist damit jedoch nicht behoben. Ihm widmet sich der folgende Exkurs.

aa) Problemstellung

Die beschriebene Konzeption geht davon aus, daß jede Anwendung, die sich der Dienste von Atlas annehmen möchte, in ihrem Installationsprogramm die von ihm benötigten oder bedienten Relationen selbst einrichtet. Die Beziehungen sind ab diesem Zeitpunkt Knotenpunkte des hier entwickelten Modells zum Austausch von Fakten.

Zwei Anwendungen können nur dann Fakten austauschen, wenn sie dieselben Relationen einsetzen. Diese Definition derselben Relation beinhaltet aus formaler Sicht, daß derselbe Bezeichner und dieselben Parameter (in Anzahl und Reihenfolge) definiert sein müssen. Inhaltlich müssen mit den formal identischen Definitionen auch dieselben Tatbestände gemeint sein. Eine Beziehung $y = Gehalt(x)$ ist nur dann für einen Austausch der benötigten Fakten geeignet, wenn alle am Austauschprozeß beteiligten Anwendungen hierunter dasselbe Faktum verstehen. Es muß also Einigkeit herrschen, ob Brutto- oder Nettogehalt gemeint ist. Wenn das Nettogehalt beschrieben werden soll, dann muß Klarheit herrschen, welche Abzüge berücksichtigt wurden und welche nicht.

bb) Begriffsdefinition

Die Schwierigkeiten bei der Bildung von Rechtsbegriffen wurden bereits dargelegt[317]. Gleichwohl muß für einen widerspruchsfreien Austausch eines Faktums mittels einer Relation deren inhaltliche Bedeutung, mithin der zugrundeliegende

[316] Vgl. S. 97.

[317] S.o. S. 30.

Rechtsbegriff festgelegt werden[318]. Für den hier verfolgten Zweck würde es allerdings ausreichen, die Identität mehrerer Begriffe für unterschiedliche Vorgänge festzustellen. Auf den eigentlichen Inhalt muß dabei nicht zwingend eingegangen werden. Das bereits bemühte Beispiel des Begriffs *Sache* zeigt jedoch, daß eine einheitliche Wortwahl innerhalb des Normtextes noch kein ausreichender Anhaltspunkt für eine begriffliche Identität darstellt.

Die Problematik kann jedoch oftmals auf pragmatischem Wege entschärft werden. Soll beispielsweise das Ergebnis einer Berechnung oder Subsumtion in einen Schriftsatz übernommen werden, reduziert sich das Problem der Begriffsbildung auf eine Definition eines eindeutigen Bezeichners. Hier ist nämlich die begriffliche Identität aufgrund des angestrebten Ziels unproblematisch. Das Problem stellt sich also nur dann, wenn die inhaltliche Identität zweier Relationen sich nicht ohnehin aus einer definitorischen Abhängigkeit ergibt. Das ist einerseits dann der Fall, wenn ein reales Sachverhaltsmerkmal mit einem Merkmal eines Vorgangs zur Deckung gebracht werden soll oder andererseits wenn Fakten zwischen zwei Regeln ausgetauscht werden sollen, bei denen die begriffliche Identität nicht zwingend ist. Hier zeigt ein Rückgriff auf die Ansätze der Methodenlehre, daß die eigentlichen Probleme meist nur in Grenzfällen auftauchen[319]. Selbst wenn also für diese Grenzfälle die Bedeutung eines Begriffs fraglich ist, mithin die Identität zweier Begriffe nicht gesichert ist, so berührt das nicht die Vielzahl der praktischen Anwendungsfälle.

Daraus ergibt sich die Empfehlung eines pragmatischen Vorgehens. Ist die Bedeutung eines Begriffs in den Randbereichen nicht sicher, so kann für den gesicherten Kernbereich ein eigener Begriff gebildet werden, der jedenfalls den weiteren Begriff impliziert. Ist bei zwei unterschiedlichen Begriffen ein identischer Kernbereich unproblematisch, so kann ein eng zu interpretierender Begriff gebildet werden, der die Schnittmenge der Begriffe abdeckt. Für das Beispiel der unterschiedlichen Sachbegriffe würde das bedeuten, daß ein Begriff *Sache_generell* zu bilden ist, der beide Sachbegriffe impliziert:

Sache_generell ⟶ Sache_Strafrecht
Sache_generell ⟶ Sache_Zivilrecht

Ein Gegenstand wie ein Auto kann problemlos dem engeren Begriff zugeordnet werden. Das Faktum, daß eine Sache vorliegt, steht sodann Vorgängen aus dem Familien- und aus dem Strafrecht zur Verfügung. Erwartet ein Vorgang beispielsweise das Faktum *Sache_Strafrecht*, so wird er bei einer Anfrage sowohl die allgemeine aber engere Beschreibung mittels des Begriffs *Sache_generell* als auch die weitergefaßte aber speziellere Beschreibung verwenden. Bei der Definition von Relationen sollte diese Mehrstufigkeit optimal genutzt werden. Das entsprechende Wissen um die Hierarchie der Begriffe muß jedenfalls in die Vorgänge implementiert werden, die mit unklaren Begriffen zu tun haben. So erhöht sich auf pragmatische Weise die Kommunikationsfähigkeit der Vorgänge zumindest für eine Vielzahl von Standardfällen.

cc) Wahl eindeutiger Bezeichner

Sind die Begriffe ausreichend definiert, so ist weitere zwingende Voraussetzung für einen Datenaustausch, daß die Bezeichner vereinheitlicht werden. Das oben genannte Beispiel *Gehalt* genügt dieser Voraussetzung nicht. Der Bezeichner selbst

---

[318] Über ein widerspruchsfreies System von Begriffen *Larenz*, S. 441 ff., der dieses Ideal für unerreichbar hält (S. 453).

[319] S.o. FN. 98.

müßte über das Präfix *Brutto* oder *Netto* bereits für Klarheit sorgen. Vernünftigerweise ist aus dem Namen des Bezeichners der juristische Anknüpfungspunkt erkennbar. So zeigt der Bezeichner *Monatseinkommen DüDoTab*, daß er das monatliche Einkommen beschreibt, das die Grundlage für die Berechnung nach der Düsseldorfer Tabelle ist. Dabei ist es gleichgültig, welche potentiellen Streitigkeiten über die Berechnung dieses Wertes herrschen. Diese Streitigkeiten sind Bestandteil der Konkretisierung des Wertes, sie liegen also vor der Zuweisung des Wertes zu dem entsprechenden Objekt. Die Relation bezeichnet einen eindeutigen Wert, der sich möglicherweise auf alternativen Wegen ermitteln läßt. Ist dies der Fall, so ist es denkbar, daß weitere Relationen implementiert werden, die die Ergebnisse der beiden Rechnungen repräsentieren. Dann bleibt es entweder dem Benutzer oder aber einer dritten Kollisionsregel überlassen, einen der Ausgabewerte der Berechnungen auch dem Eingangswert der Düsseldorfer Tabelle zuzuweisen. Der letztlich gewählte Wert wäre dann als eine Art Dublette im System vorhanden.

dd) Einrichtung einer Normungs-AG

Praktisch ergeben sich bei der Normung von Datenaustauschvorgängen immer zwei Wege. Einerseits kann ein marktbeherrschender Anbieter de facto Standards schaffen und somit andere zwingen, sich an derartige Standards anzuhängen. Andererseits kann eine unabhängige Arbeitsgemeinschaft derartige Standards setzen. Sie besteht aus Mitarbeitern mehrerer Anbieter und oft aus Teilnehmern der betroffenen Wissenschaftszweige.

Die erste Methode ist nicht von vornherein zu verwerfen. Oft sind derartige Standards sehr praxistauglich. Insbesondere unterliegen sie nicht den sehr trägen Verfahrenszwängen eines Normenausschusses. Nicht selten werden derartige Standards auch von offiziellen Normen übernommen und verfeinert. Dennoch scheint das Verfahren methodisch bedenklich, da nicht ausreichend vielfältiges Wissen in die Erstellung derartig zentraler Standards eingebracht wird. Zudem ist ein entsprechend marktdominierender Anbieter für juristische Anwendungen derzeit am deutschen Markt auch nicht erkennbar[320].

Demzufolge ist die Bildung einer Arbeitsgemeinschaft erstrebenswert, die sich folgende Ziele setzen sollte:

1. Aufstellung von konkreten Regeln zur Bildung von Relationen, soweit diese in der vorliegenden Arbeit nicht ausreichend konkretisiert sind.
2. Entwicklung eines Zulassungs- und Veröffentlichungsverfahrens für neu gebildete Relationen. Jede von einem Anbieter benötigte bzw. bediente Konstante sollte dieses Verfahren durchlaufen. Dabei muß der Anbieter das Faktum, welches mit der Beziehung charakterisiert werden soll, in einer definierten Weise determinieren.
3. Durchführung des festgelegten Verfahrens. Dabei muß insbesondere abgeprüft werden, ob ein Bezeichner bereits mit anderen Inhalten oder anderen Parametern im System vorhanden ist. Weiterhin muß untersucht werden, ob dasselbe Faktum bereits mit einem anderen Bezeichner beschrieben wird.

Die Durchsetzungsmacht einer solchen Normungsstelle könnte durch eine bedingte Lizensierung der notwendigen Software oder durch Lizensierung eines Zertifikati-

---

[320] Nach *Schleicher*, jur-pc 91, S. 1216 f. fallen auch Programme in die Domäne juristischer Verlage. Sind gerade in diesem Bereich Anbieter unterschiedlichster Herkunft (allgemeine und juristisch spezialisierte Softwarehäuser, Verlage mit juristischem aber auch mit EDV–Schwerpunkt) erkennbar.

onszeichens untermauert werden. Eine derartige Normungsstelle kann für private Anwender bei dem entsprechenden Berufsverband (etwa der Anwaltskammer) angesiedelt werden. Für öffentliche Anbieter wäre eine Normierung durch das BMJ oder eine andere öffentliche Stelle denkbar. Weiterhin sind die Ausarbeitung von Richtlinien und Unterbreitung von Standardisierungsvorschlägen durch eine Institution wie den EDV-Gerichtstag e.V. denkbar.

Die Normung der dem System zugrundeliegenden Relationen ist ein wesentlicher Faktor für den sinnvollen Einsatz des Systems im Alltag. Sie kann durch Richtlinien an die Begriffsbildung vereinfacht werden, sollte jedoch durch eine Zertifizierung einer unabhängigen Stelle durchgesetzt werden.

D. Schnittstellen von Atlas

Im vorangegangenen Abschnitt wurden vor allem Systeminterna beschrieben. In diesem Kapitel werden die einzelnen Varianten beschrieben, wie Atlas mit der Außenwelt in Kontakt tritt. Diese Einstiegspunkte, mit denen der Nutzer oder andere Programme eine Anwendung beeinflussen können, werden als Schnittstellen der Anwendung bezeichnet.

Die folgende Programmbeschreibung geht wie auch die vorangegangene Systembeschreibung von einem fiktiven Programm Atlas aus. Das beigefügte Veranschaulichungsprogramm weicht in Details der Bedienung und der Terminologie von dem hypothetischen Programm ab. Im Anhang findet sich eine kurze Bedienungsanleitung des ablauffähigen Begleitprogramms.

1. Benutzerschnittstelle

Bereits in der Beschreibung der Systemarchitektur wurde gesagt, daß Atlas als eine selbstablaufende Applikation konzipiert ist. Eine solche Applikation besitzt in der Regel auch eine Benutzerschnittstelle. In diesem Fall hat die Benutzerschnittstelle nicht die Bedeutung etwa eines Textverarbeitungsprogramms oder einer Tabellenkalkulation.

Die Benutzerschnittstelle des Programms dient zur Pflege der globalen Datenbasis sowie zur Beobachtung des Zustandes des Systems. Sie stellt lediglich eine von vier Schnittstellen von Atlas dar. Weitere Schnittstellen dienen der eigentlichen internen Kommunikation (DDE), der Datenbankanbindung und der Funktionserweiterung. Da Atlas über letztere Schnittstelle von anderen Programmen aus vollends ferngesteuert werden kann, dient die Benutzerschnittstelle letztlich nur der Veranschaulichung der Arbeitsweise des Programms. Für den praktischen Einsatz wäre eine Benutzerschnittstelle gänzlich verzichtbar. In modernen Betriebssystemen wie Windows wird ein solches Programm zur Laufzeit mit seinem Programmsymbol dargestellt.

Das folgende Abbild skizziert das Anwendungsfenster:

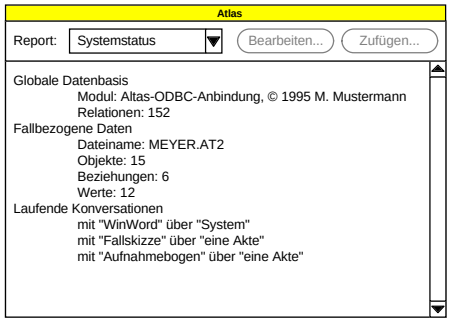


Abbildung 26: Darstellung des Hauptanwendungsfensters einer fiktiven Benutzerschnittstelle von Atlas.

Das Programm stellt in seinem Hauptfenster mehrere Sichten auf die bereitgestellten Informationen zur Verfügung. Eine solche Sicht wird zukünftig als *Report* bezeichnet. In der aufklappbaren Liste *Report* kann ausgewählt werden, welche Sicht angezeigt werden soll. Im Abbild wurde z.B. der Report *Systemstatus* ausgewählt. Die Darstellung der Informationen erfolgt in dem unter der Liste liegenden Bereich. Sie ist in vielen Fällen hierarchisch gegliedert. Die Gliederungsebenen werden durch entsprechend starke Einzüge dargestellt.

Der Report *Beziehungen* ermöglicht neben einer Anzeige auch die Veränderung bestehender und Eingabe neuer Relationen. Die Schalter *Bearbeiten...* und *Zufügen...* sind in der Ansicht dieses Reports zugänglich. In allen übrigen Reports sind die Schalter grau respektive nicht bedienbar.

a) Systemstatus

Der Report *Systemstatus* ist im vorangegangenen Abbild dargestellt. Er gibt einen kurzen Überblick über die Größe der globalen und der fallbezogenen Datenbasis.

Unter dem Stichwort *Modul* wird eine Information angezeigt, die vom jeweils verwendeten Datenverwaltungsmodul abgefragt wird. Mit *Relationen* wird die Anzahl der in der globalen Datenbasis eingetragenen Relationen angegeben. *Dateiname* gibt den Namen der Datei für die fallbezogenen Daten wieder. *Objekte* zeigt die Anzahl der Objekte an, die im aktuellen Fall bereits instantiiert wurden.

Der Punkt *Werte* zeigt die Anzahl aller an das Programm übermittelten Werte an. Atlas legt jeden übermittelten Wert erneut ab, selbst wenn er bereits vorhanden ist. Entsprechendes gilt für den Punkt *Beziehungen*, der alle an Atlas übermittelten Beziehungen anzeigt.

Die Liste *Laufende Konversationen* zeigt an, welche anderen Anwendungen auf Atlas aktuell zugreifen. Dabei wird der Anwendungsname sowie das Konversationsthema ausgegeben.

b) Relationen

Der Report *Relationen* erlaubt auch die Pflege der globalen Datenbasis. Er wird deshalb ausführlicher behandelt.

aa) Darstellung

Der Report *Relationen* gibt den Inhalt der globalen Datenbasis wieder. Abbildung 27 zeigt einen Ausschnitt aus einer fiktiven Datenbasis. In der ersten Gliederungsebene werden die Bezeichner der Relationen angezeigt. In der zweiten Ebene werden die zugehörigen Parameter aufgelistet.

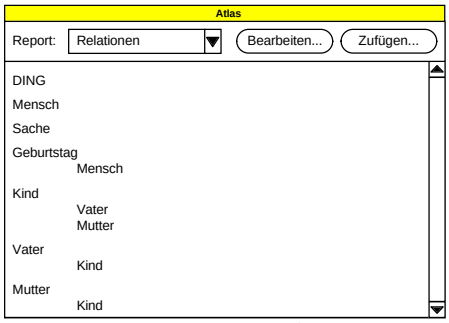


Abbildung 27: Ausschnitt aus einer fiktiven Datenbasis.

Es mag zunächst verwirren, daß die Relation *Mensch* keinerlei Parameter besitzt. Insbesondere fehlt also ein Parameter, der das Objekt repräsentiert, welches letztlich Mensch ist. Ebenso fehlt in der Beziehung *Kind* ein Parameter, der das Kind selbst

repräsentiert. Man mag geneigt sein, anzunehmen, daß die Relation *Kind* lediglich eine Beziehung zwischen *Vater* und *Mutter* herstellt.

Diese letztlich nur verkürzte Notation ergibt sich aus der vormals aufgestellten Prämisse, daß jede Beziehung auch funktional darstellbar sein muß[321]. Eine Beziehung muß also kein, ein oder mehrere Objekte auf ein weiteres Objekt abbilden. So bildet die Beziehung *Kind* die Objekte *Vater* und *Mutter* eben auf ein Kind ab. Die Relation repräsentiert aus dieser Sicht gleichzeitig das vermeintlich fehlende Objekt. Eigenschaften wie *Mensch*, *Sache* oder etwa die inhaltslose Eigenschaft *DING* werden danach ohne Parameter dargestellt.

bb) Bearbeiten und Erweitern der globalen Datenbasis.

Das Programm erlaubt bei dieser Reportfunktion neben der einfachen Anzeige auch die Bearbeitung und Erweiterung der globalen Datenbasis. Für die Bearbeitung einer bestehenden Beziehungsvariablen ist eine solche in der Liste zu markieren und der Schalter *Bearbeiten...* zu betätigen. Will man eine neue Relation zur Datenbasis hinzufügen, so ist der Schalter *Zufügen...* zu betätigen. In beiden Fällen wird ein gebundenes[322] Fenster geöffnet, welches das folgende Abbild skizziert.

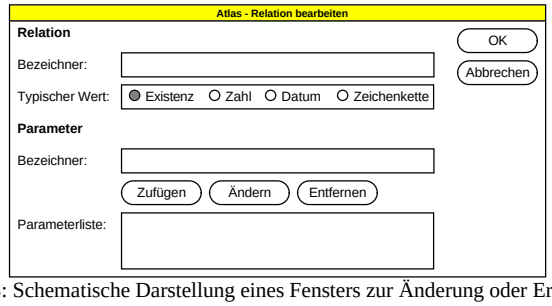


Abbildung 28: Schematische Darstellung eines Fensters zur Änderung oder Erweiterung der globalen Datenbasis.

Die Box *Bezeichner* in der Sektion *Relation* enthält den Bezeichner der zu ändernden oder neu einzufügenden Relation. Der Bezeichner ist die Zeichenkette, mit der die Relation identifiziert wird.

Die Box *Typischer Wert* erlaubt die Auswahl des Wertetyps, der standardmäßig bei Anfragen unter Verwendung des Bezeichners zurückgegeben werden soll. Für die Relation *Name* wäre dies etwa *Zeichenkette*, für *Preis* wäre dies *Zahl* und für *Geburtstag* *Datum*. In den meisten Fällen wird hier der Standardtyp *Existenz*, der einen Wahrheitswert übermittelt, anzugeben sein.

In der Sektion *Parameter* kann über die Schalter *Zufügen*, *Ändern* und *Entfernen* die Parameterleiste modifiziert werden. Die Box *Bezeichner* dieses Bereichs enthält den Bezeichner des gerade bearbeiteten Parameters.

Parameter müssen, wie in der Spezifikation festgelegt, selbst durch eine Relation darstellbar sein[323]. Der hier einzugebende Parameterbezeichner muß entsprechend als Bezeichner einer Relation in der globalen Datenbasis verfügbar sein. Ist dies nicht der Fall, so wird neben der bearbeiteten Relation eine dem unbekannten Parameter entsprechende Relation angelegt. Einziger Parameter dieser weiteren Relation ist die eigentlich bearbeitete Relation. Wird beispielsweise eine Relation *Kind* mit

---

[321] Regel 1 s.S. 92.

[322] Unter einem „gebundenen" oder auch „modalen" Fenster versteht man ein Programmfenster, das, solange es angezeigt wird, einen Zugriff auf die übrigen Programmfunktionen sperrt.

[323] Vgl. Regel 2 s.S. 93.

den Parametern *Mutter* und *Vater* angelegt, und existieren die Relationen *Mutter* und *Vater* noch nicht in der Datenbasis, so legt Altas automatisch die beiden fehlenden Relationen an und weist ihnen den Parameter *Kind* zu.

Es können mehrere Parameter mit demselben Bezeichner verwendet werden. Falls der Parameter in Ausnahmefällen keinerlei Typisierung gestattet, ist als Bezeichner *DING* einzutragen. *DING* ist inhaltslos und somit eine Eigenschaft, die per Definition auf jedes Objekt zutrifft.[324]

cc) Beispiel

Das folgende Beispiel mag das Vorgehen verdeutlichen: In der Datenbasis soll die Relation *Ehe* mit den Parametern *Ehemann* und *Ehefrau* in klassischer Syntax, also *Ehe*(*ehe, ehemann, ehefrau*) neu angelegt werden. Keiner der Bezeichner existiert zuvor in der globalen Datenbasis. Im Systemstatusreport werden für die Datenbasis 10 Relationen ausgewiesen.

1. In der Reportansicht wird der Schalter *Zufügen...* betätigt: Es wird das auf Seite 130 abgebildete Fenster angezeigt.
2. In die Box *Bezeichner* der oberen Sektion wird der Bezeichner *Ehe* eingetragen. Der *Typische Wert* wird auf der Standardeinstellung *Existenz* belassen.
3. In der Box *Bezeichner* der Sektion *Parameter* wird zunächst *Ehemann* eingetragen und der Schalter *Zufügen* betätigt. Danach wird *Ehefrau* eingetragen und wiederum der Schalter *Zufügen* betätigt.

Das Fenster stellt sich nun wie folgt dar:

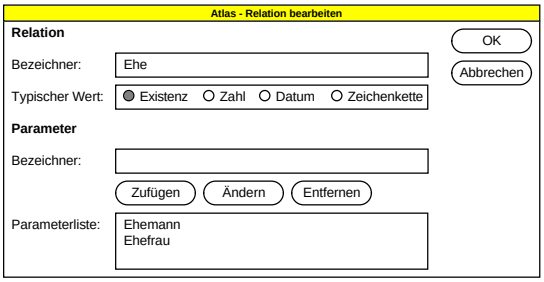


Abbildung 29: Atlas - Relation Bearbeiten

Nach der Betätigung des Schalters *OK* weist das System darauf hin, daß neben der Relation *Ehe* weitere Relationen in der Datenbasis angelegt werden. Im Systemreport werden schließlich 13 Relationen ausgewiesen. Die hinzugekommenen Einträge werden im Report *Relation* wie folgt dargestellt:

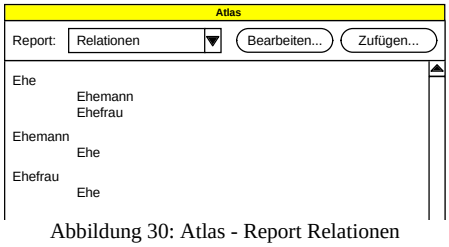


Abbildung 30: Atlas - Report Relationen

Neben der Relation *Ehe*(*ehe, ehemann, ehefrau*) wurden implizit die Relationen *Ehemann*(*ehemann, ehe*) und *Ehefrau*(*ehefrau, ehe*) angelegt. Will man dieses

[324] DING ist im Beispielsprogramm nicht als feste Relation implementiert sondern muß ebenfalls angelegt werden.

Vorgehen vermeiden, dann muß man als Parameter beispielsweise den Eigenschaftsbezeichner *DING* eingeben. Die neuen vom Programm selbständig angelegten Relationen mögen zunächst einer Alltagsansicht widersprechen. Hier ist nämlich der Ehepartner scheinbar nicht Partner des anderen, sondern der Ehe selbst. Betrachtet man dagegen denselben Vorgang bei der Anlage der Relation *Kind*(*kind, vater, mutter*) so führt dies zweifellos zu einem auf Anhieb sinnvollen Ergebnis.

Um jedoch die Fragwürdigkeit des ersten Anscheins bezüglich der Relationen einer Ehe näher zu beleuchten, soll das Familienrecht kurz verlassen werden und als Beispiel der Kaufvertrag dargestellt werden. Bei der Anlage der Relation *Kauf*(*kauf, verkäufer, käufer, kaufgegenstand, kaufpreis*) legt das System unter der Voraussetzung, keiner der Bezeichner ist bekannt, weitere Relationen *Verkäufer*(*verkäufer, kauf*), *Käufer*(*käufer, kauf*), *Kaufgegenstand*(*kaufgegenstand, kauf*) sowie *Kaufpreis*(*kaufpreis, kauf*) an. Umgangssprachlich ist der Käufer ein *Käufer einer Sache*, ebenso wie der Verkäufer *eine Sache verkauft*. Auch der Kaufpreis ist der *Kaufpreis der Sache*, nur die Sache ist der *Vertragsgegenstand*. Die Umgangssprache geht mit einem Wort etwas **wahllos** mit der Bildung von Beziehungen zwischen den an einem Vertrag beteiligten Objekten um. Tatsächlich erscheint es hingegen sinnvoll, alle am Vertrag beteiligten Objekte zunächst in Beziehung zu eben diesem Vertrag zu setzen. Die entsprechenden Relationen beschreiben dann die Rollen, die das jeweilige Objekt in dem gesamten Vertragsszenario spielt.

So mag schließlich und endlich auch die automatisch gefundene Lösung, nach der der Ehemann und die Ehefrau in dieser Beziehung nicht zueinander stehen, sondern quasi diese Rollen in einem Szenario *Ehe* einnehmen, einsichtig sein. Es handelt sich jedenfalls um eine Regelung, die von sprachlichen Eigenheiten abstrahiert und die die notwendige Einheitlichkeit erzielen kann.

c) Objekte und Werte

Der Report *Objekte* zeigt eine Liste aller in einem Fall angelegten Objekte sowie ihrer Werte an:

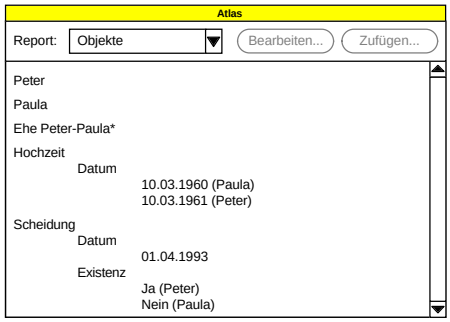


Abbildung 31: Atlas - Report Objekte

Die erste Ebene zeigt die Bezeichner der bisher instantiierten Objekte. Ein Sternchen zeigt an, daß der Bezeichner vom Programm gebildet wurde. Dies geschieht, wenn ein Objekt implizit instantiiert wird, da es in einer Beschreibung eines Faktums vorausgesetzt wurde. Die zweite Ebene dient zur Unterteilung der übermittelten Werte nach ihren Typen. In der dritten Ebene werden die zu einem Objekt bekannten Werte abgelegt. Bestehen Quellenangaben oder Angaben über den Gültigkeitszeitraum, so werden diese in Klammern hinter den Wert geschrieben.

Das Abbild zeigt die kuriose Situation an, daß für die *Hochzeit* zwei unterschiedliche Datumsangaben von Peter und Tina existieren. Bei der *Scheidung* wird zwar der Termin nicht angezweifelt, offensichtlich aber bestreitet Petra die Ernsthaftigkeit der Scheidung.

Ein Eingriff des Benutzers auf die Daten ist an dieser Stelle nicht vorgesehen.

d) Beziehungen

Der Report *Beziehungen* zeigt eine Liste der konkret an das System übermittelten Beziehungen:

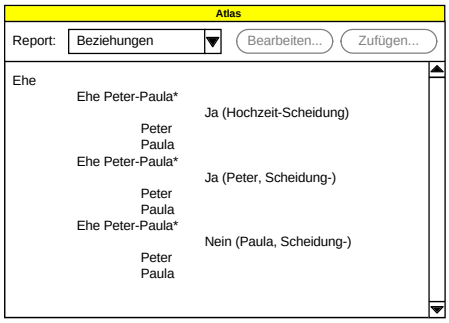


Abbildung 32: Atlas - Report Beziehungen

In der ersten Ebene wird der Bezeichner der einer Beziehung zugrundeliegenden Relation angegeben. In der zweiten Ebene erfolgt die Auflistung der Bezeichner des Objekts, welches durch die Relation selbst (funktional) repräsentiert wird. In der dritten Ebene werden die Bezeichner der Objekte dargestellt, die die weiteren Parameter ausfüllen. Optional werden in der vierten Ebene Angaben über die tatsächliche Existenz der Beziehungen sowie die Informationen über die Quelle und den Gültigkeitszeitraum dargestellt.

Dieses Abbild konkretisiert die im vorangegangenen Bild gezeigte Streitlage. Einig sind sich offensichtlich beide Ehepartner über das Bestehen der Beziehung zwischen der Hochzeit und der Scheidung. In dem darauffolgenden Zeitraum wird die Beziehung hingegen von Peter angezweifelt, da er von der Rechtsgültigkeit der Scheidung am 1. April ausgeht.

e) Filter

Der Report *Filter* zeigt den aktuellen Status der in Atlas vorgesehenen Filter an. Als Filter werden in diesem Zusammenhang Auswahlkriterien bezeichnet, die die Sicht auf den gesamten Bestand an Informationen einschränken (*Ausgabefilter*) bzw. die einfließenden Daten durch Zusatzinformationen kanalisieren (*Eingabefilter*). Filter dienen im Prinzip der Realisierung der Zusatzinformationen, die mit jeder Aussage abgelegt werden[325].

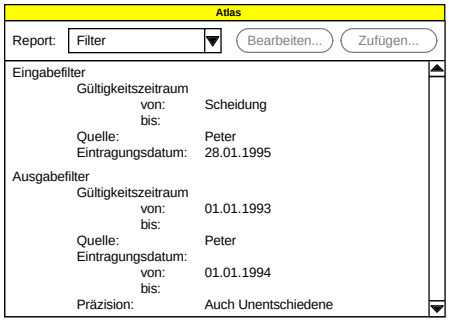


Abbildung 33: Atlas - Report Filter

[325] Vgl. S. 100.

Der *Eingabefilter* legt die Informationen fest, die mit einem neu übermittelten Wert oder einer Beziehung abgelegt werden. Für den Gültigkeitszeitraum werden zwei Objektbezeichner angegeben. Fehlt ein Bezeichner, so gilt der Bereich in diese Richtung als offen. Der Gültigkeitszeitraum im Beispiel beginnt mit der Scheidung und reicht jeweils bis zum Datum der konkreten Anfrage. Bei der *Quelle* wird ebenfalls der Bezeichner eines Objekts angegeben. Im Beispiel ist dies *Peter*. Für das Eintragungsdatum verwendet Atlas standardmäßig das Systemdatum. Es kann jedoch willentlich verändert werden.

Der *Ausgabefilter* grenzt den Bereich der Werte ein, die von Atlas bei der Anfrage nach Werten berücksichtigt werden sollen. In diesem Fall werden alle Datumsangaben als Datumswerte angezeigt, auch diejenigen des Gültigkeitszeitraums[326]. Unter *Präzision* wird angezeigt, ob zusätzlich zu den Fakten, die dem Filter entsprechen, auch diejenigen ausgegeben werden, bei denen keine Angabe zu den entsprechenden Kriterien vorliegen.

f) Module

Der Report *Module* liefert Informationen zu den eingesetzten Modulen. Module dieser Art liefern Lösungen zur Ermittlung spezifischer Werte oder zur Behandlung von Anfragen, die vom System selbst nicht beantwortet werden können. Sie können den Standardfunktionsumfang des Programms flexibel erweitern.

Der Report *Module* entspricht etwa dem folgenden Abbild:

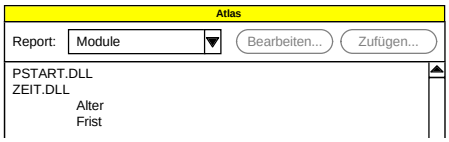


Abbildung 34: Atlas - Report Module

In der ersten Ebene werden die Dateinamen der Filter aufgelistet. In der zweiten Ebene werden die Funktionen angezeigt, die das Modul bereitstellt[327].

Die Benutzeroberfläche von Atlas bietet sechs hierarchisch aufgebaute Reports. Diese ermöglichen eine Übersicht über die im System enthaltenen globalen Informationen, die Fakten des aktuellen Falls sowie systemtechnische Angaben. Im Report *Relationen* ist zudem die Pflege der globalen Datenbasis möglich.

2. DDE-Schnittstelle

Die DDE-Schnittstelle ist die zentrale Schnittstelle für Anwendungen, um mit Atlas zu kommunizieren. Diese wird genutzt, um Daten an das System zu übermitteln oder von dort zu erfragen. Zudem gibt es Funktionen, um auf den Ablauf von Atlas Einfluß zu nehmen. Im folgenden Abschnitt wird diese Schnittstelle präzise beschrieben.

a) Syntax

Wie bei jeder formalen Sprache, muß bei der Kommunikation eine korrekte Syntax entwickelt und vom Anwender beziehungsweise Anwendungsprogramm eingehalten werden. Bei der Entwicklung einer solchen Syntax müssen die Bedürfnisse für eine technisch optimale Umsetzung mit der Verständlichkeit der gebildeten Ausdrücke in Einklang gebracht werden. Die hier verwendete Syntax achtet vor allem darauf, daß die Ausdrücke auch für einen Laien nach kurzer Einarbeitung lesbar sind. Dies ist vor allem dort nötig, wo der Endnutzer unmittelbar mit den Anfragen oder Befehlen

[326] Näheres hierzu s.S. 151.

[327] Näheres hierzu s.S. 171.

konfrontiert wird. Das dürfte vorwiegend in Textverarbeitungsdokumenten oder Formularen der Fall sein, wo Anfragen als Felder im Klartext einsehbar sind.

aa) Interne Funktionen

Interne Funktionen sind solche, die von Atlas selbst zur Verfügung gestellt werden. Sie erfüllen vielfältige Aufgaben, wie etwa die Beschreibung des Wertetyps eines Objekts oder auch das Abspeichern von Daten. Über Zusatzmodule kann der Standardumfang an internen Funktionen individuell erweitert werden.

Der Aufruf interner Funktionen zeigt das folgende Syntaxdiagramm. Dabei werden konstant auftretende Zeichen halbfett und variable Zeichen kursiv dargestellt.

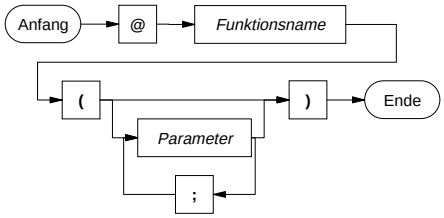


Abbildung 35: Syntax der internen Funktionen

Das Zeichen @ steht als Präfix für interne Funktionen. Auf diese Weise können Beziehungen und interne Funktionen selbst dann unterschieden werden, wenn sie denselben Bezeichner haben. Die Parameter sind abhängig von den einzelnen Funktionen. Sie werden später im einzelnen beschrieben. Regelmäßig müssen alle Parameter ausgefüllt werden. Die ausdrückliche Angabe einer internen Funktion kann in bestimmten Einzelfällen verzichtbar sein.

bb) Bezeichner

Objekte und Relationen werden durch einen **Bezeichner** intern identifiziert. D.h. Objekte mit identischen Bezeichnern werden wie ein Objekt behandelt. Objekte mit divergierenden Bezeichnern werden wie unterschiedliche Objekte behandelt. Als Bezeichner kann jede Zeichenkette mit maximal 32 beliebigen Zeichen verwendet werden. Nicht zugelassen sind die Zeichen *$* und @ sowie #. Das Leerzeichen kann nur in Objektbezeichnern eingesetzt werden[328].

Wird ein Bezeichner der beschriebenen Form eingesetzt, so muß das System in den existierenden Bezeichnern nach dem verwendeten Bezeichner suchen. Es durchläuft dabei die Liste der Objekte oder Relationen, bis der verwendete Bezeichner gefunden wird. Bei großen Listen kann dies zu erheblichen Einbußen im Laufzeitverhalten einer Anwendung führen. Atlas erlaubt deshalb anstelle der Angabe einer Zeichenkette auch den Verweis auf die Indexposition eines Objekts bzw. einer Relation. Diese Angabe folgt der Syntax

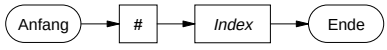


Abbildung 36: Syntax der Indexposition

Das sogenannte Nummernzeichen # leitet den Ausdruck ein. Danach folgt die Position des Bezeichners in der entsprechenden Liste. Der Ausdruck wird anstelle der Zeichenkette eingesetzt. Insbesondere wird bei Objektbezeichnern diesem Ausdruck das Objektpräfix *$* vorangestellt[329]. Atlas besitzt Funktionen, mit denen ein Pro-

[328] In der EDV hat sich allgemein der Brauch durchgesetzt, anstelle des Leerzeichens in Bezeichnern einen Unterstrich ("_") einzusetzen.

[329] S.u.

gramm die Listenpositionen von Objekten und Relationen in Erfahrung bringen kann[330].

cc) Beziehungen

Beziehungen verbinden die Objekte eines Falls über eine Relation miteinander. Sie ermöglichen es so, quasi den Weg zu einem Faktum zu beschreiben. Die Konzeption von Beziehungen geht davon aus, daß diese immer auch in funktionaler Syntax dargestellt werden können[331]. So kann die Beziehung eines Kindes *K* zu seinen Eltern *V* und *M* einerseits als Relation *Kind*(K, V, M), andererseits aber auch als Funktion $K = Kind(V, M)$ dargestellt werden. Atlas verwendet eine zwitterartige Syntax, indem es dem ersten Parameter einer Beziehung die Funktionalität eines *Hauptparameters* zuweist. Gleichzeitig bildet die Beziehung die übrigen Parameter auf den Hauptparameter ab, gibt diesen also als Ergebnis zurück. Die allgemeine Syntax der Beschreibung einer Beziehung lautet wie folgt:

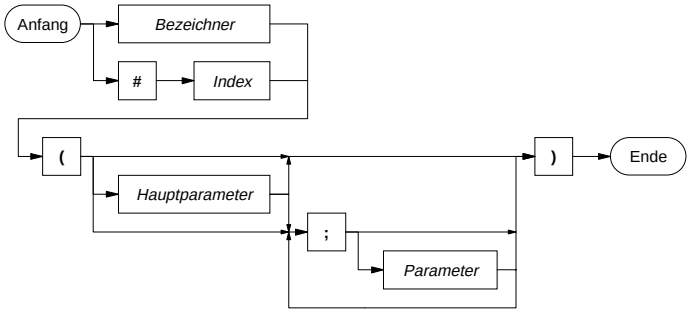


Abbildung 37: Syntax einer Beziehung

Als Parameter sind konkrete Objektbezeichner oder Beziehungen zulässig, die andere Objekte auf ein Objekt abbilden. Letztere Option erlaubt nahezu beliebig verschachtelte Ausdrücke. Der Hauptparameter und nahezu alle weiteren Parameter sind optional. Selbst wenn die Angabe eines Parameters entfällt, muß jedoch das trennende Semikolon eingesetzt werden. Dabei verhält sich Atlas je nach Situation unterschiedlich, wenn Parameter entfallen.

Einzelne Beispiele zu diesem Thema werden im Abschnitt Fakten austauschen[332] beschrieben. Den Bezeichner der Beziehung sowie Anzahl und Art der Zusatzparameter entnimmt Atlas aus der globalen Datenbasis. Jede Relation muß zunächst im System installiert werden.

[330] S. S. 155.

[331] S.o.

[332] S. S. 139 ff.

dd) Objekte

Objekte bezeichnen real existierende Sachen, Personen oder Einrichtungen[333]. Objekte werden durch einen Objektbezeichner symbolisiert. Der Objektbezeichner kann von einem Anwendungsprogramm vergeben werden. Atlas selbst vergibt Objektbezeichner, wenn in einem Ausdruck ein Objekt vorausgesetzt, aber nicht benannt wird. Die Angabe eines Objektbezeichners folgt folgender Syntax:

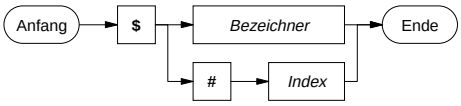


Abbildung 38: Syntax des Objektbezeichners

Der Bezeichner darf aus maximal 32 beliebigen Zeichen mit Ausnahme der Zeichen *$* und *@* bestehen. Bei der Verwendung von Leerzeichen kann es zu syntaktisch bedingten Konflikten kommen. Das Zeichen *$* dient als Präfix für Objektbezeichner, um eine Unterscheidung zu Beziehungen und internen Funktionen zu ermöglichen. Auf diese Weise kann etwa ein Objekt den Bezeichner *Kind* tragen, ohne mit der Relation *Kind* in Konflikt zu geraten.

ee) Wertangaben

Objekte können wie bereits beschrieben Werte unterschiedlichen Typs besitzen. Die Eingabe dieser Werte muß einer definierten Syntax folgen, damit eine Anwendung einen Wert korrekt interpretieren kann. Atlas akzeptiert von anderen Anwendungen in der Regel mehrere Varianten von Wertangaben, beschränkt sich jedoch bei der Ausgabe von Werten auf eine Möglichkeit.

aaa) Zeichenketten

Als **Zeichenketten** sind beliebige Zeichenfolgen der unter Windows verfügbaren Zeichen zulässig.[334]

bbb) Zahlen

Zahlen werden mit arabischen Ziffern dargestellt. Als Dezimaltrennzeichen wird das Komma interpretiert. Ein Punkt innerhalb einer Zahl wird ignoriert. Er kann als Trennung zwischen Tausenderstellen eingesetzt werden. Die Ausgabe der Zahlen erfolgt mit zwei Dezimalstellen und ohne Tausendertrennung. Wird beispielsweise die Zahl *2.345,4* als Wert übermittelt, so wird auf Anfrage der Wert *2345,40* zurückgegeben[335].

ccc) Datumsangaben

Datumsangaben erfolgen in deutscher Notation. Sie können mit zwei- oder vierstelliger Jahresangabe erfolgen. Der Monat kann als Zahl oder mit dem Namen angegeben werden. Die Angabe der ersten drei Buchstaben eines Monats ist ausreichend. Als Trennzeichen wird ein Punkt oder eine Leerstelle oder beides akzeptiert. Weiterhin kann eine Ganzzahl als Datum interpretiert werden. Dabei bezeichnet diese Zahl die Anzahl der Tage, die seit dem 31.12.1899 verstrichen sind. Die folgenden Datumsangaben sind beispielsweise zulässig:

23.8.94
23 8 94
23.08.1994

---

[333] S.o.

[334] Beim Prototypen beträgt die maximale Länge einer Zeichenkette 128 Zeichen.

[335] Der Prototyp rundet alle Fließkommazahlen auf zwei Dezimalstellen.

23. Aug 94
23 August 1994
34569

Ausgegeben wird immer ein Datum mit zweistelliger Tages- und Monatsangabe sowie vierstelliger Jahresangabe. Trennzeichen bei der Ausgabe ist ein Punkt. Im Beispiel wird also unabhängig von der Eingabe bei jeder Anfrage immer der Wert *23.08.1994* zurückgegeben.

ddd) Logische Werte

Atlas kennt die logischen Werte *wahr*, *falsch* und einen Wert für eine *unklare Situation*. Die folgende Tabelle beschreibt die möglichen Eingaben und die von Atlas zurückgegebenen Werte:

Logischer Wert	Eingabe	Ausgabe
wahr	w j t Zahl ≠ 0	Ja
falsch	f n 0	Nein
unklar	u v p	Unklar

Atlas interpretiert nur den ersten Buchstaben eines Wortes. Groß- und Kleinschreibung werden identisch behandelt. Entsprechend können auch die Ausdrücke *wahr*, *falsch*, *JA*, *NEIN*, *vielleicht* etc. verwendet werden.

b) Darstellung der Transaktionen

Dieser und die folgenden Abschnitte behandeln nun den Funktionsumfang, den Atlas per DDE teilnehmenden Anwendungen zur Verfügung stellt. Um den Vorgang einerseits technisch korrekt, andererseits praktisch nachvollziehbar zu veranschaulichen, werden die Beispiele anhand einer Tabelle dargestellt, die in den Grundzügen die Transaktionen des DDE-Protokolls widerspiegelt. Die Darstellung ist so allgemein gehalten, daß eine Abstraktion von der eigentlichen Windows-typischen Realisierung erreicht wird. Sie hat folgendes Muster:

Aktion	**Beschreibung**	**Wert**
Anfrage	Beschreibung der Anfrage	
Antwort		X

Die erste Spalte beschreibt die Transaktionsform. Der Inhalt der weiteren Spalten ist von der gewählten Transaktion abhängig. Möglich sind folgende Transaktionen:

Anfrage — Eine Anwendung schickt eine Anfrage nach einem bestimmten Wert an Atlas. In der zweiten Spaltung wird der Wert beschrieben. Die dritte Spalte bleibt frei.

Antwort — Atlas antwortet auf eine Anfrage mit einem Wert. Die zweite Spalte bleibt frei, die dritte Spalte enthält den Wert.

Übermittlung	Eine Anwendung übermittelt an Atlas einen Wert. In der zweiten Spalte wird der Wert beschrieben, in der dritten Spalte befindet sich der Wert selbst.
Befehl	Eine andere Anwendung sendet an Atlas einen systeminternen Befehl. Im Gegensatz zu den Konventionen des DDE-Protokolls wird der Befehl in der zweiten Spalte angegeben. Die dritte Spalte bleibt frei.
Verbinden	Eine Anwendung stellt eine Verbindung zu Atlas her. Die zweite und dritte Spalte bleiben frei.
Abbrechen	Eine mit Atlas verbundene Anwendung bricht die Verbindung ab. Die zweite und dritte Spalte bleiben frei.

c) Verbindungsaufnahme

Im DDE-Protokoll wird für die Verbindungsaufnahme der Anwendungsname des Partners sowie ein Konversationsthema benötigt[336]. Atlas wird über den Anwendungsnamen *ATLAS* angesprochen. Mögliche Themen der Konversation sind *System*[337] oder *eine Akte*.

d) Fakten austauschen

Hauptanliegen von Atlas ist der Transport von Fakten von einer Anwendung zu einer anderen. Dieser vollzieht sich in zwei Stufen: Zunächst werden die Fakten praktisch ungefragt von einer Anwendung an Atlas übermittelt, danach werden sie auf Anfrage an jede andere Anwendung weitergegeben[338]. Fragt eine Anwendung nach Fakten, die Atlas unbekannt sind, so erhält sie standardmäßig eine negative Antwort. Das Verfahren kann allerdings über ein Erweiterungsmodul dahingehend modifiziert werden, daß dieses Modul eine andere Anwendung aufruft und auffordert, die entsprechenden Fakten zu ermitteln und an Atlas zu übertragen[339].

Die folgenden Punkte zeigen schrittweise anhand von Beispielen, wie Daten an Atlas übermittelt und abgefragt werden können. Dabei wird aufgezeigt, welche Auswirkungen Anfragen und Übermittlungen auf den Bestand von Objekten, deren Werte und deren Beziehungen zueinander haben. Die verwendeten Relationen müssen in der globalen Datenbasis vorhanden sein.

aa) Werte von Objekten

Werte eines bestimmten Typs werden durch interne Relationen, das heißt Relationen, die dem System immanent sind, angesprochen[340]. Es bestehen folgende Relationen:

@Zahl(x)
@Datum(x)
@Name(x)
@Existiert(x)

`@Zahl` gibt den numerischen Wert eines Objekts zurück und `@Datum` den Datumswert. Die Relation `@Name` bezeichnet eine Zeichenkette und `@Existiert` einen logischen Wert. Als *x* kann zunächst der Bezeichner eines Objekts eingesetzt

---

[336] Vgl. S. 110.

[337] Das Thema *System* wird per Definition von jeder DDE-Anwendung unter Windows bedient.

[338] Vgl. S. 101.

[339] S. S. 115.

[340] Vgl. Regel 6 s.S. 97.

werden. Soll etwa dem System direkt mitgeteilt werden, daß ein Objekt *Peter* tatsächlich existiert, so geschieht das wie folgt:

Aktion	**Beschreibung**	**Wert**
Übermittlung	@Existiert($Peter)	Ja

Atlas instantiiert das Objekt mit dem Bezeichner *Peter* und weist ihm den Existenzwert *wahr* zu. Im folgenden Beispiel wird ein Ereignis *Peters Geburtstag* instantiiert. Ihm wird der Wert *23.09.1972* zugewiesen:

Aktion	**Beschreibung**	**Wert**
Übermittlung	@Datum($Peters Geburtstag)	23.Sept 72

Das folgende Schaubild zeigt die neu angelegten Objekte und Werte:

Objekte

Bezeichner	Wertetyp	Werte
Peter	Existenz	Ja
Peters Geburtstag	Datum	23.09.1972

Hier muß klargestellt werden, daß Atlas bisher noch keine Beziehung zwischen den Objekten *Peter* und *Peters Geburtstag* kennt. Die Abfrage eines Datums erfolgt analog zur Datenübermittlung. Im folgenden Abbild wird eine Anfrage nach dem Datumswert des Objekts *Peters Geburtstag* an Atlas geschickt.

Aktion	**Beschreibung**	**Wert**
Anfrage	@Datum($Peters Geburtstag)	

Die Antwort auf diese Frage lautet folgendermaßen:

Aktion	**Beschreibung**	**Wert**
Antwort		23.09.1972

Einziger Unterschied zur vorangegangenen Übermittlung des Datums ist, daß das Datum im Standardformat *TT.MM.JJJJ*[341] zurücktransferiert wurde.

bb) Mehrere Angaben von Werten

Atlas legt, wie mehrfach beschrieben wurde, jeden übermittelten Wert selbst dann gesondert ab, wenn es sich jedesmal um denselben Wert handelt. Das Programm geht davon aus, daß die Werte einem Objekt in Form einer Werteliste zugeordnet werden. Angenommen, im obigen Beispiel behauptet nun Peters Mutter, Peter sei erst am 23.9.73, also ein Jahr später geboren. Dieses Datum wird wie folgt an das System übermittelt:

Aktion	**Beschreibung**	**Wert**
Übermittlung	@Datum($Peters Geburtstag)	23. 9. 73

Nach der Übermittlung zeigt der Objektreport folgendes Bild:

Objekte

Bezeichner	Wertetyp	Werte
Peter	Existenz	Ja
Peters Geburtstag	Datum	23.09.1972 23.09.1973

[341] D.h. genau zweistellige Tages- und Monatsangabe sowie viertstellige Jahresangaben (s.S. 137).

Es werden nun beide übermittelten Daten bereitgehalten. Für die oben bereits beschriebene Anfrage nach dem Datumswert von *Peters Geburtstag* ergibt sich folgende Transaktion:

Aktion	Beschreibung	Wert
Anfrage	@Datum($Peters Geburtstag)	
Antwort		23.09.1973

Der Zugriff auf einen anderen als den zuletzt übermittelten Wert erfolgt über einen vom System standardmäßig zur Verfügung gestellten Parameter. Da der Parameter nicht fester Bestandteil der Parameterleiste einer Relation ist, wird er durch den Systembezeichner @Pos(Index) gekennzeichnet. Er wird durch ein Semikolon getrennt hinter den festen Parametern der Relation eingetragen. Der Parameter kann in systeminternen Relationen genauso wie in Relationen der globalen Datenbasis eingesetzt werden. *Index* bezeichnet dabei die Position des gesuchten Wertes in der Werteliste. Die Anfrage nach dem zuerst eingegebenen Datum würde entsprechend wie folgt lauten:

Aktion	Beschreibung	Wert
Anfrage	@Datum($Peters Geburtstag;@Pos(1))	
Antwort		23.09.72

Da es nicht möglich ist, einmal eingegebene Werte zu ändern, wird der Parameter @Pos(n) ignoriert, wenn er bei der Übergabe von Werten eingesetzt wird. Der übergebene Wert wird also an die Werteliste des bezeichneten Objekts immer hinten angehängt.

cc) Zugriff auf Werte über Beziehungen

Bisher wurde der direkte Zugriff auf den Wert eines Objekts durch Angabe des Wertetyps sowie des konkreten Objekts beschrieben. Der wohl praktisch weitaus relevantere Zugriff erfolgt jedoch über eine oder mehrere ineinander verschachtelte Relationen.

Um dieses Vorgehen zu demonstrieren soll wiederum von einer leeren Akte ausgegangen werden. Wiederum soll zunächst der Geburtstag von *Peter* mit dem Wert *23.09.1972* übermittelt werden.

aaa) Übermittlung

Hierzu müssen das Objekt *Peter* sowie ein Objekt, das den Geburtstag repräsentiert, angelegt werden. Weiterhin muß die Beziehung *Geburtstag* zwischen den beiden Objekten angelegt werden. Für die Anlage des Objekts, welches den Geburtstag, also den Hauptparameter der Relation darstellt, stellt Atlas zwei Möglichkeiten zur Verfügung, eine **explizite** und eine **implizite**. Bei der expliziten wird für dieses Objekt ein eigener Objektbezeichner angegeben, den Atlas verwenden soll. Wird hingegen kein Objektbezeichner angegeben, so legt das Programm selbständig einen solchen an. Die explizite Variante ist nur dann sinnvoll, wenn auf das Objekt auch direkt zugegriffen werden soll. Auch auf implizit angelegte Objekte kann unmittelbar zugegriffen werden. In diesem Fall muß jedoch der Objektbezeichner beim System erfragt werden. Die Systematik, nach der die Bezeichner gebildet werden, darf nicht als dauerhaft verläßlich behandelt werden.

In der folgenden Transaktion wird über die Relation $g = Geburtstag(mensch)$ der Geburtstag von Peter explizit mit *Peters Geburtstag* bezeichnet und auf das Datum *23.9.72* festgelegt:

Aktion	Beschreibung	Wert
Übermittlung	Geburtstag($Peters Geburtstag;$Peter)	23.9.72

Der Hauptparameter wird als erster in der Parameterleiste eingetragen. In einem weiteren Beispiel wird der Geburtstag von *Tina* auf den *12.5.73* festgelegt, ohne daß eine explizite Nennung des Bezeichners erfolgt:

Aktion	**Beschreibung**	**Wert**
Übermittlung	Geburtstag(;$Tina)	12.5.73

Um die Anzahl der Parameter gleich zu halten, muß selbst dann ein trennendes Semikolon vor den zweiten Parameter gesetzt werden, wenn der erste Parameter unbelegt bleibt. Nach diesen beiden Übermittlungen zeigt der Objektreport folgende Eintragungen:

Objekte

Bezeichner	Wertetyp	Werte
Peter		
Tina		
Peters Geburtstag	Datum	23.09.1972
Geburtstag Tina*	Datum	12.04.1973

Atlas hat also vier Objekte instantiiert. Das Objekt *Geburtstag Tina* wurde dabei implizit angelegt und vom System benannt. Dies wird durch ein Sternchen angezeigt. Die Benennung erfolgt nach folgendem Schema:

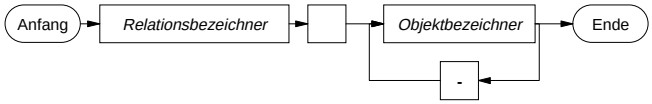


Abbildung 39: Syntax eines automatisch angelegten Objektbezeichners.

Zu Beginn steht also der Name der Relation, mit deren Hilfe der Wert übermittelt wurde. Nach einer Leerstelle folgen die Bezeichner der Objekte, die die weiteren Parameter ausfüllen. Letztere werden mit Bindestrich verbunden.

Weiterhin hat Atlas den Geburtstagen die entsprechenden Datumswerte zugewiesen. Dies mag zunächst verwundern, da bei der Übermittlung der Werte auf die Systemrelation $y=Datum(x)$ verzichtet wurde. Wie jedoch bereits zuvor beschrieben[342], wird bei der Anlage einer Relation ein Standardwertetyp festgelegt. Soll ein Wert dieses Typs über eine Relation abgefragt oder übermittelt werden, so ist die ausdrückliche Benennung dieses Typs über die entsprechende Systemrelation verzichtbar. In diesem Fall hat die Relation $g = Geburtstag(mensch)$ als Standardwertetyp *Datum*, weshalb z.B. der Ausdruck @Datum(Geburtstag(;$Tina)) auf den verwendeten Ausdruck Geburtstag(;$Tina) verkürzt werden kann.

Neben den Objekten hat Atlas aufgrund der beschriebenen Übermittlungen auch zwei Beziehungen angelegt, die im Beziehungsreport entsprechend dem folgenden Abbild dargestellt werden:

Beziehungen

Relation	Hauptparameter	Parameter	Existenz
Geburtstag	Peters Geburtstag	Peter	
	Geburtstag Tina*	Tina	

Zwar dienten die Übermittlungsvorgänge zunächst dem Transfer der beiden Datumswerte. Sie haben jedoch die Beziehungen *Peters Geburtsag* = $Geburtstag(Peter)$ und *Geburtsag Tina* = $Geburtstag(Tina)$ postuliert. Um also einen Zugriff über diese Beziehung zu ermöglichen, hat das System dieses Postulat

[342] S. S. 97 und 130.

in eine quasi hypothetische Beziehung umgesetzt. Dementsprechend enthält es keinerlei Informationen darüber, ob die Beziehung als tatsächlich existent angenommen wird.

bbb) Anfrage

Bestehen zwischen mehreren Objekten Beziehungen, so kann ein Wert nicht nur über die Objekte direkt, sondern auch indirekt über eine Beschreibung einer Beziehung wiederum abgefragt werden. Hierzu wird im einfachsten Fall eine Relation angegeben und die entsprechenden Parameter werden mit Objektbezeichnern ausgefüllt. Der Hauptparameter einer Relation repräsentiert das Objekt, welches durch die Relation selbst identifiziert werden soll. Bei einer Anfrage nach einem Wert ist es Zweck der Verwendung einer Beziehung, das Objekt von Atlas ermitteln zu lassen, welches den letztlich gesuchten Wert hat. Deshalb ist es bei einer Anfrage keinesfalls sinnvoll, den ersten Parameter einer Beziehung mit einem Objektbezeichner auszufüllen. Kennt die anfragende Anwendung nämlich den Objektbezeichner des Objekts, so kann der Wert direkt über das Objekt erfragt werden. Eine Beschreibung mit Hilfe einer Beziehung erübrigt sich entsprechend.

Eine Anfrage nach dem Geburtsdatum von *Peter* sieht deshalb im Normalfall folgendermaßen aus:

Aktion	**Beschreibung**	**Wert**
Anfrage	@Geburtstag(;$Peter)	

Wie bei der Übermittlung von Werten ist die Angabe des Wertetyps dann entbehrlich, wenn dieser mit dem Standardtyp der Relation übereinstimmt. Ebenfalls entsprechend der Übermittlung eines Wertes, muß der fehlende Hauptparameter mit einem Semikolon dargestellt werden. Auf diese Anfrage hin sucht Atlas das Objekt, welches den Hauptparameter ausfüllt, und liefert das in der Werteliste des Objekts zuletzt eingetragene Datum.

Aktion	**Beschreibung**	**Wert**
Antwort		23.09.1972

Eine Anfrage nach dem Datumswert des Objekts *Peters Geburtstag* liefert dasselbe Ergebnis:

Aktion	**Beschreibung**	**Wert**
Anfrage	@Datum($Peters Geburtstag)	
Antwort		23.09.1972

dd) Übermittlung mehrerer Werte über Beziehungen

Wie auch bei der unmittelbaren Übermittlung von Werten, werden bei der Übermittlung mittels einer Beziehung alle Werte in die Werteliste des beschriebenen Objekts eingetragen. Dabei ist es unschädlich, ob der Hauptparameter explizit angegeben ist oder nicht. Das folgende Beispiel übermittelt wiederum einen alternativen Datumswert für Peters Geburtstag:

Aktion	**Beschreibung**	**Wert**
Übermittlung	Geburtstag(;$Peter)	23.9.73

Atlas sucht zunächst, ob ein Objekt besteht, welches die Relation erfüllt. Findet es dieses, so trägt das Programm den Wert in die Werteliste des gefundenen Objekts ein. Im Objektreport ergibt dies folgendes Bild:

Objekte

Bezeichner	Wertetyp	Werte
Peter		

Peters Geburtstag	Datum	23.09.1972 23.09.1973

Da die angegebene Beziehung lediglich als Postulat und nicht als Aussage interpretiert wird, wird sie in der Beziehungsliste auch nicht erneut abgelegt. Der Zugriff auf bestimmte Werte innerhalb der Liste erfolgt wiederum mit dem Systemparameter *@Pos*(n):

Aktion	**Beschreibung**	**Wert**
Anfrage	Geburtstag(;$Peter);@Pos(1)	
Antwort		23.09.1972

Pos tritt dabei nicht als Parameter von *Geburtstag*, sondern als Parameter der weggelassenen Systemrelation *Datumswert* auf. Er wird deshalb nicht innerhalb der Klammern positioniert. Atlas gibt als Wert *23.09.1972* zurück. Denselben Wert erhält man auch mit der Anfrage

Aktion	**Beschreibung**	**Wert**
Anfrage	@Datum(Geburtstag(;$Peter);@Pos(1))	
Antwort		23.09.1972

ee) Verschachtelte Ausdrücke

In vielen Fällen ist es nicht ausreichend, das Objekt, dessen Wert erfragt wird, über eine Beziehung zu beschreiben. Sucht man beispielsweise den Geburtstag der Mutter von Peter, so sind bereits zwei Schritte nötig. Zu diesem Zweck erlaubt Atlas beliebige Verschachtelungen von Relationen. Der Geburtstag kann dem System beispielsweise folgendermaßen übermittelt werden:

Aktion	**Beschreibung**	**Wert**
Übermitteln	Geburtstag(;Mutter(;$Peter))	17.02.1945

Die folgenden Tabellen stellen die hieraus resultierenden Ausschnitte aus dem Objekte– und aus dem Beziehungsreport dar:

Objekte

Bezeichner	Wertetyp	Werte
Peter		
Mutter Peter*		
Geburtstag Mutter Peter*	Datum	17.02.1945

Beziehungen

Relation	Hauptparameter	Parameter	Existenz
Mutter	Mutter Peter*	Peter	
Geburtstag	Geburtstag Mutter Peter*	Mutter Peter	

Atlas bearbeitet einen verschachtelten Ausdruck von innen nach außen. Deshalb wurde hier zunächst implizit das Objekt *Mutter Peter* und danach das Objekt *Geburtstag Mutter Peter* angelegt. Die bereits sehr komplexen Objektbezeichner, die von Atlas angelegt wurden, zeigen, daß hier eine explizite Nennung des Objekts, sofern sie möglich ist, auch sinnvollerweise eingesetzt werden sollte. Die Beschreibung

```
Geburtstag(;Mutter($Gisela;$Peter))
```

bei der Übermittlung informiert Atlas en passant, daß Gisela die Mutter von Peter ist und übermittelt deren Geburtstag. So werden zwei Fakten gleichzeitig transportiert und die Objektbezeichner sind besser lesbar als im ersten Fall.

Objekte

Bezeichner	Wertetyp	Werte
Peter		
Gisela		
Geburtstag Gisela*	Datum	17.02.1945

Beziehungen

Relation	Hauptparameter	Parameter	Existenz
Mutter	Gisela	Peter	
Geburtstag	Geburtstag Gisela	Gisela	

Das Problem liegt darin, daß im Regelfall der Objektbezeichner eben unbekannt ist oder nur in mehreren Stufen ermittelt werden kann. Ausgehend von den letzten Transaktionen führen folgende Anfragen zu identischen Antworten:

```
Geburtstag(;Mutter(;$Peter))
Geburtstag(;$Gisela)
@Datum(;$Geburtstag Gisela)
```

Die erste Beschreibung verwendet dieselbe Verschachtelung wie die Übermittlung. Hier ist es jedoch wie bei jeder Anfrage nicht sinnvoll, den Objektbezeichner in den Hauptparameter einzutragen, da es gerade dieser Bezeichner ist, der vom System ermittelt werden soll. Bei der zweiten Anfrage ist der Bezeichner der Mutter bekannt. Bei der dritten Anfrage ist auch der Bezeichner des Geburtsdatums positiv bekannt. Hier wird der Datumswert des Objekts ermittelt.

ff) Existenz von Beziehungen

Die bisherigen Beispiele gingen davon aus, daß Beziehungen bei einer Datenübermittlung postuliert wurden. Atlas legt daraufhin die Beziehung an, weist ihr jedoch keinerlei Existenzwert zu. Das System erlaubt es nun auch, Aussagen darüber zu machen, ob eine Beziehung tatsächlich besteht oder nicht. Diese Aussage ist von dem Existenzwert des durch eine Relation abgebildeten Objekts zu unterscheiden. Betrachtet man die Beziehung *Clemens = Kind*$(\mathit{Peter, Tina})$, so hat die Aussage *Clemens existiert nicht* eine vollkommen andere Bedeutung als *Clemens ist nicht das Kind von Peter und Tina*. Bei der folgenden Übermittlung geht Atlas davon aus, daß die Aussage den Existenzwert des Kindes betrifft, sofern die Relation *Kind* den Standardwertetyp *Existenz* hat:

Aktion	**Beschreibung**	**Wert**
Übermitteln	Kind($Clemens;$Peter;$Paula)	Ja

Eine Anfrage nach der Existenz von CLEMENS wird entsprechend positiv beantwortet. Die Einbettung in die Beziehung *Kind* war in diesem Fall also überflüssig. Damit nun Aussagen über das Bestehen oder Nichtbestehen einer Beziehung übermittelbar sind, besitzt Atlas eine systeminterne Relation,

```
@Beziehung(beziehung)
```

die eine Beziehung auf einen Wahrheitswert abbildet. Die oben abgebildete Übermittlung muß entsprechend abgeändert werden, um die Aussage *Clemens ist Kind von Peter und Tina* zu übermitteln:

Aktion	**Beschreibung**	**Wert**
Übermitteln	@Beziehung(Kind($Clemens;$Peter;$Paula))	Ja

Im Beziehungsreport spiegelt sich diese Übermittlung wie folgt wider:

Beziehungen

Relation	Hauptparameter	Parameter	Existenz
Kind	Clemens	Peter Paula	Ja

Auch in diesem Zusammenhang ist es möglich, den Hauptparameter offen zu lassen. Das System legt auch dann implizit ein dem Kind entsprechendes Objekt an. Im Beispiel trüge dies den Bezeichner *Kind Peter–Tina*. Es ist jedoch nur in wenigen Fällen sinnvoll, eine Aussage über eine Beziehung zu machen und ein an der Beziehung beteiligtes Objekt nicht zu nennen. Die Aussage hieße im Klartext: *Peter und Paula haben jedenfalls ein Kind, das allerdings unbekannt ist.* Anders wäre dies bei einer Anfrage wie etwa *Ist Clemens das Kind von Peter und Tina?*

Aktion	**Beschreibung**	**Wert**
Anfrage	@Beziehung(Kind($Clemens;$Peter;$Paula))	
Antwort		Ja

Läßt man hingegen den Hauptparameter weg, so bedeutet die Anfrage *Haben Peter und Tina ein Kind?*

Aktion	**Beschreibung**	**Wert**
Anfrage	@Beziehung(Kind(;$Peter;$Paula))	
Antwort		Ja

Atlas antwortet allerdings auf diese Frage - wie in jedem Fall - mit einer Fehlermeldung, falls überhaupt keine Informationen hierzu vorhanden sind.

gg) Mehrere offene Parameter

In den bisherigen Beispielen wurde davon ausgegangen, daß bei einer Frage oder Übermittlung lediglich der Hauptparameter in bestimmten Konstellationen offengelassen wird. Hat eine Relation jedoch mehr als einen weiteren Parameter, so mag es interessant sein, auch mehr als einen Parameter bewußt offen zu lassen.

Folgender Sachverhalt soll den weiteren Operationen zugrunde liegen: Peter hat zwei Kinder mit Tina: Anton geb. 11.11.73 und BERTA geb. am 29.4.77. Weiterhin hat Peter ein Kind mit Doris: CLEMENS geb. am 22.8.80. Tina und Doris haben jeweils ein Verfahren gegen Peter angestrengt. Neben Tina ist auch Anton Antragsteller in ihrem Verfahren:

Objekte

Bezeichner	Wertetyp	Werte
Peter	Name	Peter Müller
Paula	Name	Paula Meyer
Anton	Name	Anton Meyer
Berta	Name	Berta Meyer
Clemens	Name	Clemens Schulz
Doris	Name	Doris Lehmann
Geburtstag Anton*	Datum	11.11.1973
Geburtstag Berta*	Datum	29.04.1978
Geburtstag Clemens*	Datum	22.08.1980
Sache Paula./.Peter		
Sache Doris./.Peter		

Beziehungen

Relation	Hauptparameter	Parameter	Existenz
Geburtstag	Geburtstag Anton*	Anton	
	Geburtstag Berta*	Berta	
	Geburtstag Clemens*	Clemens	

Kind	Anton	Peter Paula
	Berta	Peter Paula
	Clemens	Peter Doris
Verfahren	Sache Paula./.Peter	
	Sache Doris./.Peter	
Antragsteller	Paula	Sache Paula./.Peter
	Anton	Sache Paula./.Peter
	Doris	Sache Doris./.Peter
Antragsgegner	Peter	Sache Paula./.Peter
	Peter	Sache Doris./.Peter

Angenommen, es steht die Frage nach allen Kindern von Peter, ungeachtet ihrer jeweiligen Mutter, im Raum, so muß eine Beschreibung für *Kind von Peter* gefunden werden. Für diesen Fall wird in der Relation *kind = Kind*(*vater, mutter*) sowohl der Hauptparameter *kind* als auch der Parameter *mutter* offen gelassen. Die folgende Anfrage ermittelt den Geburtstag eines Kindes von Peter, ungeachtet der Mutter:

Aktion	**Beschreibung**	**Wert**
Anfrage	Geburtstag(;Kind(;$Peter;))	
Antwort		22.08.1980

Wurden die Geburtstage in der Reihenfolge des Datums eingegeben, so liefert die gezeigte Anfrage den Wert *22.08.1980*. Dabei geht das System wiederum rekursiv von innen nach außen vor: Zunächst werden alle Objekte ermittelt, die die offenen Parameter belegen können. Auf diese Weise wird eine Liste aller Kinder von Peter zusammengestellt, völlig unabhängig von dem offenen Parameter (also Anton, BERTA und CLEMENS). Intern übergibt diese Rekursionsebene nun das zuletzt eingetragene Kind *Anton* an die nächst höhere Ebene.

In dieser Rekursionsebene, die die Relation *Geburtstag* abarbeitet, wird aus allen eingetragenen Geburtstagen des ermittelten Kindes der zuletzt eingetragene gewählt und wiederum an die nächst höhere Ebene weitergeleitet. Im vorliegenden Fall erfüllt freilich nur ein Objekt *Geburtstag Anton* die Relation. Die höchste Ebene arbeitet die - hier imaginäre - Relation *Datum* ab. Letztere ermittelt als Ergebnis der Anfrage den zuletzt eingetragenen Datumswert für das Objekt *Geburtstag Anton*. Dieses Vorgehen macht deutlich, an welcher Stelle der vom System zur Verfügung gestellte Parameter *@Pos*() einzusetzen ist, um die Standardauswahl des jeweils jüngsten Wertes zu umgehen. Will man beispielsweise den Geburtstag des ersten Kindes ermitteln, so wird die Relation *Kind* um den Parameter @Pos(1) zu erweitern sein:

Aktion	**Beschreibung**	**Wert**
Anfrage	Geburtstag(;Kind(;$Peter;;@Pos(1)))	
Antwort		11.11.1973

Die Anfrage liefert den Wert *11.11.1973*, also Antons Geburtstag. Der Fall wird etwas komplexer, wenn nun Antons Geburtstag streitig ist. Das folgende Beispiel zeigt zunächst eine mögliche Variante, den alternativen Wert zu übermitteln:

Aktion	**Beschreibung**	**Wert**
Übermitteln	Geburtstag(;Kind(;$Peter;;@Pos(1)))	11.11.74

Der Wert läßt sich einfacher mit dem Ausdruck @Geburtstag(;$Anton) beschreiben. Dem Objekt *Geburtstag Anton* sind nun zwei Datumswerte zugeordnet:

Objekte

Bezeichner	Wertetyp	Werte
...		
Geburtstag Anton*	Datum	11.11.1973 11.11.1974
...		

Eine Anfrage mit dem oben beschriebenen Ausdruck Geburtstag(;Kind(;$Peter;;@Pos(1))) führt zur Rückgabe des zweiten Wertes *11.11.1974*. Aufgrund des rekursiven Abarbeitens des verschachtelten Ausdrucks hat der Parameter @Pos(1) keine Auswirkungen auf die Auswahl des Wertes, sondern lediglich auf das zugrundeliegende Objekt. Will man nun den ersten Wert erfragen, so muß der Positionsparameter wiederholt werden:

Aktion	**Beschreibung**	**Wert**
Anfrage	Geburtstag(;Kind(;$Peter;; @Pos(1)));@Pos(1)	
Antwort		11.11.1973

Auf diese Anfrage hin antwortet Atlas mit dem ersten eingetragenen Wert *11.11.1973*. Zur Klarstellung sei nochmals betont, daß es sich bei dem gewählten Ausdruck um eine Kurzform des Ausdrucks @Datum(Geburtstag(;Kind(;$Peter;;@Pos(1)));@Pos(1)) handelt. Dies Tatsache verdeutlicht die Stellung des Parameters @Pos(1) außerhalb jeder Klammer. Zur Vertiefung sei noch die folgende Variante besprochen:

Aktion	**Beschreibung**	**Wert**
Anfrage	Geburtstag(;Kind(;$Peter;)); @Pos(1)	
Antwort		22.08.1980

Auf diese Anfrage hin liefert Atlas den Wert *22.08.1980*. Es wird also nicht aus einer Liste aller Geburtstage von allen Kindern der erste, und zwar von Peter, ausgewählt. Das ergebe den Wert *11.11.1973*. Vielmehr übergibt der Ausdruck Kind(;$Peter;) das zuletzt eingetragene Kind CLEMENS als Parameter an die Relation *Geburtstag*. Diese übergibt an die Relation *Datum* den hier einzigen Geburtstag von CLEMENS, nämlich *Geburtstag Clemens*. Dessen einziger Datumswert wiederum ist der *22.08.1980*.

hh) Eigenschaften

Eigenschaften besitzen die Form $x = E$ in funktionaler Schreibweise oder $E(x)$ in Relationenschreibweise. Sie besitzen nach der hier verwendeten Terminologie neben dem Hauptparameter keine weiteren Parameter. Im Beispielsfall wird zwei Objekten die Eigenschaft *Verfahren* zugewiesen. Dies mag zunächst befremdlich wirken, da ein Verfahren geeignet ist, eine Beziehung zwischen vielerlei Objekten herzustellen. Es scheint deshalb geboten, beispielsweise die Verfahrensbeteiligten als Parameter der Relation *Verfahren* zu sehen. Ebenso könnte das Gericht als Parameter eingeführt werden. Gerade in der Vielfalt der hier nur angedeuteten Möglichkeiten liegt letztlich auch das praktische Problem: Es gibt keinen sinnvollen Numerus Clausus an Verfahrensbeteiligten. Insbesondere kann es für nahezu jede der Beteiligungsformen mehrere tatsächlich Beteiligte geben[343].

Eine Einführung der Parameter in eine Relation ist letztlich nur dann sinnvoll und geboten, wenn durch die Belegung der Parameter mit Objekten aus mehreren gesuchten Objekten eines identifiziert werden kann oder wenigstens eine effektive Einschränkung vorgenommen werden kann und muß. Weiterhin ist zu berücksichtigen, daß der Parameter $@Pos(n)$ immer die Möglichkeit einer Identifizierung genau

[343] Vgl. S. 125.

eines Objekts bietet. Dabei muß nochmals klargestellt werden, daß das hier entwikkelte System von einer fallbezogenen Ablage der Daten ausgeht. Es ist daher im vorliegenden Beispiel eine eher willkürliche Konstellation, daß überhaupt mehr als ein Verfahren pro Fall instantiiert wurde.

Ergebnis dieser Überlegungen ist, daß in einem Fall regelmäßig von **dem** Verfahren, d.h. von einem Objekt ausgegangen wird, welches die Eigenschaft hat, das im Fall relevante Verfahren zu sein. Die einzelnen Verfahrensbeteiligten werden über Relationen wie *k = Kläger*(*verfahren*) oder *b = Beklagter*(*verfahren*) etc. zu dem Objekt *Verfahren* in Beziehung gesetzt.

Eine der wesentlichen Fragestellungen wird es sein, zu überprüfen, ob ein Objekt eine bestimmte Eigenschaft hat. Wie bei Relationen mit mehreren Parametern wird auch bei Eigenschaften die Funktion *@Beziehung*(*x*) benötigt, um diese Tatsache abzufragen:

Aktion	**Beschreibung**	**Wert**
Anfrage	@Beziehung(Verfahren($Sache Paula./.Peter))	
Antwort		Fehler

Dies entspricht der generellen Syntax. Die Formulierung erscheint jedoch unnötig kompliziert, zumal der Ausdruck Verfahren($Sache Paula./.Peter) in seiner definierten Bedeutung - er ermittelt den Wahrheitswert von *Sache Paula./.Peter* - wenig sinnvoll scheint. Es wäre denkbar, an dieser Stelle eine syntaktische Ausnahme zu gestatten. Sie könnte etwa darin bestehen, daß ein Ausdruck mit einer Relation dann das Bestehen der Relation bzw. das Vorliegen der Eigenschaft erfragt, wenn der Hauptparameter determiniert ist. Das hätte beispielsweise zur Folge, daß die Fragen Kind(;$Peter;$Paula) oder Antragsteller() auf den Existenzwert des freien Objekts (im Beispiel *Existiert*(*Anton*)) abzielen, wohingegen die Ausdrücke Kind($Anton;$Peter;$Paula) bzw. Antragsteller($Anton) per se bereits die Frage nach der Existenz der Beziehung *Kind*(*Anton*, *Peter*, *Paula*) oder des Vorliegens der Eigenschaft *Antragsteller*(*Anton*) beantworten. Nach der dennoch gewählten Implementierung haben beide Varianten dieselbe Funktion. Für die Realisierung der beschriebenen Ausnahme mag sprechen, daß diese zu einer Verkürzung der Ausdrücke führt und wohl auch dem sprachlichen Verständnis nahe kommt. Dagegen spricht jedoch das Bestreben, eine möglichst stringente, eben ausnahmen–arme Syntax zu entwickeln. Dies erleichtert die technische Implementierung und vermeidet letztlich Fehler im Ablauf. Es bleibt jedoch weiterhin möglich, derartige syntaktische Varianten durch einen vorgeschalteten Filter zu implementieren.

ii) Anzahl

Neben den Möglichkeiten, Objekte auf Eigenschaften zu prüfen oder durch Eigenschaften zu identifizieren, mag eine weitere häufig auftauchende Frage sich auf die Anzahl der Objekte beziehen, die eine bestimmte Eigenschaft besitzen. Um die Anzahl der durch eine Relation abgebildeten Objekte zu bestimmen, stellt Atlas eine Funktion

y = @Anzahl(ausdruck)

zur Verfügung. Sie bildet einen *ausdruck* auf die Anzahl der durch ihn beschriebenen Objekte eines Falls ab. Die folgende Transaktion zeigt die Verwendung:

Aktion	**Beschreibung**	**Wert**
Anfrage	@Anzahl(Antragsteller())	
Antwort		3

Atlas gibt also die Zahl *3* zurück. Dies ist die Gesamtsumme aller Antragsteller in allen Verfahren. (Nebenbei sei auf eine syntaktische Großzügigkeit hingewiesen:

Obwohl die Relation *Antragsteller* zwei Parameter besitzt, kann auf das Semikolon verzichtet werden, wenn der zweite Parameter offen gelassen wird.) Mit der folgenden Anfrage erhält man die Anzahl der Verfahren (eines Falls), also hier *2*:

Aktion	Beschreibung	Wert
Anfrage	@Anzahl(Verfahren())	
Antwort		2

Nun mag es von größerem Interesse sein, die Anzahl und später auch die Namen der Beteiligten der einzelnen Verfahren zu erfragen. Zunächst sei ein Beispiel gezeigt, das letztlich nicht zu dem gewünschten Ergebnis führt:

Aktion	Beschreibung	Wert
Anfrage	@Anzahl(Antragsteller(;;@Pos(1)))	
Antwort		1

Hier richtet sich der Positionsparameter auf Objekte, die Antragsteller sind. Da es nur einen ersten Antragsteller gibt, muß die Anfrage als Ergebnis *1* liefern[344]. Statt dessen ist die Anzahl der Verfahrensobjekte zu reduzieren. Dies erfolgt in der Anfrage

Aktion	Beschreibung	Wert
Anfrage	@Anzahl(Antragsteller(;Verfahren()))	
Antwort		1

Die Eigenschaft *Verfahren* liefert an die übergeordnete Relation *Antragsteller* genau ein Objekt, nämlich das letzte, welchem die Eigenschaft zugewiesen wurde. Im Fall ist dies die *Sache Doris./.Peter*. Die Anzahl der Antragsteller dieses Verfahrens ist *1*. Will man die Anzahl der Antragsteller des ersten Verfahrens ermitteln, so erfolgt dies über den Positionsparameter mit dem Ausdruck

Aktion	Beschreibung	Wert
Anfrage	@Anzahl(Antragsteller(;Verfahren(;@Pos(1))))	
Antwort		2

Dieser Ausdruck liefert im Ausgangsfall die Anzahl der Antragsteller des ersten Verfahrens *Sache Tina./.Petra* also den Wert *2*. Soll nun in einem Schriftsatz der Name eines Antragstellers oder Gegners eingesetzt werden, so erfolgt der Zugriff auf einen spezifischen Beteiligten eines spezifischen Verfahrens über einen weiteren Positionsparameter mit beispielsweise folgender Anfrage

Aktion	Beschreibung	Wert
Anfrage	@Name(Antragsteller(;Verfahren(;@Pos(1));@Pos(2)))	
Antwort		Anton Meyer

Geht man jedoch von der Prämisse aus, daß wenigstens bei bestimmten typischen Fallkonstellationen jeweils nur ein Verfahren anhängig ist, so ist der erste Positionsparameter $@Pos(1)$ verzichtbar.

Bisher sind die syntaktischen Festlegungen für die Anfrage und Übermittlung von Fakten bezogen auf Relationen und Werte bekannt. Weiterhin wurden die systemimmanente Funktion $@Pos(x)$, die quasi korrespondierende Funktion $@Anzahl(x)$ sowie die Funktionen für den Zugriff auf bestimmte Werte vorgestellt.

[344] Das Beispielsprogramm liefert hier fälschlicherweise die Zahl „3".

e) Filter

Dieser Abschnitt geht nun besonders auf die Realisierung bestimmter juristischer Besonderheiten bei der Betrachtung von Fakten ein, wie sie in der Konzeption ausführlich dargelegt wurden[345].

aa) Allgemein

Wie bereits festgestellt besteht ein wesentlicher Teil der juristischen Arbeit im Umgang mit unterschiedlichen Aussagen zu derselben Frage. Atlas läßt hierzu die Eingabe beliebiger auch redundanter Informationen zu. Im folgenden wird beschrieben, wie diesen Daten Informationen über die Quelle sowie den Geltungszeitraum bei der Eingabe beigefügt werden können und wie die Daten bei der Ausgabe entsprechend kanalisiert werden können.

Atlas realisiert diese Möglichkeit über eine Einrichtung, die als *Filter* bezeichnet werden soll. Mit Filtern wird dem System global mitgeteilt, welche zusätzlichen Informationen allen einfließenden Daten beizufügen sind. Für die Ausgabe sorgt der Filter dafür, daß nur Informationen berücksichtigt werden, die entsprechende Zusatzinformationen enthalten. Funktional ähneln Filter dem systemeigenen Parameter *@Pos*(n). Ist beispielsweise ein Filter gesetzt, der lediglich Informationen zuläßt, die von einem Verfahrensbeteiligten stammen, so könnte dieser auch als Parameter *@Quelle*(x) realisiert werden. Dieser Parameter wäre dann jeder Relation anzufügen. Die zuletzt formulierte Anfrage nach dem Namen eines Verfahrensbeteiligten wäre dann als @Name(Antragsteller(;Verfahren(;@Pos(n);@Quelle(x));@Pos(m);@Quelle(y));@Quelle(z)) zu formulieren. Dasselbe gälte für einen denkbaren Parameter *@Geltungszeitraum*(x, y). Die Anfrage ist tatsächlich auf diese Weise sehr flexibel zu gestalten, da für jeden Informationsbestandteil eine eigene Quellenangabe erfolgen kann. Der Praxis wird dieser komplexe Ausdruck jedoch nicht gerecht. Zum einen wird er sehr lang und kaum mehr leserlich, zum anderen besteht schwerlich ein Bedürfnis dafür, die Informationsquellen derartig zu differenzieren. Vielmehr wird bei der Abfrage eines Wertes davon auszugehen sein, daß alle zugrundeliegenden Informationen aus derselben Quelle stammen sollen oder die Quelle zumindest gleichgültig ist. Ebenso ist davon auszugehen, daß der Geltungszeitraum aller Informationen einheitlich bzw. für einen Teil zumindest unerheblich ist. Atlas geht deshalb davon aus, daß diese Parameter vor einer Abfrage bzw. einer Übermittlung global eingestellt werden. Hierzu dienen die Ausdrücke *@Eingabefilter* und *@Ausgabefilter*.

bb) Eingabefilter

Der Ausdruck *@Eingabefilter* dient der Übermittlung und Anfrage nach den für die Eingabe verwendeten Parametern. Die Parameter werden als eine Zeichenfolge übergeben, in der die Einzelparameter durch Semikola voneinander getrennt sind. Die Zeichenfolge hat folgende Syntax:

ObjektVon;ObjektBis;ObjektQuelle;WertEingabeZeitpunkt

Sowohl der Gültigkeitszeitraum als auch die Quellenangabe erfolgen durch Verweis auf ein Objekt. Bei den Grenzen des Gültigkeitszeitraumes muß es sich zweckmäßigerweise um Ereignisse handeln, d.h. um Objekte, denen ein Datumswert zugeordnet ist. Der Zeitpunkt der Eingabe des Wertes wird als Datumswert angegeben. Fehlt eine Angabe, so wird die entsprechende Information auch den nachfolgend übermittelten Werten nicht beigefügt. Als Eingabezeitpunkt geht Atlas standardmäßig vom Systemdatum des Computers aus. Das folgende Beispiel setzt den Eingabe-

[345] S. S. 97 ff.

filter so, daß die nachfolgend übermittelten Fakten Angaben sind, die *Peter* am 31.1.95 gemacht hat und die von der *Trauung* bis zur *Scheidung* gelten sollen:

Aktion	Beschreibung	Wert
Übermittlung	@Eingabefilter	$Trauung;$Scheidung;$Peter;31.01.1995

Die folgende Übermittlung beinhaltet in Zusammenhang mit dem gesetzten Filter die Information, daß Peter behauptet, seine Ehe mit Tina habe von der Trauung bis zur Scheidung bestanden:

Aktion	Beschreibung	Wert
Übermittlung	@Beziehung(Ehe(;$Peter;Tina))	Ja

In der nächsten Transaktionsfolge wird diese Behauptung dahingehend präzisiert, daß die Ehe nach der Scheidung nicht mehr besteht.

Aktion	Beschreibung	Wert
Übermittlung	@Eingabefilter	$Scheidung;;$Peter;31.01.1995
Übermittlung	@Beziehung(Ehe(;$Peter;Tina))	Nein
Nun wird Tinas Gegenmeinung dem System mitgeteilt:		
Übermittlung	@Eingabefilter	$Trauung;;$Tina;31.01.1995
Übermittlung	@Beziehung(Ehe(;$Peter;Tina))	Ja
Zuletzt werden die unstreitigen Zeitpunkte der Ereignisse an Altas übermittelt:		
Übermittlung	@Eingabefilter	;;;31.01.1995
Übermittlung	@Datum($Trauung)	1.5.70
Übermittlung	@Datum($Scheidung)	23.11.75

cc) Ausgabefilter

Mit Hilfe des Ausdrucks *@Ausgabefilter* können die Angaben für den Ausgabefilter verändert oder abgefragt werden. Der Ausgabefilter schränkt bei der Anfrage eines anderen Programms die berücksichtigten Fakten ein. Wird der Ausgabefilter beispielsweise so eingestellt, daß als Quelle *Peter* angegeben ist, so werden bei allen Operationen lediglich Fakten berücksichtigt, die bei der Eingabe mit der Quellenangabe *Peter* versehen wurden.

Wie beim Eingabefilter werden auch beim Ausgabefilter die einzelnen Parameter in einer Zeichenkette durch Semikolon getrennt übermittelt. Diese Zeichenkette hat jedoch eine etwas andere Syntax:

```
ObjektPunkt;ObjektQuelle;WertEingabeVon;WertEingabeBis;Prüfung
```

Anstelle des Gültigkeitszeitraumes bei der Übermittlung eines Faktums wird für die Anfrage nach Fakten von einem Gültigkeitszeitpunkt ausgegangen[346]. Will eine Anwendung überprüfen, ob ein Faktum über einen Zeitraum hin gültig ist, so muß sie die in dem Zeitraum relevanten Zeitpunkte der Reihe nach abprüfen. Dies ist vernünftig, da sich die Frage nach der Geltung einer Aussage zu jedem juristisch relevanten Zeitpunkt neu stellt. Der Parameter ObjektQuelle entspricht dem Eingabeparameter.

[346] An dieser weicht die tatsächliche Realisierung erheblich von der nun folgenden Beschreibung ab. In dem Beispielsprogramm sind für den Gültigkeitszeitraum Datumswerte anzugeben. Es bleibt dem anfragenden Programm überlassen, zu klären, welches Ereignis welchen Datumswert hat.

Anstelle eines Eingabezeitpunktes werden bei der Ausgabe quasi spiegelbildlich zum Geltungszeitraum die Grenzen eines Eingabezeitraumes angegeben. Es muß also kein genauer Zeitpunkt festgelegt werden, an dem eine gesuchte Eingabe erfolgte, sondern es kann ein beliebig enger oder weiter Zeitraum bestimmt werden. Wird eine der Grenzen offen gelassen, so wird dieser Wert als unendlich (*seit ewig* bzw. *bis heute*) interpretiert.

Der Parameter `Prüfung` legt das Verhalten von Atlas fest, falls Angaben ohne jede Zusatzinformation auftreten. Er kann folgende Werte annehmen:

Streng	Es werden nur Fakten berücksichtigt, die zu den Angaben des Filters auch Eingabeinformationen haben.
Locker	Es werden alle Fakten berücksichtigt, die jedenfalls keine dem Filter widersprechenden Informationen haben.
Negativ	Es werden nur Fakten berücksichtigt, die keine zusätzlichen Informationen haben.

Die Standardeinstellung ist *locker*. Sie geht juristisch gesehen davon aus, daß Angaben, die keinerlei Quell– und Gültigkeitsinformationen haben, zunächst unbestritten sind und für den gesamten fallrelevanten Zeitraum gelten.

Mit Hilfe des Ausgabefilters soll nun der Frage nachgegangen werden, ob der unstreitig am 11.11.73 geborene Anton während der Ehe geboren wurde oder nicht. Hierzu wird der Geltungszeitpunkt auf Geburtstag(;$Anton) gesetzt. Daraufhin wird gefragt, ob zu diesem Zeitpunkt die Ehe bestand. Die folgende Transaktion ermittelt zunächst die Aussage von Peter:

Aktion	**Beschreibung**	**Wert**
Übermittlung	@Ausgabefilter	@Geburtstag(;$Anton);$Peter;;;
Anfrage	@Beziehung(Ehe(;$Peter;$Tina))	
Antwort		Nein

Eine Anfrage nach der Ansicht von Tina kommt zum gegenteiligen Ergebnis:

Aktion	**Beschreibung**	**Wert**
Übermittlung	@Ausgabefilter	@Geburtstag(;$Anton);$Tina;;;
Anfrage	@Beziehung(Ehe(;$Peter;$Tina))	
Antwort		Ja

dd) Auswertung von Ausdrücken in Filtern

aaa) Problematik

Die Beispiele zeigen, daß innerhalb von Ausdrücken, die Filter festlegen, Referenzen auf Objekte eingesetzt werden. Für die Auswertung, ob eine bestimmte Aussage an einem Zeitpunkt gültig ist, werden jedoch konkrete Datumsangaben benötigt. Die Datumsangaben für die zugrundeliegenden Ereignisse können je nach Quelle divergieren. Hieraus ergibt sich das Problem, ob und welche Filter zu welchem Zeitpunkt bei der Umsetzung einer Objektreferenz auf eine Datumsangabe verwendet werden. Die Situation wird dann sogar noch etwas problematischer, wenn in einem Filterausdruck kein Objektbezeichner, sondern wiederum eine komplexe Beschreibung eines Ausdrucks auftritt. Hier stellt sich die Frage, wann und mit welchem Filter der Ausdruck ausgewertet wird, mit dem der neue Filter eingestellt werden soll. An dieser Stelle differenziert Atlas zwischen Ein- und Ausgabefiltern.

bbb) Eingabefilter

Bei der Interpretation von Ausdrücken des Eingabefilters wird grundsätzlich der voreingestellte Ausgabefilter zugrundegelegt. Dabei erfolgt die Umsetzung einer komplexen Beschreibung eines Objekts zum Zeitpunkt der Übermittlung des Filters. Die Umsetzung des Objekts in einen verwertbaren Datumswert hingegen erfolgt erst

zum Zeitpunkt der Anfrage eines Wertes. Wird beispielsweise folgender Eingabefilter übermittelt

Aktion	Beschreibung	Wert
Übermittlung	@Eingabefilter	$Trauung;$Scheidung;Antragsgegner();31.01.1995

so wird der Ausdruck Antragsgegner() sofort bei der Eingabe gemäß dem aktuellen Ausgabefilter in das entsprechende Objekt umgesetzt. Dies entspricht dem Ergebnis, den ein anderes Programm durch die Transaktion

Aktion	Beschreibung	Wert
Anfrage	@Objekt(Antragsgegner())	
Antwort		$Peter

erhält. Die nachfolgenden übermittelten Fakten erhalten also als Quellenangabe nicht den Ausdruck Antragsgegner(), sondern $Peter. Dasselbe gilt für die Auswertung von Ausdrücken, die den Eingabezeitpunkt betreffen. Wird ein solcher Zeitpunkt etwa mit einem Ausdruck Empfang(;$Schriftsatz2) beschrieben, so ermittelt Atlas das zugehörige Datum sofort und zwar mit der Maßgabe des eingestellten Ausgabefilters.

Die Ermittlung der Datumswerte von den Ereignisobjekten, die den Geltungsbereich einer Aussage abstecken, erfolgt hingegen erst zum Zeitpunkt einer akuten Anfrage. Auch hier wird der eingestellte Ausgabefilter bei der Auswahl aus mehreren möglichen Datumswerten eines Objekts zugrundegelegt.

ccc) Ausgabefilter

Bei der Übermittlung eines Ausgabefilters wird der vorher eingestellte Ausgabefilter zunächst gelöscht. Die Parameter werden dann von rechts nach links ausgewertet. Die Auswertung von Ausdrücken, die den Eingabezeitraum beschreiben, erfolgt also zuerst und gänzlich ungefiltert. Die Auswertung des Ausdrucks, der das Quellobjekt bezeichnet erfolgt unter Berücksichtigung des Eingabezeitraumes. Zuletzt wird der Ausruck ausgewertet, der den Gültigkeitszeitpunkt bezeichnet.

Die Umsetzung des Objekts, welches den gewünschten Gültigkeitszeitpunkt repräsentiert, zu einem mathematisch auswertbaren Datumswert erfolgt im Zeitpunkt einer konkreten Anfrage. Dabei werden dieser Auswertung die übrigen Angaben des Filters, insbesondere also die Quellenangabe zugrunde gelegt. Das Verfahren soll an einem Beispiel verdeutlicht werden. In einem leeren Fall werden folgende Transaktionen durchgeführt:

Aktion	Beschreibung	Wert
Übermittlung	@Beziehung(Kind($Anton;$Peter;$Tina))	Ja
Übermittlung	@Datum($Trauung)	1.5.70
Übermittlung	@Eingabefilter	$Trauung;$Scheidung;;
Übermittlung	@Beziehung(Ehe(;$Peter;$Tina))	Ja
Übermittlung	@Eingabefilter	$Scheidung;;;
Übermittlung	@Beziehung(Ehe(;$Peter;$Tina))	Nein
Übermittlung	@Eingabefilter	;;$Tina;
Übermittlung	@Datum($Scheidung)	01.11.73
Übermittlung	@Datum($Geburtstag(;$Anton))	11.11.73
Übermittlung	@Eingabefilter	;;$Peter
Übermittlung	@Datum($Scheidung)	01.11.72
Übermittlung	@Datum($Geburtstag(;$Anton))	11.11.72

Im Klartext beschreiben diese Eingaben eine Situation, in der unstreitig ist, daß die Ehe zwischen Peter und Tina mit der Trauung begann und mit der Scheidung endete. Weiterhin ist unstreitig, daß Anton das Kind der beiden ist. Die Trauung fand nach einhelliger Ansicht am 1.5.70 statt. Der Zeitpunkt der Scheidung sowie Antons Geburtstag sind zwar streitig, offensichtlich ist jedoch unstreitig, daß Anton nach der Scheidung geboren wurde. Ein erwartungskonform reagierendes System müßte bei einer Anfrage, ob zum Zeitpunkt des Geburtstages von Anton eine Ehe bestand, unabhängig von der abgefragten Meinung zu diesem Ergebnis kommen. Die folgende Transaktion behandelt die Ansicht von Peter. Sie ist insofern unproblematisch, als, unabhängig davon, welches Geburtsdatum letztlich verwendet wird, der Scheidungstermin jedenfalls vor diesem Datum liegt.

Aktion	Beschreibung	Wert
Übermittlung	@Ausgabefilter	Geburtstag(;$Anton);$Peter;;
Anfrage	@Beziehung(Ehe(;$Peter;$Tina))	
Antwort		Nein

Bei Tinas Ansicht ergibt sich ein Problem folgender Art: Würde das System bei der Ermittlung des Wertes von *Geburtstag Peter* keinerlei Filter einsetzen, so würde der zuletzt eingegebene Wert, also der 11.11.1972 als Gültigkeitszeitpunkt zugrunde gelegt werden. Dieser Wert liegt jedoch vor dem von Tina angegebenen Scheidungstermin. Das System würde bei diesem Vorgehen also von einem Bestehen der Ehe zum Zeitpunkt der Geburt ausgehen. Tatsächlich berücksichtigt Atlas bei der Ermittlung des Datumswertes im Ausgabefilter bereits die Quellenangabe desselben Filters. Es legt also als Datumswert für den Geburtstag den 11.11.73 zugrunde. Dieses Vorgehen führt dazu, daß die Transaktion letztlich erwartungskonform verläuft:

Aktion	Beschreibung	Wert
Übermittlung	@Ausgabefilter	Geburtstag(;$Anton);$Tina;;
Anfrage	@Beziehung(Ehe(;$Peter;$Tina))	
Antwort		Nein

Ein erwartungsgemäßes Verhalten erscheint zunächst trivial, ist aber, wie das Beispiel zeigt, mit einem erheblichen Regelungsaufwand verbunden.

Die Unsicherheitsfaktoren der juristischen Praxis werden in Atlas durch Setzen von Filtern gelöst. Eingabefilter fügen die notwendigen Informationen an die neuen Aussagen an. Ausgabefilter ermöglichen die konkrete Selektion bestimmter Aussagen.

f) Weitere Funktionalität

Der vorangegangene Abschnitt hat die wesentlichen Kommunikationselemente zum Austausch von Fakten beschrieben. Er hat dabei neben der Syntax der Faktenbeschreibung bereits einige systeminterne Funktionen vorgestellt. Der folgende Abschnitt bietet einen weiteren systematischen Überblick über die Systemfunktionen.

Die Systemfunktionen können Zugriff auf die fallbezogenen Daten sowie die globale Datenbasis geben. Weiterhin ermöglichen sie eine begrenzte Programmablaufsteuerung.

aa) Fallbezogene Informationen

Die bisher bereits verwendeten Systemfunktionen ermöglichen den Zugriff auf Informationen. Diese können systeminterner Art sein, sie können jedoch auch selbst ein Faktum liefern, welches in einem juristischen Verarbeitungsvorgang eine Rolle spielen mag. Die folgenden Ausführungen fassen diese Funktionen systematisch zusammen.

aaa) Umwandlung von Werten in Beweisobjekte[347]

In Regel 10[348] wurde gefordert, daß jede Aussage durch Beweise belegbar sein muß. Diese Regel ist ein wesentlicher Faktor der juristischen Sonderbehandlung von Fakten. Hier wurde weiterhin festgestellt, daß Beweismittel der rechtlichen und tatsächlichen Würdigung unterliegen, ähnlich wie Sachverhaltsmerkmale. Im bisher beschriebenen System ist eine derartige Abbildung von Beweisantritten allerdings per se nicht möglich, da an einer Beziehung wie etwa *Zeuge*(*Z,A*) nur Objekte und keine Aussagen beteiligt sein können.

Ferner wurde für Beweise gefordert, daß die Liste der Beweismittel nicht geschlossen sein darf. Die technische Lösung kann entsprechend nicht durch die Implementierung von systemimmanenten Beweisrelationen wie *Zeuge, Augenschein* etc. erfolgen. Bei derartigen Funktionen wären zwar syntaktische Ausnahmen der beschriebenen Art leicht zu realisieren, die notwendige Erweiterbarkeit wäre jedoch nicht möglich, da jede Erweiterung mit einem Systemeingriff verbunden wäre. Das Beweismittel kann aus den gleichen Gründen erst recht nicht wie andere Unsicherheitsfaktoren (Zeit, Quelle etc.) einer Aussage systemintern angefügt werden.

Statt dessen bildet in der gewählten Lösung eine Systemfunktion *@Beweisobjekt*(*aussage*) zu diesem Zweck eine Aussage, also die Feststellung eines Wertes oder der Existenz einer Beziehung, in ein Objekt ab. Dieses Objekt wird jedoch nicht konkret instantiiert, sondern als Verweis auf die Aussage dargestellt. Deshalb kann ein derartiger Ausdruck einzig und allein als Gegenstand von Relationen in Erscheinung treten. Folgende exemplarische Übermittlung teilt in dem zuletzt beschriebenen Fall[349] dem System mit, daß ein konkretes Objekt *Trauschein* als Beweis für den von *Peter* behaupteten Wert des Hochzeitsdatums dient.

Aktion	**Beschreibung**	**Wert**
Übermittlung	@Eingabefilter	;;$Peter
Übermittlung	@Ausgabefilter	;$Peter;;
Übermittlung	@Beziehung(@Urkundsbeweis($Trauschein, @Beweisobjekt(@Datum($Scheidung))))	Ja

Es wird also die Aussage übermittelt, daß zwischen dem Objekt *Trauschein* und dem von *Peter* übermittelten Datumswert der *Scheidung* die Beziehung *Urkundsbeweis* besteht. Quelle dieser Beweisbehauptung ist ebenfalls *Peter*. Dabei dient der Ausgabefilter zur Identifizierung von Peters Aussage über den Datumswert. Der Eingabefilter dient dazu, auch als Quelle für den Beweisantritt Peter zu identifizieren. Im Beziehungsreport stellt sich die Beweisaussage wie folgt dar:

Beziehungen

Relation	Hauptparameter	Parameter	Existenz
Urkundsbeweis	Trauschein	Datum(Scheidung,2)	Ja (Peter)

Anstelle eines Objekts wird als zweiter Parameter ein Verweis auf den zweiten Werteeintrag in der Werteliste des Objekts *Scheidung* angegeben. Da die Aussagen über Werte eines Objekts keiner Veränderung unterliegen, ist dieser Verweis dauerhaft eindeutig. Macht Peter eine neue Aussage über diesen Wert, so wird er an die Liste angehängt. Es muß demgemäß auch ein neuer Beweisantritt erfolgen.

---

[347] Die Funktion dieses Abschnitts ist nicht im Beispielprogramm implementiert.

[348] S. S. 100.

[349] S. S. 154.

Das Verfahren erlaubt es nun im Prinzip, Aussagen zum Gegenstand einer beliebigen Relation zu machen. Sinnvoll erscheint diese Möglichkeit jedoch nur bei Relationen, die Beweismittel beschreiben. Deshalb wurde auch die Bezeichnung *Beweisobjekt* gewählt. Zu überlegen wäre allenfalls, ob diese Option auch überall dort Anwendung finden kann, wo Aussagen Gegenstand rechtlicher Beurteilung sind. Insbesondere ist dies bei strafrechtlichen Tatbeständen wie Falschaussage oder Verleumdung denkbar. Eine Untersuchung dieser Option ist jedoch im Rahmen dieser Arbeit nicht zu leisten.

Die gewählte Lösung bietet jedenfalls ein ausreichendes Maß an Flexibilität, da sie mithin alle rechtlichen und faktischen Beurteilungsmöglichkeiten auf die Beweismittel überträgt. Es können beliebige Relationen gebildet werden, die Beweismittel verkörpern. Die Relationen können bestimmten Regeln unterworfen werden. Beweisantritte können mit denselben individuellen Faktoren über Quelle, Geltung und Zeitpunkt belegt werden, wie jede andere Aussage auch. Letztlich ist auch ein rekursiver Beweisantritt über einen Beweisantritt denkbar, wenn auch wenig sinnvoll.

bbb) Positionsparameter

Falls für ein durch einen Ausdruck beschriebenes Faktum mehrere Angaben existieren, so generiert Atlas eine virtuelle Liste dieser Angaben. In diese Liste werden nur Angaben eingestellt, die zu dem aktuellen Ausgabefilter passen. Falls keine weitere Spezifizierung erfolgt, verwendet Atlas bei den weiteren Operationen den letzten Eintrag dieser Liste. Dies ist regelmäßig auch die physikalisch zuletzt übermittelte Angabe.

Der Systemparameter *@Pos*(n) ist auf die meisten Relationen und internen Funktionen anwendbar. Er wird an die Parameter eines Ausdrucks wie ein zusätzlicher Parameter mit dem Standardtrenner (*;*) angefügt. Er erlaubt es, auf eine andere als die letzte Eingabe der virtuellen Liste zuzugreifen. Dabei gibt *n* die Position der Angabe in der durch die Beschreibung generierten Liste an. Für $n > 0$ wird vom Beginn der Liste an gezählt. Für $n < 0$ wird von der letzten Angabe zurückgezählt. Der Ausdruck *@Pos*(-2) beispielsweise bezeichnet die vorletzte Eintragung in der Liste. Ist der Betrag von *n* größer als die Anzahl der in der Liste befindlichen Einträge, so wird die Anfrage negativ bescheidet. Der Positionsparameter ermöglicht letztlich die Kennzeichnung eines jeden Objekts mit Hilfe seiner Eigenschaften auch dann, wenn mehrere Objekte diese Eigenschaft haben[350].

Der Positionsparameter wird bei der Übermittlung von Werten immer ignoriert. Dies ist deshalb sinnvoll, weil das System keine nachträgliche Änderung von Werten zulassen soll. Eine solche Änderung muß immer durch Anfügen eines neuen Wertes erfolgen.

Die Funktion *@Anzahl*$(Ausdruck)$ liefert die Anzahl der Einträge einer imaginären Liste, die anhand des übergebenen Ausdrucks gebildet wurde. Je nachdem, ob Ausdruck einen Wert, eine Beziehung oder ein Objekt beschreibt, wird die Anzahl der auf die Beschreibung passenden Werte, Beziehungen oder Objekte zurückgegeben. Dabei wird der aktuelle Ausgabefilter berücksichtigt. Die Kombination der Ausdrücke *@Pos*(n) und *@Anzahl*$(Ausdruck)$ erlauben es Programmer, beliebige Vorgänge über mehrere Objekte oder Werte durchzuführen.

ccc) Objekte

Atlas bietet mehrere Funktionen, um auf Informationen über Objekte zugreifen zu können: Die Systemeigenschaft *@Objekt*(x) ist inhaltslos. Jedes Objekt hat diese

[350] Vgl. S. 93 sowie zur Kennzeichnung s. S. 44.

Eigenschaft mit seiner Instantiierung. Sie kann nicht mit Hilfe der Relation *@Beziehung* einem Objekt mehrfach zugewiesen werden. *@Objekt*(x) besitzt im Gegensatz zu anderen Relationen keinen Standardwertetyp. Auf eine Anfrage hin übermittelt die Eigenschaft vielmehr den Bezeichner des beschriebenen Objekts[351]. Der zurückgegebene Ausdruck kann anstelle komplexer Beschreibungen eines Objekts eingesetzt werden. Eine Anwendung kann sich den Bezeichner geben lassen, wenn sie oftmals auf ein Objekt zugreifen will. Der Zugriff über den Bezeichner vermeidet die mehrfache Suche nach dem Objekt in Atlas und mag deshalb das Laufzeitverhalten einer Anwendung verbessern.

Die Eigenschaft *@Objekt*(x) bietet in Verbindung mit dem Positionsparameter und der Funktion *@Anzahl*(n) einen Zugang zu allen Objekten. So liefert die folgende Transaktion die Anzahl der Kinder von Peter:

Aktion	**Beschreibung**	**Wert**
Anfrage	@Anzahl(@Objekt(Kind(;$Peter;)))	
Antwort		3

Eine Anwendung kann sich diese Eigenschaft auch in einem dynamischen Prozeß zunutze machen, indem sie beispielsweise die einzelnen Objekte abarbeitet. So kann sie etwa die Stammdaten der Kinder erfassen oder ihre Unterhaltsansprüche aufsummieren etc. In Verbindung mit der Systemeigenschaft *@Objekt*(x) liefert die Funktion die Anzahl aller in einem Fall verfügbaren Objekte. Das folgende Beispiel zeigt, wie dieses Zusammenspiel für den Zugang zu allen Objekten genutzt werden kann:

Aktion	**Beschreibung**	**Wert**
Anfrage	@Anzahl(@Objekt())	
Antwort		x
Anfrage	@Objekt(;@Pos(1))	
Antwort		Peter
Anfrage	@Objekt(;@Pos(2))	
Antwort		Tina
. . .		
Anfrage	@Objekt(;@Pos(x))	
Antwort		Bezeichner

Zunächst wird also die Anzahl aller Objekte abgefragt. Danach wird eine Schleife so oft durchlaufen, bis ein beliebiger Prozeß auf alle Objekte angewendet wurde. Die Funktion *@Index*$(objekt)$ liefert die Position eines Objekts in der Liste aller Objekte eines Falls. Auf diese Indexposition kann über die alternative Bezeichnung *#Index*[352] zugegriffen werden. Beispiel:

Aktion	**Beschreibung**	**Wert**
Anfrage	@Index($Berta)	
Antwort		5
Anfrage	@Objekt($#5)	
Antwort		Berta

Der Zugriff über den Index eines Objekts anstelle des Bezeichners erhöht - unabhängig von der verwendeten Zugriffstechnik des Datenverwaltungsmoduls -

[351] Der Prototyp übermittelt auf eine Anfrage hin einen Fehler. Hier kann *@Objekt*(x) nur als Parameter einer anderen Funktion auftauchen wie etwa *@Zahl*$(@Objekt(@Pos(1)))$.

[352] S. S. 134 ff.

jedenfalls die Ablaufgeschwindigkeit, da der Objektbezeichner nicht erst gesucht werden muß. Weiterhin ermöglicht der Zugang über den Index ebenfalls eine iterative Behandlung aller Objekte.

ddd) Vervollständigen von Ausdrücken

Die Funktion *@AlleParameter*(*ausdruck*) liefert den gegebenen *ausdruck* mit allen Parametern, insbesondere denen, die durch Unterausdrücke beschrieben oder ganz ausgelassen wurden. Ein Programm kann so schnell alle nicht bekannten Objekte einer Beziehung in Erfahrung bringen. Das folgende Beispiel zeigt, wie die Eltern eines Kindes aus der Relation *Kind*(*k, v, m*) auf kurzem Wege zu ermitteln sind.

Aktion	Beschreibung	Wert
Anfrage	@AlleParameter(Kind($Anton;;))	
Antwort		Anton;Peter;Tina

Diese Anfrage entspricht nicht dem juristisch korrekten Vorgehen. Vielmehr sollten die einzelnen Eltern über die Relationen *Vater*(*v, k*) und *Mutter*(*v, k*) erfragt werden. Die Funktion dient entsprechend auch nicht der Anfrage nach den Objekten, die über korrelierende Beziehungen zu ermitteln sind. Vielmehr ermöglicht sie das sequentielle Durchlaufen aller in einem Fall auftretenden Beziehungen. Eine Anwendung, die diese Option nutzen will, verwendet hierzu zum Beispiel die Funktion *@Relationsbezeichner*(*#x*), um Zugang zu den einzelnen Relationen zu erhalten. Des weiteren wird mit der Funktion *@AnzahlParameter*(*x*) ermittelt, wie viele Parameter die Relation hat. Dadurch kann die Anwendung über den Parameter *@Pos*(*y*) nun alle Beziehungen, die über eine Relation definiert werden, vollständig abfragen. Die folgenden Transaktionen zeigen einen Ausschnitt einer solchen Sequenz:

Aktion	Beschreibung	Wert
Anfrage	@Relationsbezeichner(#3)	
Antwort		Ehe
Anfrage	@AnzahlParameter(Ehe)	
Antwort		2
Anfrage	@AlleParameter(Ehe(;;;@Pos(1))	
Antwort		Ehe Peter-Tina;Peter;Tina

eee) Werte und Beziehungen

Die Funktionen *@Name*(*x*), *@Zahl*(*x*), *@Datum*(*x*) und *@Existiert*(*x*) geben den entsprechenden Wert des benannten Objekts zurück[353]. Besitzt ein Objekt mehrere Werte, so besteht die Möglichkeit durch den Positionsparameter gezielt auf einen bestimmten Wert zuzugreifen. Die folgende Transaktion greift auf den vorletzten eingegebenen Wert von Peters Geburtstag zu:

Aktion	Beschreibung	Wert
Anfrage	@Datum(Geburtstag(;$Peter);@Pos(-2))	
Antwort		11.11.1973

Die Funktionen können auch zur Übermittlung von Werten eingesetzt werden:

Aktion	Beschreibung	Wert
Übermitteln	@Datum(Geburtstag(;$Peter))	11.11.74

Wird ein Wert durch eine Relation beschrieben, deren Standardwertetyp mit dem gesuchten Wertetyp übereinstimmt, so ist die Funktionsangabe verzichtbar.

[353] Vgl. S. 94, 96 f.

Die Funktion *@Beziehung*(*ausdruck*) liefert den Existenzwert einer Beziehung. Sie kann auch zur Übermittlung eines entsprechenden Wertes eingesetzt werden. Das Zusammenspiel von Existenzwert eines Objekts und einer Beziehung sei anhand folgenden Beispiels dargestellt:

Aktion	Beschreibung	Wert
Übermittlung	@Beziehung(Kind($Anton;$Peter;$Tina))	Ja
Übermittlung	@Datum($Trauung)	1.5.70
Übermittlung	@Eingabefilter	$Trauung;$Scheidung;;
Übermittlung	@Beziehung(Ehe($Trauma;$Peter;$Tina))	Ja
Übermittlung	@Eingabefilter	$Scheidung;;;
Übermittlung	@Beziehung(Ehe(;$Peter;$Tina))	Nein
Übermittlung	@Eingabefilter	$Trauung;;$Peter;
Übermittlung	@Existiert($Trauma)	Nein

Diese Transaktion läßt sich so interpretieren, daß Peter die Existenz der Ehe mit dem Bezeichner *Trauma* anzweifelt. Sie kann womöglich aufgrund einer Annullierung rückwirkend entfallen. Er zweifelt hingegen nicht an, daß wenn diese Ehe besteht, sie auch zwischen Peter und Tina besteht.

Die Funktion *@Info*(*ausdruck*) liefert den Eingabefilter, der bei der Übermittlung des durch *ausdruck* beschriebenen Wertes oder der Beziehung eingestellt war. Als zusätzlicher Parameter wird der Name der Anwendung angefügt, die die Information übermittelt hat[354]. Auf diese Weise kann nicht nur die Quelle der Behauptungen, sondern insbesondere bei algorithmisch gewonnenen Fakten die Quelle des Algorithmus von jeder anderen Anwendung geprüft werden. Die folgende Transaktion versucht zu ermitteln, wer zuletzt etwas über die Ehe *Trauma* gesagt hat:

Aktion	Beschreibung	Wert
Anfrage	@Info(@Existiert($Trauma))	
Antwort		Trauung;;Peter;;Aufnahmebogen

In Verbindung mit dem Positionsparameter kann eine Anwendung somit alle Angaben zu einer Fragestellung in einer Schleife abarbeiten und bestimmten Quellen oder Geltungszeitpunkten zuordnen.

Die Möglichkeit, einer Aussage ein Beweismittel zuzuordnen, realisiert Atlas, indem eine Aussage zum Objekt einer beliebigen Beziehung gemacht werden kann. Atlas bietet neben den einfachen Zugängen zu Fakten auch ein erhebliches Instrumentarium an Funktionen, die es speziell entwickelten Programmen ermöglichen, auf die Fakten eines Falls zuzugreifen. Derartige Programme haben insbesondere die Möglichkeit, unbekannte Fallkonstellationen durch sequentielle Analyse der Fakten auszuwerten.

bb) Globale Informationen

Die vorangegangenen Funktionen betrafen fallbezogene Daten, d.h. Fakten sowie Objekte als Träger von Fakten. Atlas ermöglicht aber auch über die DDE-Schnittstelle einen lesenden Zugriff auf Informationen zu Relationen der globalen Datenbasis.

Die meisten Anwendungen gehen zwar von der Existenz bestimmter Relationen in der globalen Datenbasis aus. Sie installieren diese Relationen in der Regel mit Hilfe ihres Installationsprogrammes. Es sind aber auch Anwendungen denkbar, die flexibel auf die vorhandenen Relationen reagieren. So ermöglicht es das später beschriebene Fallskizzenprogramm (s.S. 190), graphisch beliebige Beziehungen zwischen Objekten herzustellen. Dabei legt es die installierten Relationen der individuellen

[354] Diese Funktionalität ist im Prototyp nicht realisiert.

Datenbasis zugrunde. Lediglich für die Verwaltung der objektbezogenen Stammdaten werden bestimmte Relationen vorausgesetzt. Die hierfür benötigten Informationsmechanismen werden im folgenden beschrieben:

Die Funktion *@AnzahlRelationen*( ) übergibt die Anzahl der in der globalen Datenbasis verfügbaren Relationen. Die Funktion *@Relationsindex*(*bezeichner*) übergibt den Index der durch *bezeichner* identifizierten Relation. Die Funktion *@Relationsbezeichner*(*bezeichner*) übergibt den Bezeichner einer Relation. Sie ist dann sinnvoll, wenn anstelle des Bezeichners ein Ausdruck der Form *#Index* verwendet wird.

Die Funktion *@AnzahlParameter*(*bezeichner*) übergibt die Anzahl der Parameter einer mit *bezeichner* identifizierten Relation abzüglich des Hauptparameters. Die Funktion *@ParameterBezeichner*(*bezeichner;index*) übergibt den Bezeichner der zu einem Parameter korrespondierenden Relation. *index* bezeichnet dabei die Position des Parameters in der Parameterleiste der durch *bezeichner* identifizierten Relation.

Die Funktionen sollen am folgenden Beispiel verdeutlicht werden. Hier wird zunächst der Bezeichner der dritten in der Datenbasis vorliegenden Relation abgefragt. Daraufhin werden die Bezeichner der Parameter ermittelt:

Aktion	Beschreibung	Wert
Anfrage	@RelationsBezeichner(#3)	
Antwort		Ehe
Anfrage	@AnzahlParameter(#3)	
Antwort		2
Anfrage	@ParameterBezeichner(#3;1)	
Antwort		Ehemann
Anfrage	@ParameterBezeichner(#3;2)	
		Ehefrau

Ein ausführlicheres Beispiel demonstriert die weiter unten beschriebene Anwendung *Fallskizze*[355].

cc) Falldatenablage

Neben dem Zugriff auf Informationen der fallbezogenen und der globalen Daten kann eine andere Anwendung Atlas bedingt fernsteuern, um eine Sicherung der Daten zu ermöglichen.

Die Funktionalität beruht auf der Annahme, daß fallbezogene Daten zunächst im Arbeitsspeicher zwischengespeichert und erst auf Anweisung auf dem Massenspeicher abgelegt werden. Dennoch muß eine Anwendung jederzeit damit rechnen, daß Daten auch ohne ihre Anweisung dauerhaft gespeichert werden. Einmal kann jede andere Anwendung eine Speicherung auslösen. Weiterhin kann der Benutzer in der Atlas-Oberfläche eine Speicherung beauftragen. Atlas ermahnt den Benutzer hierzu, falls er die Serveranwendung selbst schließen möchte. Letztlich ist es denkbar, daß ein Datenverwaltungsmodul so organisiert ist, daß es bei jeder Transaktion die Daten sofort auf dem Massenspeicher ablegt.

Die Terminologie der folgenden Funktionen geht davon aus, daß fallbezogene Daten in einem mit *Akte* bezeichneten individuellen Datenbereich abgelegt werden. Es wird weiter vorausgesetzt, daß es jeweils eine Datei auf dem Massenspeicher gibt, die den Namen der gewählten Akte repräsentiert. Die Datei kann je nach Datenver-

[355] S. S. 190.

waltungsmodul die fallbezogenen Daten tatsächlich enthalten oder lediglich als Verweis auf einen Datenbereich in einer Datenbank fungieren.

Die einzige für die Verwaltung der Akten vorgesehene Funktion *@Akte*( ) gibt auf Anfrage den Namen der aktuell bearbeiteten Akte zurück. Da der Name der aktuellen Akte global Geltung hat, muß eine Anwendung damit rechnen, daß eine andere Anwendung diesen Namen ändert. Jedes Programm sollte deshalb vor der Übermittlung von Daten den Namen der aktuellen Akte prüfen und gegebenenfalls ändern. Die Funktion ermöglicht es auch, den Namen einer Akte zu übermitteln. Der Name muß den Anforderungen des Betriebssystems an Dateinamen gerecht werden. Alle nachfolgenden Operationen aller Anwendungen beziehen sich dann auf die neu eingestellte Akte. Die Übermittlung veranlaßt Atlas zudem dazu, die Daten in der vorangegangenen Akte sowie in der neu angegebenen Akte auf dem Massenspeicher dauerhaft abzulegen. Wird ein bislang noch nicht verfügbarer Aktenname angegeben, so wird die Akte von Atlas angelegt. Jede Anwendung ist entsprechend selbst dafür verantwortlich, zu überprüfen, ob die benannte Datei auf dem Massenspeicher des Computers existiert und gegebenenfalls den Benutzer darauf hinzuweisen, daß eine neue Akte angelegt wird. Unter echten **Multitaskingbetriebssystemen** ist selbst dann mit einer erneuten Änderung des eingestellten Namens zu rechnen, wenn die laufende Anwendung die Kontrolle noch nicht an das Betriebssystem zurückgegeben hat. Bei längeren Prozessen muß eine Anwendung deshalb dafür Sorge tragen, daß jede Transaktion sich auf die Akte bezieht, der auch die vorangegangene Transaktion galt. Spätestens sobald der Benutzer eine Anwendung aktiviert, um in ihr zu arbeiten, sollte die Anwendung jedenfalls überprüfen, ob die bearbeiteten Daten noch mit der aktuellen Akte in Einklang stehen.

Die folgende Transaktion legt zunächst in der aktuellen Akte *94000201.AKT* den Namen von Peter ab. Danach legt sie in der Akte *94010034.AKT* den Antragsteller und den Antragsgegner fest.

Aktion	Beschreibung	Wert
Anfrage	@Akte	
Antwort		94000201
Übermitteln	@Name($Peter)	Peter Müller
Übermitteln	@Akte	94010034
Übermitteln	@Beziehung(Verfahren($MyMy))	Ja
Übermitteln	@Beziehung(Antragsteller(;$Doris))	Ja
Übermitteln	@Beziehung(Antragsgegner(;$Peter))	Ja
Übermitteln	@Akte	94010034

Die letzte Transaktion erzwingt eine Ablage der Daten aus der Akte *94010034.AKT* auf dem Massenspeicher. Hiermit sollen die Daten dauerhaft gesichert werden. In einem idealen System, das frei von System- und Anwenderfehlern operiert, ist eine solche Aktion nicht zwingend geboten, da spätestens beim nächsten Aktenwechsel die Daten gesichert werden. Bis dahin kann der Anwender den Computer bereits - wenn auch nur versehentlich - ausgeschaltet haben.

Die vorangegangenen Ausführungen haben gezeigt, daß Atlas jeder Anwendung neben einem Zugriff auf die Fakten eines Falls auch einen DDE-Zugang zu der globalen Datenbasis eröffnet. Weiterhin kann selbstverständlich die aktuell bearbeitete Akte von jeder Anwendung ausgewählt werden.

3. Schnittstelle für Datenverwaltungsmodule

Die bisher besprochene DDE-Schnittstelle bietet den abstraktesten Zugang für eine Anwendung zu den unterschiedlichen Daten im System. Zur physikalischen Ablage und Verwaltung der Daten bedient sich Atlas einer Anzahl weniger abstrakt gestalteter

Funktionen. Diese Funktionen werden aus einem Modul hinzugezogen, das erst beim Start des Programms auf dem Computer des Benutzers eingebunden wird. Die Funktionen sind ausreichend abstrakt, um unterschiedliche Formen physikalischer Datenablage zuzulassen. Sie sind auf der anderen Seite so konkret, daß von dem Datenverwaltungsmodul keinerlei komplexe Entscheidungen verlangt werden.

Das Datenverwaltungsmodul bildet also den Werkzeugkasten, aus dem sich Atlas für die Erfüllung seiner komplexen Aufgaben bedient. Umgekehrt stellt Altas gewissermaßen einen Übersetzer dar, der die juristisch abstrakten Anfragen nach Fakten in eine der physikalischen Datenablage näher kommende Funktionalität transponiert. Die statischen, oft stark verschachtelten Ausdrücke werden in eine Kette von einfachen, flachen Datenzugriffen übersetzt, an deren Ende die Eingabe eines neuen oder die Ausgabe eines vorhandenen Faktums steht.

Die Funktionen erlauben regelmäßig das Lesen einer vorhandenen Information, deren Änderung sowie das Anfügen neuer Information. Im Gegensatz zur DDE-Schnittstelle muß also das Datenverwaltungsmodul regelmäßig auch die Manipulation vorhandener Daten zulassen.

a) Einbinden einer Datenbankmaschine

Das Datenverwaltungsmodul ist als dynamisch zur Laufzeit des Hauptprogramms eingebundene Funktionsbibliothek realisiert. Sie hat den Einheitsnamen *ATLASDV.DLL*. Zur Änderung des Datenverwaltungsmoduls muß eine Datei mit dem definierten Funktionsumfang anstelle der bis dahin benutzten Datei gesetzt werden. Da Atlas über die tatsächliche physikalische Datenablage keinerlei Angaben macht, kann nicht davon ausgegangen werden, daß beim Austausch des Datenverwaltungsmoduls auf alte Daten noch zugegriffen werden kann.

Die folgenden Punkte beschreiben die einzelnen Funktionen, die das Datenverwaltungsmodul anbieten muß. Dabei wird auf die bereits beschriebenen Datenstrukturen[356] zugegriffen. Wie im Fall der DDE-Schnittstelle und der Beschreibung der Datenstrukturen sind auch die folgenden Ausführungen als **schematische** Spezifikation zu sehen. Insbesondere sind die Funktionsnamen eingedeutscht, die Parameter vereinfacht sowie die Syntax auf BASIC übersetzt. Die konkrete Realisierung im Beispielsprogramm kann dem im Anhang abgedruckten Quellcode entnommen werden.

b) Allgemeine Funktionen

Je nach Implementation wird es für einige Module notwendig sein, vom Anfang und Ende der Benutzung informiert zu werden. Weiterhin erhält das Modul die Gelegenheit, sich im Systemreport vorzustellen.

```
Function DBAnmeldung() As Bool
```

Atlas ruft diese Funktion auf, sobald es gestartet wird, noch bevor es andere Funktionen des Moduls aufruft. Dem Modul wird damit Gelegenheit gegeben, die notwendigen Vorkehrungen für den Zugriff zu treffen[357].

```
Function DBAbmeldung() As Bool
```

Atlas ruft diese Funktion auf, kurz bevor das Programm beendet wird. Es ermöglicht dem Modul, eine letzte Datensicherung vorzunehmen und die beanspruchten Systemressourcen freizugeben.

---

[356] S. S. 116.

[357] Funktionen vom Typ *Boole* geben regelmäßig den Wert Null zurück, wenn sie fehlgeschlagen sind. Wurden sie erfolgreich ausgeführt, geben sie einen Wert ungleich Null zurück.

```
Function InfoText(_
    ByVal szText as String, _
    ByVal iMaxTextGr as Integer_
  ) As Integer
```

Die Funktion wird aufgerufen, bevor Atlas dem Benutzer den Systemstatusreport anzeigt. Das Modul kopiert eine Zeichenkette mit Informationen wie Entwickler, Urheberrechte etc. in den durch szText angegebenen Puffer. Die Zeichenkette darf die durch iMaxTextGr angegebene Größe nicht überschreiten. Der Text wird im Systemreport im Abschnitt *Datenbasis* unter dem Punkt *Modul* angezeigt. Er wird im Informationsfenster der Hauptanwendung wiederholt. Das Modul gibt die Anzahl der in den Puffer kopierten Zeichen zurück.

c) Operationen mit der globalen Datenbank

aa) Allgemeine Operationen

Atlas macht dem Modul mit den folgenden Funktionsaufrufen explizit eine Mitteilung, bevor es auf die Datenbasis zugreift und wenn es eine physikalische Sicherung für gegeben erachtet:

```
Function DBSpeichern() As Bool
```

Die Funktion weist das Modul an, die globale Datenbasis abzuspeichern. Das Modul reagiert hierauf nur dann, wenn sich Daten im Arbeitsspeicher befinden, die noch nicht auf dem Massenspeicher gesichert wurden.

```
Function DBÖffnen() As Bool
```

Die Funktion weist das Modul an, die globale Datenbasis für den Zugriff vorzubereiten. Das Modul reagiert hierauf nur dann, wenn es für den Zugriff Daten in den Arbeitsspeicher laden muß.

bb) Zugriff auf Relationen

Die folgenden Funktionen erlauben einen differenzierten Zugriff auf die Relationen in der globalen Datenbasis.

```
Function GRelation(_
    ByVal iX As Long, _
    Relation As tRelation _
  ) As Bool
```

Die Funktion GRelation (*G* steht für *gib*) erfragt Daten einer Relation. iX bezeichnet die Position der Relation in der Relationsliste, beginnend bei 1. Das Modul füllt die Informationen in den mit Relation bezeichneten Puffer.

```
Function SRelation( _
    ByVal iX As Long, _
    Relation As tRelation _
  ) As Bool
```

Die Funktion SRelation (*S* steht für *setze*) verändert Daten einer Relation. iX bezeichnet die Position der Relation in der Relationsliste. Die Informationen befinden sich in dem mit Relation bezeichneten Puffer.

```
Function ARelation(Relation As tRelation) As Long
```

Die Funktion ARelation (*A* steht für *anfügen*) fügt eine neue Relation an die globale Datenbasis an. Die Informationen zu dieser Relation befinden sich in dem mit Relation bezeichneten Puffer. Die Funktion gibt den Index der neuen Relation in der Datenbasis zurück. Dieser Index entspricht der neuen Anzahl der Relationen in der Datenbasis. Der Rückgabewert ist Null, falls die Funktion erfolglos war

```
Function ZRelationen() As Long
```

Die Funktion ZRelationen (*Z* steht für *Zahl*) gibt die Anzahl der Relationen zurück, die in der globalen Datenbasis vorhanden sind.

```
Function FRelation(_
    ByVal sSuche As String, _
    ByVal iPos as Long, _
    ByVal iFlags as Long _
  ) As Long
```

Die Funktion FRelation (*F* für *finden*) sucht eine Relation in der globalen Datenbasis und gibt ihren Index zurück. sSuche enthält den Bezeichner oder eine Zeichenkette, nach der gesucht werden soll. iFlags gibt an, wie nach der Zeichenkette gesucht werden soll. Als Standard wird davon ausgegangen, daß sSuche mit dem Bezeichner der Funktion übereinstimmen soll. Die Funktion ermöglicht es aber auch, nach ähnlichen Relationen zu suchen. Der Rückgabewert bezeichnet den Index der Relation, der in einer Liste der qualifizierten Relationen an der durch iPos definierten Position steht. Er ist Null, falls keine passende Relation gefunden wurde bzw. die Anzahl der qualifizierten Relationen kleiner als iPos ist.

cc) Zugriff auf die Parameter von Relationen

Der Zugriff auf die Parameter einer Relation erfolgt über die Angabe des Index der entsprechenden Relation und der Position des Parameters in der Parameterleiste. Die folgenden ermöglichen Lesen, Ändern, Anfügen und Löschen von Parametern:

```
Function GRelParam(_ '                                  Ermitteln eines Parameters
    ByVal iX As integer, _
    ByVal ixRel As Long, _
  ) As Long
Function SRelParam(_ '                                  Ändern eines Parameters
    ByVal iX As integer, _
    ByVal ixRel As Long, _
    ByVal Param As Long, _
  ) As Bool
Function ARelParam(_ '                                  Anfügen eines Parameters
    ByVal ixRel As Long, _
    ByVal Param As Long _
  ) As Integer
Function LRelParam(\ 'Löschen eines Parameters
    ByVal iX As integer, _
    ByVal ixRel As Long _
  ) As Bool
```

iX enthält die Position des Parameters in der Parameterleiste der mit ixRel identifizierten Relation. Param enthält den Index der Relation, die den Parameter repräsentiert. Der Rückgabewert ist Null im Fall eines Mißerfolgs der Funktion, ansonsten ist er ungleich Null. Bei der Funktion GRelParam enthält er den Index der Relation, die mit dem angegebenen Parameter korreliert. Im Fall der Funktion ARelParam bezeichnet er die Position des neuen Parameters in der Parameterleiste. Weiterhin steht folgende Funktion zur Verfügung:

```
Function ZRelParam(ByVal ixRel As Long) As Integer
```

Die Funktion gibt die Anzahl der Parameter einer Relation zurück. ixRel enthält den Index der Relation, deren Parameterzahl gesucht wird.

d) Operationen mit der fallbezogenen Datenbank

Neben der Behandlung der globalen Datenbasis, d.h. der juristisch relevanten Relationen, handhabt das Datenbankmodul als wesentliche Aufgabe die Fakten eines individuellen Falls.

aa) Akten

Zunächst einmal besitzt das Modul Funktionen zum Öffnen, Schließen und Anlegen von Akten. Dabei enthält eine Akte die Fakten eines spezifischen Falls. In der DDE-Schnittstelle wird diese Funktionalität in einer Funktion zusammengeführt.

```
Function NAkte(ByVal sAkte as String) As Integer
```

Die Funktion legt eine neue Akte an. sAkte gibt den Namen der Akte an. Der Parameter muß die Akte eindeutig identifizieren[358]. Falls die Akte bereits existiert, werden alle Daten der existierenden Akte zunächst gelöscht. Der Rückgabewert der Funktion bestätigt die erfolgreiche Anlage der neuen Akte.

```
Function SAkte(ByVal sAkte as String) As Integer
```

Die Funktion SAkte (*S* steht ausnahmsweise für *speichern*) speichert die aktuellen Fakten in der angegebenen Akte ab. sAkte identifiziert die Akte. Der Rückgabewert bestätigt den Erfolg der Funktion.

```
Function OAkte(ByVal sAkte as String) As Integer
```

Die Funktion sorgt dafür, daß die Fakten bei kommenden Anfragen aus der angegebenen Akte entnommen werden.

bb) Objekte

Auf die an dem Sachverhalt beteiligten Objekte erlaubt das Datenverwaltungsmodul mit folgenden Funktionen den Zugriff:

```
Function GObjekt( _
    ByVal iX As Long, _
    Objekt As tObjekt _
  ) As Bool
```

Die Funktion füllt die Struktur Objekt mit den Informationen des durch iX bestimmten Objekts. iX ist der Index des Objekts. Er wird für jeden Fall beginnend bei *1* hochgezählt. Der Rückgabewert bestimmt den Erfolg der Aktion.

```
Function SObjekt( _
    ByVal iX As Long, _
    ByVal Objekt As tObjekt _
  ) As Bool
```

Die Funktion setzt die Daten des mit iX bestimmten Objekts auf die in Objekt angegebenen Informationen. Der Rückgabewert bestimmt den Erfolg der Aktion.

```
Function AObjekt( _
    Objekt As tObjekt _
  ) As Long
```

Die Funktion fügt an die Objektliste ein neues Objekt mit den in Objekt abgelegten Daten an. Der Rückgabewert bezeichnet den Index des neuen Objekts. Das entspricht der Anzahl der im konkreten Fall nach dem Anfügen vorhandenen Objekte. Er ist Null, falls die Funktion fehlschlägt.

```
Function ZObjekte()
```

Die Funktion gibt die Anzahl der im aktuellen Fall verfügbaren Objekte zurück.

```
Function FObjekt( _
    ByVal sSuche As String, _
    ByVal iPos As Long, _
    ByVal iFlags As  Long _
  ) As Long
```

Die Funktion sucht im aktuellen Fall nach einem nicht konkret bestimmbaren Objekt. SSuche enthält den Bezeichner oder eine Zeichenkette, nach der gesucht werden

---

[358] In der Beispielsanwendung wird erwartet, daß der Aktenname den Betriebssystemkonventionen von DOS gehorcht. Die Sicht ist jedoch zu eng. Vielmehr muß jeder eindeutige Name zulässig sein. Die gewählte Lösung ist jedoch einfach zu praktizieren.

soll. iFlags gibt an, wie nach der Zeichenkette gesucht werden soll. Als Standard wird davon ausgegangen, daß sSuche mit dem Bezeichner des Objekts übereinstimmen soll. Die Funktion ermöglicht es aber auch, nach ähnlichen Objekten zu suchen. Der Rückgabewert bezeichnet das Objekt, das in einer Liste der durch die Suche qualifizierten Objekte an der durch iPos definierten Position steht. Er ist Null, falls kein passendes Objekt gefunden wurde bzw. die Anzahl der qualifizierten Objekte kleiner als iPos ist.

cc) Werte

Auf die Werte von Objekten wird über den Index des Objekts und die Position des Wertes in der Werteliste des Objekts zugegriffen. Die Funktionen ähneln ansonsten vom Aufbau den bisher beschriebenen Funktionen:

```
Function GObjektWert( _ '                                   Ermittelt den Wert eines Objekts
    ByVal iX As Long, _
    ByVal iObjX As Long, _
    Wert As tWert _
  ) As Bool
Function SObjektWert( _ '                                   Ändert den Wert eines Objekts
    ByVal iX As Long, _
    ByVal iObjX As Long, _
    ByVal Wert As tWert _
  ) As Bool
Function AObjektWert( _                                     Fügt einen neuen Wert an die Werteliste eines Objekts an
    ByVal iObjX As Long, _
    ByVal Wert As tWert _
  ) As Long '       Rückgabewert: Position des neuen Wertes in der Werteliste
Function LObjektWert( _                                     Löscht einen Wert aus der Werteliste eines Objekts
    ByVal iX As Long, _
    ByVal iObjX As Long _
  ) As Bool
Function ZObjektWerte( _                                    Anzahl der Werte eines Objekts
    ByVal iObjX As Long _
  ) As Long '
```

Die Funktionen dienen dem Lesen, Ändern, Anfügen und Löschen sowie zur Ermittlung der Anzahl von Werten eines Objekts. iX bezeichnet jeweils die Position der Werteinformation in der Liste der Werte des Objekts. iObjX bezeichnet den Index des Objekts selbst. tWert[359] enthält den Wert selbst sowie Zusatzinformationen.

dd) Beziehungen

Auf Beziehungen wird über die zugehörige Relation zugegriffen. Ihr wird eine Liste von Beziehungen zugeordnet. Die Position einer Beziehung in dieser Liste identifiziert die Beziehung. Die Funktionen sollen hier überblicksartig gezeigt werden.

```
Function GBeziehung( _ '                                    Ermittelt die Daten einer Beziehung
    ByVal iX As Long, _
    ByVal iRelX As Long, _
    Beziehng As tBeziehung _
  ) As Bool
Function SBeziehung( _ '                                    Ändert die Daten einer Beziehung
    ByVal iX As Long, _
    ByVal iRelX As Long, _
    ByVal Beziehng As tBeziehung _
  ) As Bool
```

[359] tWert s.S. 121.

```
Function ABeziehung( _ '                          Fügt Daten einer Beziehung an eine Relation an
    ByVal iRelX As Long, _
    ByVal Beziehng As tBeziehung _
  ) As Long
Function LBeziehung( _ '                                  Löscht eine Beziehung
    ByVal iX As Long, _
    ByVal iRelX As Long _
  ) As Bool
Function ZBeziehungen( _ '                Anzahl der Beziehungen auf Grundlage einer Relation
    ByVal iRelX As Long _
  ) As Long
```

Die Funktionen dienen zum Lesen, Ändern, Anfügen und Löschen sowie zur Ermittlung der Anzahl von Beziehungen, die auf einer Relation basieren. Hier bezeichnet iX die Position der Beziehung in der Liste der Beziehungen einer durch iRelX identifizierten Relation. Die Struktur Beziehung enthält die Daten der Beziehung[360].

Oftmals mag es interessant sein, alle Beziehungen zu suchen, an denen ein bestimmtes Objekt beteiligt ist. Eine solche Funktion könnte[361] wie folgt aussehen:

```
Function FBeziehung( _ '                                    Sucht nach einer Beziehung
    iX As Long, _
    iRelX As Long, _
    Beziehng As tBeziehung, _
    ByVal iObjX As Long, _
    ByVal iPos As Long _
  ) As Bool
```

Die Parameter iX, iRelX und Beziehung empfangen die gesuchten Informationen. iObjX enthält den Index des gesuchten Objekts. Aus der Suche nach dem Objekt wird eine Trefferliste generiert. iPos enthält die Position der gesuchten Beziehung in dieser Trefferliste. Der Rückgabewert der Funktion ist *true*, wenn mindestens iPos Beziehungen gefunden wurden und die Daten der gesuchten Beziehung in die entsprechenden Parameter kopiert wurden. Ansonsten ist der Rückgabewert *false*.

Es ist kein Pendant zu dieser Funktionalität im DDE–Befehlssatz von Atlas definiert. Dennoch kann sich eine Funktionserweiterung auch Funktionalitäten der Datenbankschnittstelle zunutze machen, die nicht in der DDE-Schnittstelle vorgesehen sind.

e) Zufügen von Relationen im Installationsprogramm

Prinzipiell gibt es zwei Arten von Programmen, die auf Atlas zugreifen. Einerseits sind dies Programme, die Atlas zur Ablage ihrer originären Daten und zum Austausch dieser Daten mit anderen Programmen verwendet. So können Programme zur Berechnung von bestimmten Werten definierte Eingabewerte aus dem System entnehmen und das Rechenergebnis als definierten Wert in das System zurückschreiben. Andererseits gibt es Programme, die jeden Wert verarbeiten können, indem sie ihn etwa auf besondere Weise visualisieren. Diese Programme orientieren sich an den Gegebenheiten des Systems, wie sie es beim Benutzer vorfinden.

Die erste Kategorie von Programmen benötigt hingegen eine definierte Ausgangskonfiguration, um Daten ablegen oder erfragen zu können. Eine solche Konfiguration beruht darauf, daß definierte Relationen mit im System vorhanden sind. Dabei ist wie bereits dargestellt jedes Programm für die Konfigurierung des Systems selbst

---

[360] tBeziehung s.S. 122.

[361] Im Beispielsprogramm ist keine Funktion mit dieser Funktionalität vorgesehen.

verantwortlich. Es erledigt diese Aufgabe zweckmäßigerweise bei seiner erstmaligen Installation, dem *Setup*. Dabei geht das Programm im Prinzip bei jeder benötigten Relation nach folgendem Schema vor:

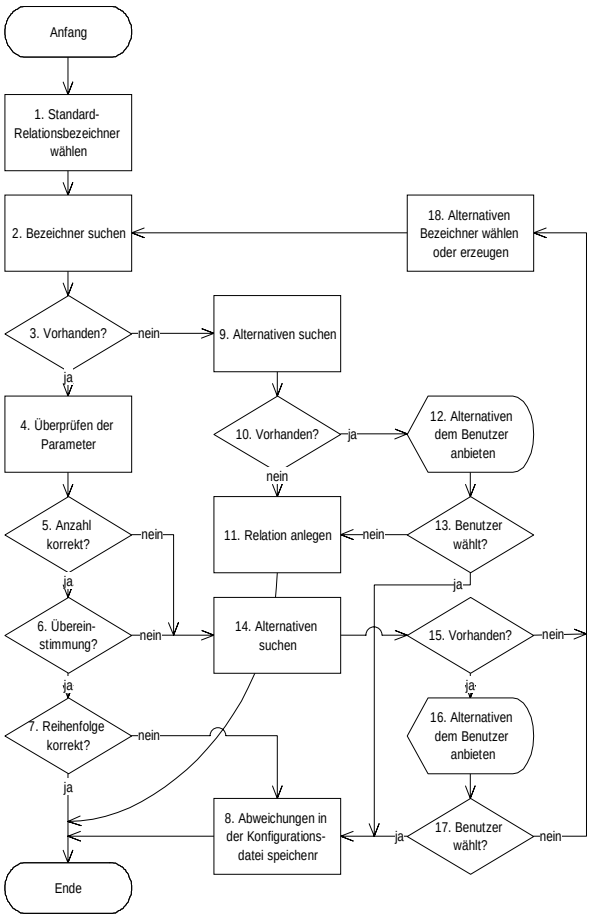


Abbildung 40: Schema zur Aktualisierung der globalen Datenbasis bei der Installation eines Programmes

Dieses Schema geht von einer nahezu optimalen Anpassungsfähigkeit des Anwenderprogramms aus. Zentrales Element dabei ist Punkt 8, nämlich die Möglichkeit, Änderungen, die sich aus dem vorgefundenen Zustand des Systems des Benutzers ergeben, durch eigene Umkonfiguration abzufangen. Dabei kommen folgende Möglichkeiten in Betracht:

– Eine benötigte Relation wird mit dem erwarteten Bezeichner nicht vorgefunden (3). Das System prüft, ob formal identische Relationen (d.h. mit identischen Parametern) vorhanden sind (9). Der Benutzer erhält die Möglichkeit (12) zu entscheiden, ob zwei Relationen inhaltlich identisch sind. Das System merkt sich dann den alternativen Bezeichner (8) und verwendet diesen.

– Es tritt ein Konflikt auf, da ein Relationsbezeichner mit formal unterschiedlichen Parametern auftritt (5 oder 6). Die Installation sucht zunächst wieder nach formal identischen Alternativen (14) und legt sie dem Benutzer vor (16). Werden keine inhaltlich und formal identischen Alternativen gefunden (15 und 17), so werden vom Installationsmechanismus ein oder mehrere alternative Bezeichner geprüft

(18 und 2). Der gewählte Bezeichner wird gespeichert (8) und zukünftig verwendet.

– Eine benötigte Relation ist auf dem System mit den gesuchten Parametern vorhanden, die Parameter werden jedoch in anderer Reihenfolge erwartet (7). Das Anwenderprogramm merkt sich die andere Reihenfolge (8) und übergibt die Parameter zukünftig in dieser Reihenfolge.

Mit diesem Verfahren ist es möglich, eine Anwendung auch dann in ein System zu integrieren, wenn dessen Vorgaben vom erwarteten Zustand abweichen. Eine derartige Konfigurierbarkeit erhöht den Programmieraufwand der Anwendung erheblich. Sie dient dabei letztlich nur dem Ausgleich von Schwächen im Bereich der eigentlich gewünschten Normierung. Im Fall einer idealen Normierung der Relationen reduziert sich die Aufgabe des Installationsprogramms auf folgendes sehr einfaches Vorgehen:

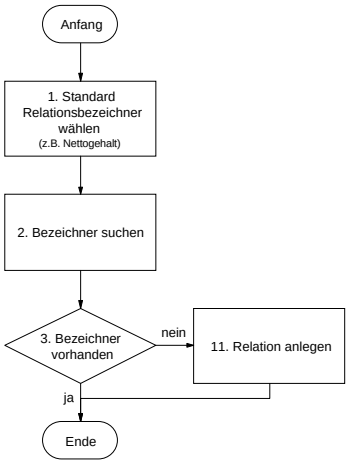


Abbildung 41: Installation bei Normierung der möglichen Relationen.

Das Installationsprogramm muß für die Erfüllung dieser Aufgaben direkt auf das Datenverwaltungsmodul zugreifen, da die DDE-Schnittstelle keine Funktionen zur Anlage von Relationen bietet. Da es sich bei einem Installationsprogramm regelmäßig um ein echtes (oft in einer BASIC-ähnlichen Skriptsprache entwickeltes) Programm handelt, stellt diese Anforderung keine Hürde dar. Das Installationsprogramm muß die Funktionen FRelation, ARelation und ARelParam einbinden. Es verwendet die Funktion FRelation für Schritt 2, d.h. zum Suchen der Relation in der globalen Datenbasis anhand des Relationsbezeichners. Mit Hilfe der Funktionen ARelation und ARelParam kann ein Programm im Bedarfsfall eine neue Relation und die benötigten Parameter anlegen. Das folgende Beispiel verdeutlicht dieses Vorgehen anhand der Relationen $Kind(k, v, m)$, $Vater(v, k)$ sowie $Mutter(m, k)$:

```
Dim Relation As tRelation
Dim Parameter As tParameter
Dim ixKind As Long ' Index der Relation "Kind"
Dim ixMutter As Long ' Index der Relation "Mutter"
Dim ixVater As Long '  Index der Relation "Vater"
Dim ix As Long '     Dummy

' Zunächst wird eine Relation gesucht. Dabei wird von einer Normierung
' der Relationen ausgegangen. D.h. auch, wenn eine Relation nicht angelegt ist,
' fehlen zwangsläufig die komplementären Relationen.

If FRelation("Kind", 1, 0) = 0 Then
   ' Wenn die Relation "Kind" nicht vorhanden ist, werden alle Relationen neu angelegt.
```

```
  Relation.lFlags = 0
  Relation.sBezeichner = "Kind"
  ixKind = ARelation(Relation)
  Relation.sBezeichner = "Vater"
  ixVater = ARelation(Relation)
  Relation.sBezeichner = "Mutter"
  ixMutter = ARelation(Relation)

  ' Nun werden die Parameter der Relationen zugefügt
  ' zuerst bei "Vater" und "Mutter" zur Relation "Kind"
  ix = ARelParam(ixKind, ixVater)
  ix = ARelParam(ixKind, ixMutter)

  ' danach bei "Kind" zu den Relationen "Vater" und "Mutter"
  ix = ARelParam(ixVater, ixKind)
  ix = ARelParam(ixMutter, ixKind)
End If
' Ende der Anlage der Relationen
```

Die Datenbankschnittstelle stellt Funktionen für den lesenden, ändernden und löschenden Zugriff auf alle Daten der Datenbasis zur Verfügung. In einigen Fällen werden auch Suchen angeboten. Für die Entwicklung eines Installationsprogramms muß direkt auf die Datenbankmaschine zugegriffen werden. Je nach Optimierung der Normierung von Relationen, ist diese Entwicklung aufwendiger oder einfacher.

4. Schnittstelle zur Funktionserweiterung

Die bisher behandelte Benutzerschnittstelle sowie die DDE-Schnittstelle dienten der Interaktion mit dem Programm Atlas. Im ersten Fall interagiert der Benutzer direkt mit dem Programm, im zweiten Fall bedient er sich einer weiteren Anwendung. Die zuletzt beschriebene Datenbankschnittstelle ist eine obligatorische Funktionsbibliothek für die Umsetzung der abstrakten Atlas-Syntax in eine physikalische Datenablage.

Im folgenden wird eine Schnittstelle beschrieben, die eine optionale Erweiterung der Atlas-Funktionalität ermöglicht. Hiermit können beispielsweise auch diejenigen Funktionen nachgebildet werden, die in dieser Arbeit zwar gefordert, jedoch nicht realisiert wurden. Diese Funktionserweiterung geschieht prinzipiell auf zwei unterschiedlichen Wegen:

1. Atlas reicht alle Anfragen, die es nicht aus eigenem Wissen beantworten konnte, an das Erweiterungsmodul weiter. Das Modul kann nun seinerseits auf beliebige Weise versuchen, die gestellte Frage zu lösen.
2. Ein Erweiterungsmodul kann die beschriebenen Atlas-Standardfunktionen um eigene Funktionen erweitern. Diese Funktionen werden über die DDE-Schnittstelle wie Systemfunktionen, d.h. über das Funktionspräfix @ angesprochen.

Hiermit kann der Leistungsumfang von Atlas gesteigert werden. Die folgenden Abschnitte behandeln die Vorgehensweise zur Einbindung dieser Funktionalität wiederum schematisch.

a) Einbinden eines Erweiterungsmoduls

aa) Voraussetzung

Ein Funktionserweiterungsmodul ist eine Laufzeitbibliothek (DLL) mit einer definierten Schnittstelle. Kernpunkt der Schnittstelle ist eine Einstiegsfunktion, über die Atlas einen ersten Kontakt mit dem Modul austauscht. Über diese Funktion werden auch diejenigen Anfragen ausgetauscht, die Atlas nicht aus eigener Kraft klären kann.

Die Einbindung eines Moduls erfolgt durch Eintragung in eine Modulliste der Konfigurationsinformationen. In einer Windows 3.x Anwendung ist dies eine Datei mit dem Namen *ATLAS.INI*[362]:

```
[Erweiterungen]
Modul=PROGSTRT.DLL
```

Die Einstiegsfunktion des Funktionserweiterungsmoduls hat den Bezeichner `AtlasModulF` mit folgenden Parametern:

```
Function AtlasModulF(ByVal iBotschaft as Integer, _
  ByVal iParam as Integer, _
  ByVal ID as Long, _
  ByVal lParam1 as Long, _
  ByVal lParam2 as Long) As Long
```

Eine Funktionsbibliothek, die als Atlas–Funktionserweiterung dienen soll, muß diese Funktion besitzen und die zentralen Prozesse der Funktion bedienen. Dabei bezeichnet `iBotschaft` die Art der Anfrage, die sogenannte Botschaft von Atlas an das Modul. Die unterschiedlichen Botschaften[363] werden in diesem und in den folgenden beiden Abschnitten systematisch behandelt. Die weiteren Parameter `iParam`, `lParam1` sowie `lParam2` enthalten Informationen, die abhängig sind von der jeweiligen Botschaft.

Für den Fall, daß mehrere Module in der Konfigurationsdatei eingetragen sind, werden alle Module geladen. Die Botschaften werden an die Module in der Reihenfolge ihrer Eintragung geschickt. An ein späteres Modul wird eine Botschaft nur dann geschickt, wenn das vorangegangene Modul die Botschaft nicht bearbeiten konnte.

bb) Initialisierung

Erkennt Atlas anhand des Eintrags in der Konfigurationsdatei, daß ein Erweiterungsmodul eingesetzt werden soll, so lädt es dieses Modul und versucht die Einstiegsfunktion zu erreichen. Gelingt dies, so ruft Atlas zunächst die Funktion mit der Botschaft `ATLM_INIT`. Es teilt dem Modul damit mit, daß es mit weiteren Aufrufen rechnen muß und gibt ihm Gelegenheit, die notwendigen Initialisierungen vorzunehmen.

Im Parameter `lParam1` übergibt Atlas die Adresse der eigenen Rückruffunktion. Über diese Funktion kann ein Modul Botschaften an Atlas senden. Die Funktion hat dieselbe Syntax wie die Einstiegsfunktion des Moduls. Sie erlaubt es dem Modul, auf die wesentlichen Funktionen von Atlas einschließlich der Datenzugriffsfunktionen zuzugreifen. Auf eine präzise Darstellung dieser Schnittstelle wird hier verzichtet, da sie in wesentlichen Punkten mit der beschriebenen Datenbankschnittstelle übereinstimmt. Weiterhin werden Hilfsfunktionen verfügbar gemacht, die die Zusammenstellung von Ausdrücken in der Atlas-Syntax unterstützen.

Im Parameter `lParam2` übergibt Atlas ein sogenanntes Handle des eigenen Programmfensters. Das Handle ermöglicht es dem Modul, selbständig mit dem Benutzer in Kontakt zu treten, ohne daß dieser einen Unterschied zum eigentlichen Atlas-Hauptprogramm feststellen kann. Insbesondere kann dem Benutzer während der Anzeige von Interaktionselementen des Moduls die Möglichkeit, auf das Hauptfen-

---

[362] In 32-BIT Windows–Versionen existiert eine allgemeine Registrierungsdatenbank.

[363] Engl.: *Message*. Dieser Parameter wird unter Windows auch oft als *wMsg* oder *wMessage* bezeichnet. *w* steht dabei nach der sogenannten polnischen Notation für den Variablentyp *Word*. Unter BASIC wird statt dessen meist *i* für *Integer* eingesetzt.

ster des Programms zuzugreifen, verweigert werden (sogenannte gebundene oder auch modale Dialogfenster).

Das Modul gibt mit der Funktion Informationen darüber zurück, welche weiteren Botschaften es unterstützt. Hierzu werden die Werte der unterstützten Botschaften mit logischem Oder verknüpft und als Rückgabewert an Atlas übergeben. Alle weiteren Botschaften werden nur dann an das Modul geschickt, wenn sie in den Rückgabeparameter eingetragen wurden. Schlägt die Initialisierung fehl, gibt die Funktion Null zurück.

cc) Modulreport

Dem Aufruf zur Initialisierung folgt die Botschaft ATLM_MODULINFO. Sie gibt dem Modul die Gelegenheit, eine Zeichenkette mit Informationen an Atlas zu übergeben. Atlas stellt diese Zeichenkette im Modulreport dar.

lParam1 enthält den Zeiger auf einen Datenpuffer, in den das Modul die Zeichenkette kopieren soll. wParam enthält die maximale Länge dieser Zeichenkette. Die Funktion gibt die Anzahl der tatsächlich in den Puffer kopierten Zeichen an Atlas zurück.

b) Anfragenbearbeitung

aa) Prinzip

Zur Anfragenbearbeitung übergibt Atlas dem Modul eine oder zwei meist unvollständig gefüllte Datenstrukturen mit der Bitte, die jeweils fehlenden Daten einzutragen. Welche Daten konkret nachgetragen werden müssen, wird durch die Art der Botschaft übermittelt.

Mit der Botschaft ATLM_GOBJEKT übermittelt Atlas an das Modul eine Anfrage nach einem Objekt. Das Objekt ist durch eine teilweise bekannte Beziehung gekennzeichnet. Die Beschreibung der Beziehung wird in lParam1 mit einem Zeiger auf eine Struktur vom Typ tBeziehung an das Modul übergeben. Das Modul versucht daraufhin, den Wert des Feldes Objekt(0) der Struktur anhand der anderen Feldinformationen zu ermitteln und dort einzutragen. Gelingt dies, gibt das Modul *1* zurück, ansonsten Null.

Mit der Botschaft ATLM_ZOBJEKT übermittelt Atlas an das Modul eine Anfrage nach der Anzahl der verfügbaren Objekte. Die Objekte sind durch eine Beziehung gekennzeichnet. Die Beschreibung der Beziehung wird in lParam1 mit einem Zeiger auf eine Struktur vom Typ tBeziehung an das Modul übergeben. Das Modul gibt die Anzahl der Beziehungen zurück. Es gibt *-1* zurück, wenn der Versuch fehlschlägt.

Mit der Botschaft ATLM_GWERT übermittelt Atlas an das Modul eine Anfrage nach dem Wert eines definierten Objekts. Im Parameter ID übergibt Atlas den Index des Objekts, im Parameter lParam1 wird ein Zeiger auf eine Struktur vom Typ tWert übergeben. Das Feld iTyp enthält den gesuchten Datentyp. Das Feld InfoAussage enthält den zugrunde zu legenden Ausgabefilter. Das Modul füllt das Ergebnis in das Feld vWert. Es gibt *1* zurück, wenn dies gelingt und Null, wenn es fehlschlägt.

bb) Werte über Relationen ermitteln

Mit der Botschaft ATLM_GWERTAUSBEZIEHUNG fragt Atlas bei einem Modul nach dem Wert eines Objekts, das durch eine Beziehung identifizierbar ist. Diese Anfrage ist die wohl wichtigste Anfrage. Ein Modul kann aus einer Relation, die einer Beziehung zugrunde liegt, möglicherweise Schlüsse darauf ziehen, wie der Wert zu ermitteln ist. Angenommen, eine Anwendung fragt nach dem Nettoeinkommen einer Person mit folgender Transaktion:

Aktion	Beschreibung	Wert
Anfrage	Nettoeinkommen(;Antragsgegner())	

Atlas ermittelt nun aus eigenem Wissen das Objekt *Antragsgegner*. Das Objekt *Nettoeinkommen* ist zwar unbekannt, wird aber nach erfolglosen Anfragen bei den Zusatzmodulen implizit instantiiert. Zuletzt soll dessen Zahlenwert ermittelt werden. Ein Zusatzmodul kann nun aus der Information, daß der Zahlenwert eines Objekts mit der Relation *Nettoeinkommen* beschrieben wird, einen mathematischen Algorithmus einsetzen, um dieses Nettoeinkommen zu ermitteln. Hierzu wird es das Bruttoeinkommen und eine bestimmte Anzahl von weiteren Angaben benötigen. Diese Angaben versucht es wiederum bei Atlas zu erfragen.

Das Modul kann auch das Wissen haben, zu welchen Relationen im System Anwendungen existieren, die entsprechend beschriebene Fakten ermitteln. Dazu startet es im Beispiel eine Anwendung zur Nettoeinkommensberechnung. Die Anwendung ermittelt die fehlenden Fakten aus eigenen Quellen, bei Atlas oder erfragt sie beim Benutzer. Sie informiert Atlas über das Faktum, sobald es ermittelt wurde.

Derartige Prozesse lassen lange Laufzeiten erwarten. Sie können zudem bei Atlas rekursiv verlaufen, wenn eine gestartete Anwendung im Zuge der Beantwortung einer Anfrage an Atlas weitere Anfragen richtet. Das Modul ist deshalb gehalten, auf eine Botschaft immer dann negativ zu antworten, wenn der Wert ohne weitere Interaktion mit dem Benutzer oder Transaktion mit Atlas nicht ermittelt werden kann. Atlas wird seinerseits eine Transaktionsanfrage der Ursprungsanwendung negativ bescheiden. Es kann jedoch sofort bei Eintreffen des korrekten Datums letztere Anwendung über diese Datenübermittlung informieren, sofern die Anwendung bei der ersten Anfrage um eine derartige ungefragte Information gebeten hat.

Zudem ist es notwendig, daß sich eine Anwendung aus bereits vorher beschriebenen Gründen in einem Multitasking-Betriebssystem regelmäßig neu über die Daten im System informiert. Sie sollte dies spätestens dann tun, wenn der Benutzer zwischenzeitlich mit einer anderen Anwendung gearbeitet hat. Die Anwendung reagiert dann auf erneute Aktivierung durch den Benutzer mit der Aktualisierung der benötigten Daten.

Das Vorgehen soll nochmals in einem Beispiel verdeutlicht werden: Eine Anwendung fragt bei Atlas nach dem Nettoeinkommen des *A*, um dieses in einen Schriftsatz einzutragen. Atlas kennt den Wert des hierbei neu instantiierten Objekts *Nettoeinkommen_A* nicht und reicht die Anfrage *Nettoeinkommen*(, *A*) an ein Erweiterungsmodul. Dieses ist in der Lage, ein Brutto-Netto-Berechnungsprogramm zu starten. Das Programm kann vorhandene Fakten auswerten, muß aber möglicherweise weitere Fakten beim Benutzer erfragen. Das Erweiterungsmodul bescheidet die Anfrage deshalb zunächst negativ, so daß im Schriftsatz eine Fehlermeldung erscheint. Gleichzeitig wird das Berechnungsprogramm gestartet und der Benutzer zur Eingabe der nötigen Angaben aufgefordert. Nach erfolgter Ermittlung der Fakten berechnet das Programm den gesuchten Wert. Spätestens sobald es beendet oder deaktiviert wird, übermittelt das Berechnungsprogramm alle erfragten und errechneten Fakten an Atlas. Sowie der Benutzer das Programm mit dem Schriftsatz wieder aktiviert, wird dieses die neuen Daten zur Grundlage der weiteren Arbeit machen. Der Fehlerwert im Schriftsatz wird nun durch den korrekten Nettowert ersetzt.

Die Botschaft ATLM_WERTAUSBEZIEHUNG stellt quasi eine Verknüpfung der Botschaften ATLM_WERT und ATLM_OBJKET dar. In lParam1 wird eine Datenstruktur des Typs tWert übergeben. In lParam2 wird ein Zeiger auf tBeziehung übergeben. Letztere Datenstruktur ist regelmäßig vollständig ausgefüllt. In der Datenstruktur tWert ist der Wert selbst vom Modul einzutragen. Der Rückgabewert der Funktion bezeichnet, ob dies gelungen (*1*) oder fehlgeschlagen ist (Null).

Mit der Bearbeitung von Anfragen, die aus der Faktenbasis des Zentralsystems nicht beantwortet werden können, kann ein Erweiterungsmodul jede erdenkliche Hilfe bei der

Suche nach Fakten leisten. Dabei bleibt die Grundlage die beschriebene DDE-Schnittstelle. Insbesondere kann ein Modul Relationen mit Regeln verknüpfen und so die beschriebenen Fakten ermitteln.

c) Zusatzfunktionen

Eine andere Form der Systemerweiterung ist dadurch gegeben, daß ein Modul den Umfang der Atlas-Systemfunktionen wie *@Datum*() oder *@Anzahl* durch eigene Funktionen erweitert. Zu diesem Zweck muß das Modul die Funktionen zunächst dem System mitteilen. Es fügt im ersten Schritt auf die ATML_INIT Botschaft hin den Parameter ATML_FUNKTIONSNAME dem Rückgabewert bei. Damit wird Atlas veranlaßt, die Namen der neuen Funktionen nacheinander abzufragen.

Atlas sendet daraufhin solange die Botschaft ATML_FUNKTIONSNAME an das Modul, bis das Modul Null zurückgibt als Zeichen dafür, daß alle Funktionsnamen übermittelt wurden. Als Parameter ID enthält die Botschaft den Index der Funktion in einer imaginären Liste von Funktionen des Moduls beginnend bei *1*. lParam1 zeigt auf einen Puffer, der den Funktionsnamen aufnehmen soll. wParam gibt die maximale Anzahl von Zeichen an, die der Puffer aufnehmen kann. Der Rückgabewert gibt die Anzahl der Zeichen an, die tatsächlich in den Puffer kopiert wurden.

Die Botschaft ATML_FUNKTIONSNAME wird auch dann an ein Modul gesendet, wenn Atlas in Erfahrung bringen möchte, ob eine benannte Funktion von diesem Modul unterstützt wird. Hierzu wird der Parameter ID von Atlas auf den Wert Null gesetzt. lParam1 zeigt auf einen Puffer, der bereits den fraglichen Funktionsnamen enthält. Das Modul gibt den Index der Funktion in seiner Funktionsliste zurück. Der Wert ist Null, falls die Funktion von dem Modul nicht bedient wird. So ist es möglich, daß sich Atlas rückversichert, ob eine unbekannte Funktion in einer Anfrage an das System von einem Modul unterstützt wird. So können auch nicht angemeldete Funktionen unterstützt werden.

Mit der Botschaft ATML_FUNKTION löst Atlas schließlich die Ausführung einer Funktion durch das Modul aus. lParam1 enthält die vollständige Zeichenkette, die von einer anderen Anwendung in einer Transaktion übermittelt wurde. ID enthält die Position des Funktionsaufrufs in der Zeichenkette. Diese Position ist normalerweise *1*. lParam2 zeigt auf einen Puffer, der den Rückgabewert der Funktion aufnehmen kann. Dieser Wert wird unverarbeitet an die anfragende Anwendung weitergereicht. wParam bezeichnet die maximale Anzahl der Zeichen, die in den Puffer kopiert werden können. Das Modul gibt Null zurück, falls die Funktion nicht ausgeführt werden konnte, anderenfalls *1*.

d) Beispiel

Das folgende Beispiel einer Einstiegsfunktion ist aus Gründen der einheitlichen Darstellung in Visual–BASIC Syntax gehalten, obwohl in dieser Sprache keine Laufzeitbibliotheken entwickelte werden können. Insbesondere läßt BASIC keine Typkonvertierungen zu, wie sie in diesem Fall benötigt werden. Die Parameter lParam1 und lParam2 werden deshalb hier als Typ *Any* dargestellt.

```
' Die von dem Modul zur Verfügung gestellten Funktionen
' werden wie von Atlas vermutet in einer Liste abgelegt.
Const Funktion(1) = "Datum" ' Gibt das aktuelle Datum zurück.
Const Funktion(2) = "Zeit" ' Gibt die aktuelle Zeit zurück.
Const Funktion(3) = "Automatik" ' Schaltet die automatische Programmausführung ein oder aus.
Const MaxFunktionen = 3

' Einige Deklarationen sollten in VB außerhalb der Funktion erfolgen
Dim eBeziehung As tBeziehung
Dim eWert As tWert
Dim eRelation As tRelation
```

```
' Die folgende Variable enthält eine Information darüber, ob das Modul bei Anfragen
' automatisch andere Programme aufrufen soll.
Dim bAutomatik As Integer


Function AtlasModulF(ByVal iBotschaft as Integer, _
    ByVal iParam As Integer, _
    ByVal ID As Long, _
    ByVal lParam1 As Any, _
    ByVal lParam2 As Any) As Long
' Zu Beginn wird der Rückgabewert auf den ablehnenden Wert 0 gesetzt
  AtlasModulF = 0
' Die Funktion besteht im Prinzip aus einer Case-Anweisung,
' die die einzelnen Botschaften abarbeitet.
  Select Case iBotschaft
    ' Die Botschaft ATLM_INIT erhält als Rückgabewert die im folgenden
    ' bearbeiteten Botschaften.
    Case ATLM_INIT:
      AtlasModulF = ATLM_MODULINFO + _
          ATLM_WERTAUSBEZIEHUNG + _
          ATLM_FUNKTIONSNAME + _
          ATML_FUNKTION
      Exit Function
'   ATLM_MODULINFO erhält Informationen über das Modul
    Case ATLM_MODULINFO:
      lParam1 = "Atlas Programmstarter, © 1995, Matthias Kraft"
      AtlasModulF = Len(lParam1)
      Exit Function
'   Hier werden die Funktionen angemeldet
    Case ATLM_FUNKTIONSNAME:
      If ID = 0 Then ' Überprüfen, ob die Funktion in lParam1 bedient wird.
        For x = 1 To MaxFunktionen
          If UCase(lParam1) = UCase(Funktion(x)) Then
            AtlasModulF = x
          EndIf
        Next
      Else '      Angabe des mit ID identifizierten Funktionsnamens
        If ID <= MaxFunktionen then
          lParam1 = Left$(Funktion(ID), wParam)
          AtlasModulF = Len(lParam1)
        EndIf
      EndIf
      Exit Function
'   An dieser Stelle reagiert das Modul auf die Anfrage nach einem Wert.
'   Wird nach dem Alter eines Objekts gefragt,
'   so wird ein Programm gestartet. Ihm wird als Kommandozeilenparameter der Bezeichner
'   des Objekts übergeben, dessen Alter gesucht wird.
    Case ATML_WERTAUSBEZIEHUNG:
      eBeziehung = lParam1
      eWert = lParam2
'     Die folgede Funktion wurde vorher (nicht abgedruckt) deklariert.
'     Sie ermittelt die Relation, die einer Beziehung zugrunde liegt.
      eRelation = RelationAusBeziehung(eBeziehung)
      if UCase$(eRelation.sBezeichner) = "ALTER" Then
        Shell "alter.exe" + " " + eBeziehung.Objekt(0).szBezeichner
      EndIf
      Exit Function '  Die Funktion wird mit Null beendet!
```

```
'   Zuletzt wird auf die Botschaft zur Ausführung von Funktionen reagiert.
    Case ATML_FUNKTION:
      F$=UCase$(lParam1)
      If Instr(F$,"@"+UCase$(Funktion(1)))>0 Then '                    Angabe des Tagesdatums
        lParam2 = Left$(Date$(), wParam)
      ElseIf Instr(F$,"@"+UCase$(Funktion(2)))>0 Then '                Angabe der aktuellen Uhrzeit
        lParam2 = Left$(Time$(), wParam)
      ElseIf Instr(F$,"@"+UCase$(Funktion(3)))>0 Then '                Umschalten des Programmstarts
'     In den Antwortpuffer wird immer der vorangegangene Wert kopiert.
        If bAutomatik = True Then
          lParam2 = Left$("EIN", wParam)
        Else
          lParam2 = Left$("AUS", wParam)
        EndIf
        If Instr(F$,"EIN") > 0 Then
          bAutomatik = True
        ElseIf Instr(F$,"AUS") > 0 Then
          bAutomatik = False
        End If
      End If
      AtlasModulF = Len(lParam2)
  End Select
End Function
```

Über Erweiterungsmodule kann nicht nur das Auffinden unbekannter Fakten unterstützt werden. Vielmehr kann auch der Befehlsumfang des Systems beliebig erweitert werden.

Atlas erlaubt den Zugriff über Schnittstellen unterschiedlicher Funktionalitäten. Die DDE-Schnittstelle dient dem eigentlichen Zugriff auf die Fakten eines Falls. Die Datenverwaltungsschnittstelle und die Erweiterungsschnittstelle dienen vor allem einer zukunftsorientierten Systemanpassung. Sie sind gerade bei Prototypen sinnvoll, um einen abgeschlossenen Funktionsumfang erweitern zu können, ohne in das Programm selbst eingreifen zu müssen.

E. Anwendungsbeispiele

In diesem Kapitel werden exemplarisch einige Beispielanwendungen vorgestellt, die Atlas als Austauschbasis für Fakten verwenden. Wie bei Atlas selbst wird hier lediglich eine detaillierte Spezifikation abgegeben. Dazu werden Schemata der Bedienerführung sowie Transaktionsprotokolle mit Atlas eingesetzt. Zentrale Vorgänge werden anhand von Visual–BASIC Quellcode erläutert. Als Beispielsdomäne wird das Familienrecht gewählt. Bei einigen Anwendungen wird eine Annäherung an existierende Produkte oder Konzepte gesucht. Gleichzeitig soll hiermit ein exemplarischer Test für die Brauchbarkeit des Systems erstellt werden.

1. Aufnahmebogen

a) Konzept

Der Aufnahmebogen *Familiensachen* wurde bereits beschrieben[364]. Er dient der Erfassung der regelmäßig benötigten Fakten eines familienrechtlichen Falls. Wie die meisten gedruckten Arbeitswerkzeuge ist er nur wenig flexibel. Er geht von einem Standardszenario eines Falls aus. Hinzu kommen Optionen für typischerweise auftretende Fallvarianten.

Ein elektronischer Aufnahmebogen erfüllt wie auch das gedruckte Pendant die Aufgabe, während des ersten Kontaktes mit einem Mandanten die wesentlichen Fakten zu erfassen. Hierbei kommt es auf eine möglichst wenig aufwendige Erfassungsform

[364] S. S. 54

an, die nicht zum zentralen Thema des Mandantengespräches wird. Insbesondere können Fragen, die sich nicht klären lassen auch dann hintangestellt werden, wenn sie möglicherweise errechenbar sind oder in anderer Weise auf sie geschlossen werden kann. Wesentlich bei dem ersten Mandantengespräch ist vielmehr, daß es möglichst reibungslos abläuft. Erst wenn dezidiert nach bestimmten Beträgen, etwa den Prozeßkosten oder der zu erwartenden Unterhaltsleistung gefragt wird, mag es angesagt sein, über den Aufnahmebogen hinaus Anwendungen aufzurufen, die hierzu eine Lösung bereithalten. Zudem sollte die Organisation der Anwaltskanzlei gewährleisten, daß der Anwalt selbst nicht Informationen eintragen muß, die etwa ein Anwaltsgehilfe oder der Mandant selbst eingeben kann[365].

Anders als der gedruckte Aufnahmebogen, muß sich ein elektronischer allerdings flexibler auf die individuellen Einzelheiten des Falls anpassen. Insbesondere wird er anstelle von meist statischen Listen für Kinder, bestimmte Eigentumsgegenstände etc. eine dynamische Verwaltung dieser an dem Szenario Beteiligten bieten. Dem gedruckten Formular immanente Obergrenzen sind hier genauso wenig notwendig wie die Anzeige leerer und nicht auszufüllender Bereiche.

b) Oberfläche

Es ist nicht Ziel dieser Arbeit vielschichtige Details einer Benutzeroberfläche darzustellen. Insbesondere der Bereich der Softwareergonomie ist so komplex, daß er einer eigenständigen Abhandlung bedürfte[366].

In dem hier vorgestellten Konzept wird davon ausgegangen, daß ein Aufnahmeformular wie ein gedrucktes Formular eine Kette von aneinanderhängenden Eingabeposten darstellt. Die einzelnen Posten sind nach Themenbereichen gruppiert. Eine Gruppe paßt zweckmäßiger Weise auf eine Bildschirmseite bzw. in ein Programmfenster. Die Gruppen werden dynamisch zusammengesetzt. Gibt der Benutzer beispielsweise eine Zahl von 2 Kindern ein, so werden zwei Gruppen zur Stammdatenerfassung für die Kinder aneinandergehängt. Der Benutzer kann durch die aktuelle Kette beliebig blättern. Alle Eingabeelemente, die sich in einer Kette befinden, sind direkt zu erreichen. Die Änderung ihrer Werte kann sich unmittelbar auf die weitere Zusammensetzung der Kette der Eingabeelemente auswirken.

Neben dem direkten Zugang des Benutzers auf jedes Eingabeelement ist es zweckmäßig, daß der Benutzer durch die Eingaben vom Programm geführt wird. Das Programm bestimmt dann nach jeder Eingabe, welches Eingabeelement als nächstes auszufüllen ist.

c) Exkurs: Realisierung der Oberfläche mit einem objektorientierten Programmsystem

Der folgende Exkurs widmet sich den Grundlagen der Gestaltung von Windowsoberflächen mit Visual–BASIC. Er dient dem Grundverständnis der nachfolgenden Ausführungen. Windows stellt, wie viele andere Betriebsysteme, einen definierten Satz von Eingabeelementen zur Verfügung. Diese können beliebig in dem sogenannten Client–Bereich[367] des Programmfensters plaziert werden. Dies erfolgt in der Regel mit einem graphischen Bearbeitungswerkzeug. Das Ergebnis wird in einem Skript abgespeichert.

---

[365] Vgl. *Kirchner*, CoR 1995, 324 ff.

[366] Eine ausführliche Untersuchung hierzu im juristischen Bereich stammt von *Viefhues*, **Methodik...**, a.a.O.

[367] Das ist der Bereich eines Fensters, dessen Inhalt das Programm unmittelbar beeinflussen kann. Hierzu gehören insbesondere **nicht** der Rand, die Titelzeile sowie das Menü des Fensters.

Visual–BASIC betrachtet jedes dieser Eingabeelemente als ein *Object*[368]. Aussehen, Position und Verhaltensweisen werden zum einen durch die Art des *Objects* (seine sogenannte Objektklasse) und durch seine *Properties*[369] bestimmt. Ein Skript für eine Maske besteht nun aus einer Aufzählung von Eingabeelementen, ihren Objektklassen und ihren Properties. Die Beschreibung für ein Element folgt der Syntax:

```
Begin ObjektKlasse ObjektName
  PropertyName = PropertyWert
...
[ untergeordnete Elemente]
End
```

Mehrere Elemente können verschachtelt werden. Hierdurch erfolgt eine Gruppierung der Elemente. Allen Elementen übergeordnet ist ein Object der Klasse *Form*. Der folgende Quellcode zeigt ein Visual–BASIC Skript für eine Maske. Das Ergebnis ist in dem Abbild darunter skizziert:

```
Begin Form HauptForm
  Caption = "Aufnahmebogen"
  ClientHeight = 4440
  ClientLeft = 345
  ClientTop = 1035
  ClientWidth = 4905
  Height = 4845
  Left = 285
  LinkTopic = "Form1"
  ScaleHeight = 4440
  ScaleWidth = 4905
  Top = 690
  Width = 5025
  Begin VScrollBar VertBildLauf
    Height = 4455
    Left = 4680
    TabIndex = 18
    Top = 0
    Width = 255
  End
  Begin Frame GrpPerson
    Caption = "Ehemann"
    Height = 2055
    Index = 0
    Left = 120
    TabIndex = 0
    Top = 120
    Width = 4335
    Begin TextBox EdGebDat
      Height = 285
      Index = 0
      Left = 1440
      TabIndex = 8
      Top = 1620
      Width = 1335
    End
    Begin TextBox EdGebName
      ...
    End
    Begin TextBox EdVorname
      ...
    End
    Begin TextBox EdName
      ...
    End
    Begin Label StGebDat
      Caption = "Geburts&datum:"
      Height = 255
      Index = 0
      Left = 120
      TabIndex = 7
      Top = 1620
      Width = 1215
    End
    Begin Label StGebName
      ...
    End
    Begin Label StVorName
      ...
    End
    Begin Label StName
      ...
    End
  End
End
```

[368] Im Deutschen ist auch die Bezeichnung Objekt geläufig. Zur Vermeidung von Verwechslungen mit Objekten im Sinne dieser Arbeit wird hier jedoch der englische Ausdruck beibehalten.

[369] Auch hier wird der englische Ausdruck für Eigenschaften wegen der Verwechslungsgefahr beibehalten.

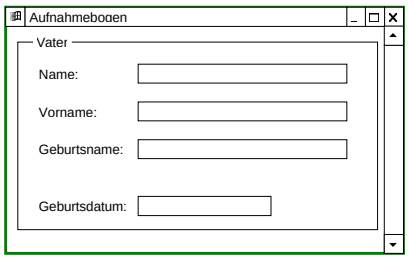


Abbildung 42:Aufnahmebogen

Das Skript enthält dabei nur die Properties eines Objects, die nicht mit dem Standardwert für diese Objektklasse belegt sind. Durch die Änderung der Properties beim Programmlauf kann das Aussehen des Fensters und seiner Elemente verändert werden. Sie können etwa über die Properties *Top* und *Left* verschoben oder über die Property *Visible* gänzlich ausgeblendet werden.

Die eigentliche Programmsteuerung erfolgt über sogenannte Methoden[370]. Das sind Funktionen, mit denen ein Object auf bestimmte Ereignisse[371] im System reagiert. Die Reaktion ist für eine Objektklasse solange identisch, bis sie vom Programmautor für ein bestimmtes Element überschrieben wird. Eine Funktion oder Prozedur wird dadurch zur Methode, daß sie bei der Namengebung folgende Syntax einhält:

```
Sub ObjectName_EreignisName([Parameter])
```

Zudem besitzt jede Objektklasse Methoden, die nicht auf Ereignisse reagieren, sondern bestimmte Verhaltensweisen des Objects auslösen. Diese Methoden können allerdings nicht überschrieben werden.

d) Anbindung einzelner Eingabeelemente an Atlas

Visual–BASIC erlaubt eine besonders einfache Implementierung des Transaktionsprotokolls DDE in Programme. Die Einbindung erfordert nahezu keine Programmierung. Hierdurch erreicht der Code eine Abstraktionshöhe, die kaum betriebssystemspezifische Merkmale mehr aufweist. Die Sprache eignet sich deshalb besonders gut für die Erläuterung des Vorgehens.

Visual–BASIC geht prinzipiell davon aus, daß jedes Eingabeelement eine eigene Konversation mit einem DDE-Server, hier also Atlas führt. Hierzu erhält jedes Object den Konversationspartner und das entsprechende Konversationsthema als Property *LinkTopic* zugewiesen. Als Property *LinkItem* wird das Element angegeben, dessen Daten das Objekt enthalten soll. Über die Veränderung der Property *LinkMode* wird die Konversation aufgebaut bzw. abgebrochen. Der folgende Ausschnitt eines Skripts zeigt die Anbindung des Elements *EdNachName*[372] an Atlas. Dabei soll der Name des Ehemanns in das Kontrollelement eingetragen werden:

---

[370] engl.: methods.

[371] engl.: events.

[372] Das Präfix „Ed" steht für „Edit", also Eingabefeld. „St" steht beispielsweise für „Static". Damit sind Elemente gemeint, deren Inhalt der Benutzer nicht ändern kann. Meist sind dies die Beschriftungen des Eingabeelementes ansonsten gleichen Namens. Die Präfixe dienen nur der besseren Lesbarkeit des Codes. Syntaktisch haben sie keine Bedeutung.

```
...
  Begin TextBox EdName
    Height = 285
    Index = 0
    Left = 1440
    LinkItem = "Nachname(;Ehemann(;))"
    LinkMode = 2 'Manual
    LinkTopic = "Atlas|eine Akte"
    TabIndex = 8
    Top = 1620
    Width = 1335
  End
...
```

Die eigentliche Atlas-Anfrage für den Inhalt des Eingabefeldes wird als LinkItem–Property des Elements selbst übergeben. Wird die LinkMode–Property auf dem Wert *2* gesetzt, so muß das Hauptprogramm die Transaktionen selbst anweisen. Dies erfolgt über die Methoden *LinkPoke* und *LinkRequest* des jeweiligen Objects. So ist es möglich, den Inhalt eines Eingabefeldes aus den Atlas-Daten zu aktualisieren oder ihn umgekehrt an Atlas zu übermitteln.

Die einfachste Form, alle Daten einer Maske aus Atlas zu erfragen, erfolgt nun so, daß alle Elemente der Maske in einer Schleife dazu aufgefordert werden, die Daten bei Atlas zu erfragen. Jedes Element weiß dabei selbst, welche Daten es zu erfragen hat. Eine Routine, die dies in der aktuellen Maske (In Visual–BASIC hat sie immer den Bezeichner *Me*) vornimmt, könnte wie folgt aussehen:

```
Sub AktualisiereDatenAusAtlas()
' Die Kollektion Controls einer Form enthält alle Kontrollelemente der Form.
' Die Eigeschaft Controls.Count gibt die Anzahl der Kontrollelemente an.
' Die folgende Schleife arbeitet also alle Kontrollelemente der Form ab.
  For x = 0 to Me.Contros.Count - 1
''     Es werden nur die Kontrollelemente zur Anfrage aufgefordert,
''     deren LinkItem- und LinkTopic-Property nicht leer ist.
    If Me.Controls(x).LinkItem <> "" And Me.Controls(x).LinkTopic <> "" Then
      Me.Controls(x).LinkMode = 2 ' Verbindungsaufbau
''       Die an sich asynchrone DDE-Anfrage wird von BASIC synchronisiert.
''       Die Antwort wird sofort als Inhalt des Kontrollelements angezeigt.
      Me.Controls(x).LinkRequest ' Anfrage
'        Aufgrund der Synchronisation durch BASIC kann die Verbindung
'        Nach der Anfrage sofort wieder abgebrochen werden.
      Me.Controls(x).LinkMode = 0 ' Verbindungsabbruch
    End If
  Next
End Sub
```

Der Code besitzt keine Fehlerbehandlung. Eine Routine, die den Inhalt aller Kontrollelemente an Atlas übermittelt, sieht prinzipiell identisch aus. Sie ruft lediglich die LinkPoke–Methode des jeweiligen Elements auf. Dennoch muß die Übermittlung von Daten an Atlas konzeptionell genauer geplant werden, da Atlas die Daten bei jeder Übermittlung neu ablegen würde. Die Feinheiten dieser Konzeptplanung sollen im folgenden dargelegt werden.

e) Datenübermittlung

Der vorangegangene Abschnitt zeigt, daß der Umgang mit dem Transaktionsprotokoll in der abstrakten Form von Visual–BASIC prinzipiell kein erhebliches Problem darstellt. Die Konversation wird zumindest bei statischen Masken im Zeitpunkt des Entwurfs vorbereitet. Die Herstellung der Verbindung, die Anfrage und ihr Abbruch bestehen dann aus drei Zeilen Code. Dieser kann sogar noch verkürzt werden, da die LinkMode-Property bereits zur Entwurfszeit eingestellt werden kann und der Konversa-

tionsabbruch verzichtbar ist. Problematischer ist das Konzept, nach welchem die Daten in der einen oder anderen Richtung aktualisiert werden sollen.

aa) Umgang mit Szenarien

aaa) Grundproblem

Wie bereits dargestellt, geht ein Formular von einem Szenario aus, das sich zum einen in mehrere Unterszenarien aufgliedern läßt, zum anderen mehrere Varianten zuläßt. Die kleinste Form eines Szenarios ist eine Beziehung mehrerer Objekte zueinander. So kann die Ehe zweier Menschen als Szenario angesehen werden. Das Szenario des Aufnahmebogens *Familiensachen* geht von einer Ehe mit einer definierten Anzahl ehelicher Kinder sowie den gesetzlich bestimmten Vermögensmassen der Beteiligten aus. Dieses Szenario ergibt ein komplexes Gebilde von verschiedenartigen Beziehungen.

Aufgrund der Tatsache, daß die Fakten in Atlas aus den verschiedensten Anwendungen stammen können, muß sich ein Programm regelmäßig darüber erkundigen, wie sich der Fall anhand der verfügbaren Fakten darstellt. Im Fall des Aufnahmebogens ist zunächst davon auszugehen, daß diese Fakten noch unvollständig sind, da er regelmäßig am Anfang des Mandats ausgefüllt wird. Es kann jedoch auch nicht ausgeschlossen werden, daß der Aufnahmebogen später zur Orientierung geöffnet wird und dann die Fakten für das Grundszenario bereits zu komplex sind.

bbb) Eingabe der Grundszenarien

Durch die manchmal weltfremd anmutenden Konstruktionen, die Atlas bei der Zuweisung von Werten wie Namen, Geburtsdatum etc. voraussetzt, häufen sich die angenommenen Beziehungen und Objekte. Dieser Effekt wird dadurch wieder gemildert, daß Atlas die implizite Instantiierung von Beziehungen und Objekten zuläßt und dabei selbständig die Objektbezeichner vergibt.

Bevor ein Eingabefeld vom Programm aufgefordert wird, seinen Inhalt als Faktum an Atlas zu übermitteln, ist es meist notwendig, bestimmte Objekte und Beziehungen dem System mitzuteilen. So ist eine Beschreibung eines Faktums NachName(;Ehemann()) nur dann verständlich, wenn bereits eine Beziehung mit der Relation $y=Ehemann(ehe)$ festgelegt wurde. Das System ist nämlich nicht in der Lage, Objekte mit vernünftigen Bezeichnern implizit zu instantiieren, wenn nicht wenigstens einer der Parameter einen sinntragenden Namen hat. Ein Anwendungsprogramm wie der Aufnahmebogen wird also vor der ersten Übermittlung einzelner Fakten das Grundszenario erfassen müssen. Es muß zumindest fallspezifische Bezeichner der Ehegatten ermitteln.

Die Anwendung *Aufnahmebogen* kann wie jede andere Anwendung nicht davon ausgehen, daß sie geöffnet wird, ohne daß bereits Fakten für den aktuellen Fall bekannt sind. Insbesondere muß sie damit rechnen, daß sie selbst bereits einmal Fakten an das System übermittelt hat und zu einem späteren Zeitpunkt zum zweiten Mal gestartet wird. Aus diesem Grund wird sie beim Start einen Abgleich mit den bekannten Fakten vornehmen. Das Programm wird also zunächst überprüfen, ob es bereits eine Ehe mit entsprechenden Teilnehmern gibt. Ist dies allerdings nicht der Fall, so läßt es eine Datenübermittlung erst dann zu, wenn die Beziehung *Ehe* sowie die Beziehungen *Ehemann* und *Ehefrau* dem System übermittelt wurden. Hierzu synthetisiert das Programm in der Regel aus dem Namen und dem Vornamen einer Person einen Vorschlag für dessen Bezeichner. Es wird ihn eventuell dem Benutzer zur Änderung vorlegen.

Der folgende Programmausschnitt zeigt die Übermittlung des Szenarios, falls noch keine entsprechenden Fakten vorhanden waren: (Die folgenden Codefragmente setzen die Funktionen *Anfrage*[373] und *Übermittlung*[374] voraus. Diese Funktionen setzen die asynchronen Transaktionen des Protokolls in synchrone Zugriffe um. Dabei bedienen sie sich eines versteckten Visual–BASIC Kontrollelements.)

```
Sub InstalliereSzenarioEhe()
   Dim sTrans as String
'  Zunächst wird abgeprüft, ob in den Eingabeelementen Angaben für die Bezeichner
'  der Ehegatten enthalten sind. Ist dies nicht der Fall, so wird die Funtkion
'  abgebrochen.
'  Der Benutzer könnte versehentlich eine Leerstelle vor oder hinter dem Wort
'  eingegeben haben, deshalb Trim$.
'  Das Eingabefeld EdBzEhemann enthält den Bezeichner des Ehemanns.
   sMann = Trim$(EdBzEhemann.Text)
   sFrau = Trim$(EdBzEhefrau.Text)
   If sMann = "" Or sFrau = "" Then Exit Sub
'  Im String sTrans wird die Beschreibung der Ehe zusammengesetzt.
   sTrans = "@Beziehung(Ehe(;$" + sMann + ";$" + sFrau + "))"
   Übermittlung sTrans, "Ja" 's. Fußnote Fehler! Textmarke nicht definiert.
'  Atlas legt nicht automatisch die komplementären Beziehungen an.
'  Dies muß quasi von Hand geschehen.
   sTrans = "@Beziehung(Ehemann($" + sMann + ";Ehe()))"
   Übermittlung sTrans, "Ja"  ' s. Fußnote Fehler! Textmarke nicht definiert.
   sTrans = "@Beziehung(Ehefrau($" + sFrau + ";Ehe()))"
   Übermittlung sTrans, "Ja"  ' s. Fußnote Fehler! Textmarke nicht definiert.
'  Damit der Benutzer die Bezeichner nicht mehr ändert,
'  werden die Eingabefelder geperrt.
   EdBzEhemann.Enabled = False
   EdBzEhefrau.Enabled = False
End Sub

Function Anfrage(ByVal sBeschreibung As String) As String
' Diese Funktion fragt Atlas nach einem durch sBeschreibung$ beschriebenen Wert durch.
' Die Antwort von Atlas wird auch als Rückgabewert der Funktion zurückgegeben.
   StDDE.LinkItem = sBeschreibung
   StDDE.LinkMode = 2
   StDDE.LinkRequest
   StDDE.LinkMode = 0
   Anfrage = StDDE.Caption
End Function.

Sub Übermittlung(ByVal sBeschreibung As String, ByVal sWert As String)
' Diese Funktion übermittelt Fakten, die durch sBeschreibung$ beschrieben werden.
   StDDE.LinkItem = sBeschreibung
   StDDE.Caption = sWert
   StDDE.LinkMode = 2
   StDDE.LinkPoke
   StDDE.LinkMode = 0
End Sub.
```

[373] S. 183.

[374] S. 183.

Hierdurch werden also, angenommen die Eingabefelder für die Bezeichner enthalten die Werte *Peter* und *Tina*, folgende Transaktionen ausgelöst:

Aktion	Beschreibung	Wert
Übermitteln	@Beziehung(Ehe(;$Peter;$Tina))	Ja
Übermitteln	@Beziehung(Ehemann($Peter;Ehe))	Ja
Übermitteln	@Beziehung(Ehefrau($Tina;Ehe))	Ja

Nach dieser Aktion kennt Atlas also die Beziehungen *Ehe Peter-Tina = Ehe*(*Peter, Tina*) sowie die komplementären Beziehungen *Peter = Ehemann*(*Ehe Peter-Tina*) und *Tina = Ehefrau*(*Ehe Peter-Tina*). Vernünftigerweise wird ein Programm also erst dann ein Szenario übermitteln, wenn es sprechende Namen für die Objekte bilden kann. Nun ist auch eine Transaktion eines Kontrollelements etwa folgender Art sinnvoll:

Aktion	Beschreibung	Wert
Übermitteln	Nachname(;Ehemann(;))	Lehmann

Die einzelnen Kontrollelemente können ihre Daten also erst dann an das System übermitteln, wenn das Programm die Beziehungen des Grundszenarios festgelegt hat.

ccc) Komplexe Ausgangslagen

Ein elektronischer wie auch ein gedruckter Aufnahmebogen kann nicht beliebig verzweigte Sachverhalte abbilden. Ersterer kann sich zwar prinzipiell jeder Situation dynamisch anpassen. Das maskenorientierte System verliert jedoch erheblich an Transparenz und überfordert letztlich den Benutzer bei der Eingabe, wenn es eine gewisse Komplexität überschreitet. So kann es passieren, daß das tatsächliche Szenario eines Falls zu komplex ist, um es in dem Aufnahmebogen abzubilden. Betrachtet man beispielsweise die oben ausformulierte Anfrage zum Eingabefeld *Name des Ehemanns* NachName(;Ehemann()) so ist diese nur eindeutig, wenn in dem aktuellen Fall auch nur eine Person in ihrer Funktion als Ehemann relevant ist. Dies entspräche nicht nur dem Grundszenario, von dem der Aufnahmebogen ausgeht, sondern auch der überwiegenden Praxis.

Das Programm, welches den Aufnahmebogen abbildet, kann, wie bereits dargelegt, nicht sicher davon ausgehen, daß der bearbeitete Fall nicht schon von einer anderen Anwendung modifiziert wurde. Es muß sich also versichern, daß ein Szenario vorliegt, das noch von ihm verarbeitet werden kann. Ist das Szenario des Falls bereits komplexer, so muß es entweder eine Bearbeitung ablehnen oder die Elemente ausfiltern, die dem Grundszenario entsprechen. Es könnte beispielsweise mit einer einfachen Anfrage @Anzahl(Ehe()) überprüfen, ob der Fall von mehr als einer Ehe ausgeht. Es kann den Benutzer nun fragen, welche Ehe er behandeln möchte. In diesem Fall ist jedoch die Anfrage des Kontrollelements Ehemann(;) nicht mehr ausreichend präzise. Sieht also ein Programm die Möglichkeit vor, auf komplexere Szenarien einzugehen, so muß es die Anfragen der einzelnen Kontrollelemente entsprechend modifizieren. Das folgende Beispiel zeigt eine derartige Modifikation für ein Kontrollelement:

```
Sub KomplexesSzenario()
  Dim iEhe As Integer
  Dim szEhe As String
  Dim szAnfr As String
' Fragen wir erst einmal nach der Anzahl der Ehen
  iEhe = Val(Anfrage("@Anzahl(Ehe())")) 's. Fußnote Fehler! Textmarke nicht definiert.
  If iEhe <= 1 Then
    Exit Sub
' Hier beginnt die eigentliche Arbeit an dem zu komplexen Szenario
  Else
'   Zunächst werden alle Objektbezeichner von Ehen in die Liste "ObjList" des
'   Fensters "ObjDlg" eingelesen.
    For x = 1 To iEhe
      szAnfr = "@Bezeichner(Ehe();@Pos(" + Trim$(Str$(x)) + "))"
      szEhe = Anfrage(szAnfr)
      ObjDlg.ObjList.AddItem szEhe
    Next x
'   Nun wird das Fenster als Dialog angezeigt
    ObjDlg.Show 1
'   Nach der Benutzereingabe werden die LinkItem-Eigenschaften
'   der Dialogelemente verändert.
'   Falls der Benutzer "Abbrechen" im Dialog gewählt hat,
'   wird der ListIndex der Liste vom Dialog auf -1 gesetzt.
    If ObjDlg.ObjList.ListIndex >= 0 Then
      szEhe = ObjDlg.ObjList.Text
      EdName.LinkItem = "Nachname(;Ehemann(;$" + szEhe + "))"
'     Hier können noch weitere Anpassungen folgen
      Unload ObjDlg
    Else
'     Das Programm wird abgebrochen,
'     falls der Benutzer "Abbrechen" im Dialog gewählt hat.
      End
    End If
  End If
End Sub
```

Das Verfahren ist etwas aufwendig, da es der intuitiven Gestaltung einer Eingabemaske widerspricht. Angesichts der Tatsache, daß derartig komplexe Sachverhalte kaum zu erwarten sind, wird man hierauf auch verzichten und es bei einem einfachen Warnhinweis an den Benutzer belassen können.

ddd) Dynamische Objektinstanzen

Eine etwas andere Situation liegt dann vor, wenn zum Beispiel mit einer nahezu beliebigen Anzahl von Kindern, Eigentumsgegenständen etc. zu rechnen ist. Für jedes dieser Objekte sollen die entsprechenden Daten erfaßt werden. Hier muß das System auf eine dynamische Verwaltung der Daten angelegt sein. Ein Programm, das in einem solchen Fall eine künstliche Begrenzung etwa auf 5 Kinder, 3 Häuser etc. vornimmt, mag einfach zu entwickeln sein und zudem einen großen Teil der real denkbaren Fälle abdecken. Da jedoch Atlas bewußt eine dynamische Instantiierung von Objekten unterstützt, entspräche ein statisches Vorgehen nicht dem hier beschriebenen System. Zudem sollte der Computer als Hilfsmittel gerade in extremen Sachlagen behilflich sein.

Eine Realisierungsmöglichkeit für eine flexible Verwaltung von Objektdaten ist die dynamische Erzeugung der zugehörigen Dateneingabemasken. Bereits bei der Beschreibung der Benutzeroberfläche wurde davon ausgegangen, daß das Formular aus einer Kette von flexibel aneinandergebundenen Eingabemasken gebildet werden

kann[375]. Durch die Anlage eines sogenannten Arrays (d.h. einer Liste) können nun mehrere Masken gleichen Typs instantiiert werden. Sie werden dann wie in einer gewöhnlichen Liste über Ordnungsziffern angesprochen. Im folgenden Beispiel wird eine Maske *Kind* beschrieben, die die Eingabefelder `EdNameKind`, `EdVornameKind` sowie `EdGebDatKind` enthält. Beim Laden einer neuen Instanz der Maske werden die LinkItem-Eigenschaften der Kontrollfelder angepaßt:

```
Sub KinderEinrichten()
' Variablendeklaration
  Dim iKinder As Integer
  Dim szKind As String
' Zunächst wird die Anzahl der Kinder abgefragt
  iKinder = Val(Anfrage("@Anzahl(Kind(;;))")
  If iKinder = 0 Then Exit Sub
' Für jedes Kind werden nun die Kontrollelemente angelegt.
  For x = 1 To iKinder
'   Laden der Elemente.
    Load EdNameKind(x)
    Load EdVornameKind(x)
    Load EdGebDatKind(x)
'   Ändern der LinkItem-Eigenschaft.
'   Der Bezeichner des Kindes wird einmal gebaut.
'   Beispiel für x = 1: Kind(;;;@Pos(1))
    szKind = "Kind(;;;@Pos(" + Trunc$(Str$(x)) + "))"
    EdNameKind(x).LinkItem = "Nachname(;" + szKind + ")"
    EdVornameKind(x).LinkItem = "Vorname(;" + szKind + ")"
    EdGebDatKind(x).LinkItem = "Geburtstag(;" + szKind + ")"
'   Die Kontrollelemente müssen nun nach Bedarf angezeigt und
'   positioniert werden.
  Next x
End Sub
```

Die LinkItem–Eigenschaft wird also weiterhin allgemein gehalten. Über den Parameter `@Pos()` wird das jeweils für ein Eingabeelement korrekte Kind angesprochen. Das Programm muß spätestens vor der ersten Datenübermittlung die entsprechenden Beziehungen $y = Kind(v, m)$ anlegen.

f) Zeitpunkt der Aktualisierung

In einer Multitasking-Umgebung kann der Benutzer, wie bereits mehrfach festgestellt, jederzeit in eine andere Anwendung wechseln und aus dieser zurückkehren. Die zweite Anwendung ist dabei in der Lage, Daten so zu verändern, daß sich auch für die erste Anwendung die Ausgangslage ändert. Wird beispielsweise in dem Aufnahmeformular nach dem Nettogehalt einer Person gefragt, so wird der Benutzer womöglich in eine Anwendung wechseln wollen, die eine Brutto-Netto-Berechnung durchführt. Kehrt er nach der Berechnung wieder in das Aufnahmeformular zurück, so liegt der Nettowert nun vor und soll in das Formular automatisch eingetragen werden.

Das Aufnahmeformular muß spätestens vor dem Wechsel in ein anderes Programm die erfaßten Fakten an Atlas übermitteln, damit die Brutto-Netto-Berechnung auf diesen Grundlagen erfolgt. Dies kann an mehreren Stellen geschehen.

1. Das Faktum eines Eingabeelements kann nach jeder Änderung des Inhaltes an Atlas übermittelt werden.

---

[375] S. 57.

2. Das Faktum eines Eingabeelements kann immer dann übermittelt werden, wenn das Element den Fokus verliert. (Ein Kontrollelement verliert den *Fokus* sobald der Benutzer ein anderes Kontrollelement aktiviert.)
3. Die Fakten der gesamten Anwendung können immer dann übermittelt werden, wenn die Anwendung deaktiviert oder beendet wird. Eine Übermittlung geschieht dann jedenfalls noch bevor eine andere Anwendung vom Benutzer aktiviert wird, um eventuell dieselben Daten weiterzuverarbeiten.
4. Eine Übermittlung kann auf Anweisung des Benutzers erfolgen.

Die Entscheidung für eine der Varianten muß die Tatsache berücksichtigen, daß Atlas alle übermittelten Daten speichert und nicht die vorhergehenden Daten überschreibt. Insbesondere Variante 1. scheidet aus diesem Grunde aus. Würde der Benutzer etwa einen Namen mit zehn Buchstaben eingeben, so würden zehn Zwischenwerte übermittelt und abgespeichert werden. Auch die zweite Variante scheint noch zu fehleranfällig. Die oft sehr vorläufigen Eingaben würden zu früh gespeichert werden und so die Datenmenge aufblähen. Vielmehr ist eine Kombination aus den Varianten 3. und 4. anzustreben. Der Benutzer muß eine Datenübertragung jedenfalls von Hand auslösen können. Aus den oben geschilderten Gründen sollte die Anwendung auch bei einem Umschalten des Benutzers auf eine andere Anwendung die Daten an Atlas übermitteln und sie so anderen Anwendungen verfügbar machen. Der Benutzer sollte jedoch auch die Möglichkeit haben, eine Übertragung der Daten beim Wechsel zu einer anderen Anwendung zu verhindern, etwa indem er die automatische Übermittlung abschaltet oder bei jedem Wechsel eigens gefragt wird. Meldet sich beispielsweise sein Terminkalender, so verliert die aktuelle Anwendung den Fokus und übermittelt vorher die Daten selbst dann, wenn der Benutzer sich gerade mitten in einem Eingabevorgang befindet.

Für die Aktualisierung der Daten eines Programms aus den Daten von Atlas gelten ähnliche Gesichtspunkte. Dieser kann prinzipiell zu folgenden Zeitpunkten geschehen:

1. beim Programmstart,
2. beim Aktivieren des vorher deaktiven Programms,
3. sobald ein Eingabeelement den Fokus erhält,
4. auf Anweisung des Benutzers,
5. in bestimmten zeitlichen Intervallen.

Jede der Varianten erscheint prinzipiell sinnvoll. Lediglich die Variante 3. ist, sofern es der einzige Aktualisierungszeitpunkt ist, zu spät. Sie würde nicht garantieren, daß das Arbeitsblatt jemals den eigentlichen Sachstand anzeigt. Die Variante 5. ist angenehmer Luxus. Sie ermöglicht es, die Inhalte der Eingabemaske auch dann aktuell zu halten, wenn das Programm gar nicht aktiviert ist.

Zu bedenken ist, daß bei jeder Aktualisierung der Daten aus dem System eine zu komplexe Sachlage festgestellt werden kann. So kann (in einem freilich wenig realistischen Fall) eine andere Anwendung eine weitere Ehe ins Spiel bringen, mit der das ebenfalls gerade verwendete Aufnahmeformular nicht arbeiten kann. Das Programm muß hierfür eine Verhaltensweise vorsehen, wie sie bereits im vorangegangenen Abschnitt beschrieben wurde.

g) Vermeidung von Dubletten

Bei der Übermittlung an Atlas werden alle Daten vom System erneut abgespeichert. Dieses Verfahren kann dazu führen, daß unerwünschte Dubletten der Daten auftreten. Ein Datum ist dann eine unerwünschte Dublette, wenn es sich in keiner wesentlichen Aussage von einem anderen Datum unterscheidet. Insbesondere liegt dann **keine** Dublette vor, wenn sich auch nur eine der Zusatzinformationen wie Quelle,

Eingabezeitpunkt etc. voneinander unterscheiden. Eine Dublette kann man deshalb leicht vermeiden, indem man, bevor man ein Faktum an das System übermittelt, das System nach genau der verwendeten Beschreibung fragt und das erfragte und das zu übermittelnde Faktum vergleicht. Die folgende Funktion übermittelt den Wert eines beliebigen Kontrollelements an das System unter Vermeidung einer Dublette:

```
Sub ÜbermittleElement (cControl As Control)
' Es werden keine leeren Inhalte übermittelt:
  If cControl = "" Then Exit Sub
' Die Routine verwendet ein Dummy-Element, um die Kontrollabfrage
' durchzuführen. Zunächst wird das LinkItem des Ursprungselements
' übernommen und danach gefragt.
  StDDE.LinkItem = cControl.LinkItem
  StDDE.LinkMode = 2
  StDDE.LinkRequest
  StDDE.LinkMode = 0
' Falls der Inhalt nicht mit dem erfragten Wert übereinstimmt,
' wird der neue Wert übermittelt.
  If StDDE <> cControl Then
    cControl.LinkMode = 2
    cControl.LinkPoke
'   Durch die sofortige Nachfrage nach dem übermittelten Wert
'   wird dafür gesorgt, daß das Kontrollelement den Wert nun im
'   Atlas-Standardformat enthält!
    cControl.LinkRequest
    cControl.LinkMode = 0
  End If
End Sub
```

Eine Funktion, die die Inhalte aller Elemente eines Aufnahmebogens mit dieser Technik aktualisiert, sieht etwa folgendermaßen aus:

```
Sub ÜbermittleDatenAnAtlas()
  For x = 0 to Me.Controls.Count - 1
    If Me.Controls(x).LinkItem <> "" And Me.Controls(x).LinkTopic <> "" Then
      ÜbermittleElement Me.Controls(x)
    End If
  Next
End Sub
```

Selbstverständlich muß die Anwendung auch hier zunächst alle Beziehungen des Grundszenarios an Atlas übermittelt haben.

h) Einsatz von Filtern

Im vorangegangenen Abschnitt wurde eine Dublette eines Faktums damit definiert, daß sich der beschriebene Wert und die entsprechenden Zusatzinformationen nur unwesentlich voneinander unterscheiden. Unter den Begriff *unwesentlich* fallen dabei lediglich geringfügige Abweichungen im Eingabezeitpunkt. Wenn man an dieser Stelle keinen Spielraum zuläßt, so gibt es de facto gar keine Dubletten. Weichen hingegen der Wert zweier Fakten, dessen Geltungszeitraum oder die Quelle der Information voneinander ab, so stellen diese keine Dubletten dar. Die Unterscheidung derartiger Informationen erfolgt über Ein- und Ausgabefilter[376].

[376] S. S. 97 ff, S. 133 ff.

Das Programm übernimmt dabei die Werte für die Filter aus dem Aufnahmebogen selbst. Da der Aufnahmebogen immer zur Mandatsaufnahme ausgefüllt wird, ist die Informationsquelle regelmäßig der Mandant. Der Mandant wird über ein Eingabefeld erfaßt. Die Information für den Geltungszeitraum der Beziehung *Kind* ergibt sich aus den jeweiligen Geburtsdaten. Der Geltungszeitraum der Ehe aus dem Datum der Eheschließung etc. Zu differenzieren sind die beiden Filtertypen. Der Ausgabefilter, also der Filter, der für das Abfragen von Fakten gesetzt wird, beschränkt sinnvollerweise nur die Eingabequelle. Er bewirkt so, daß lediglich die Aussagen des Mandanten im Formular angezeigt werden. Im Zeitpunkt der Mandatsaufnahme kann selbstverständlich auch dieser Filter nicht gesetzt werden. Das Setzen des Ausgabefilters erfolgt unmittelbar vor der Aktualisierung der Fakten im Formular aus dem System. Der folgende Programmausschnitt zeigt eine Filtereinstellung in einer Aktualisierungsfunktion:

```
Sub AktualisiereDatenAusAtlas()
  Übermittlung "@Ausgabefilter", ";Mandant()"
  ...
End Sub
```

Der Eingabefilter steuert hingegen den Geltungszeitraum der an Atlas übermittelten Fakten. Er wird also für die entsprechenden Fakten vor der Übermittlung immer wieder zu ändern sein. Wie bereits beschrieben gruppiert das Programm bestimmte Eingabeelemente nach Sinnzusammenhängen. In der Regel sind es auch diese Gruppen, die jeweils mit einem gemeinsamen Eingabefilter abgehandelt werden können. Das folgende Programmbeispiel verwendet einen Trick, indem es in die Property *Tag* eines Gruppenelements jeweils die Beschreibung des neuen Eingabefilters einstellt. (Diese Property wird von BASIC nicht interpretiert. Sie kann für Zwecke wie den hier gezeigten frei vom Programm genutzt werden.) Auf diese Weise kann der Eingabefilter bereits beim Entwurf einer Maske und nicht erst bei der Programmierung definiert werden. Voraussetzung ist jedoch die exakte Reihenfolge der Kontrollelemente. Der folgende Ausschnitt zeigt eine Modifikation des bereits dargestellten Skripts[377] zur Gestaltung des Fensters:

```
Begin Form HauptForm
  Caption = "Aufnahmebogen"
...
  Begin Frame GrpPerson
    Caption = "Ehemann"
' NEU:  Eingabefilter für die nachfolgenden Elemente. Alle weiteren Fakten gelten ab
'    Geburtstag des Ehemanns
    Tag = "Geburtstag(;Ehemann());;Mandant();"
' Ende NEU
    Height = 2055
    Index = 0
    Left = 120
    TabIndex = 0
    Top = 120
    Width = 4335
...
  End
End
```

[377] S. S. 178.

Die folgende Funktion enthält lediglich eine geringfügige Modifikation zu einer bereits gezeigten Funktion, die alle Fakten eines Formulars an Atlas überträgt (s.S. 188):

```
Sub ÜbermittleDatenAnAtlas()
' NEU:  Der Ausgabefilter bestimmt auch die Auswertung der Ausdrücke
'     des Eingabefilters[378]. Er wird hier gesetzt,
'     da ihn eine andere Anwendung womöglich verstellt hat.
   Übermittlung "@Ausgabefilter", ";Mandant()"
' Ende NEU
   For x = 0 to Me.Controls.Count - 1
' NEU      Hier wird überprüft, ob die Tag-Property belegt ist.
'        Prinzipiell kann jedes Element einen eigenen Filter besitzen!
      If Me.Controls(x).Tag <> "" Then
'           Falls es eine Tag-Property gibt,
'           wird diese als Eingabefilter interpretiert und übermittelt.
         Übermittlung "@Eingabefilter", Me.Controls(x).Tag
      End If
' Ende NEU
'     Übermitteln der Fakten
      If Me.Controls(x).LinkItem <> "" And Me.Controls(x).LinkTopic <> "" Then
         ÜbermittleElement Me.Controls(x)
      End If
   Next
End Sub
```

Das Formular beherrscht nun die wesentlichen Funktionen, um sinnvoll Fakten für eine spätere Weiterverarbeitung zu erfassen. Es kommuniziert zudem über definierte Aktualisierungen der Fakten mit anderen Programmen. Diese erfolgen, sofern der Benutzer nichts anderes vorgibt, beim Aktivieren (Lesen aus Atlas) und Deaktivieren (Schreiben in Atlas) der Anwendung. Dubletten werden dabei vermieden. Die Filter werden für Gruppen von Fakten unterschiedlich gesetzt.

Die Problematik des Aufnahmebogens liegt in der Ausgewogenheit des zugrundeliegenden Szenarios. Ist es zu einfach, wird es den Ansprüchen an eine EDV-Unterstützung nicht gerecht. Ist es zu komplex, wird die Formulardarstellung für den Benutzer unbrauchbar.

2. Fallskizze

Mit der Anwendung *Aufnahmebogen* wurde eine Möglichkeit vorgestellt, Fakten nach einem in der Praxis bekannten Schema aufzunehmen. Die verhältnismäßig starre Eingabetechnik läßt jedoch nur begrenzte vorab kodierte Szenarien zu. Dieser Abschnitt beschreibt nun ein Programm, das speziell auf die Eingabe von sehr individuellen und komplexen Szenarien zugeschnitten ist.

a) Konzept

Die Anwendung *Fallskizze* geht von einer graphischen Erfassung von Sachverhalten aus. Sie versucht sich dabei an Graphiken anzulehnen, wie sie Juristen typischer Weise anfertigen, um sich komplexe Sachverhalte zu verdeutlichen[379]. Gleichzeitig lehnt sich die Fallskizze stark an die interne Repräsentationsform der Daten an, so daß sich praktisch das gesamte Wissen der Datenbasis mit Hilfe dieser Anwendung optisch repräsentieren läßt.

---

[378] S. S. 153 ff.

[379] Vgl. z.B. *Brühl*, S. 101; *Tettinger*, S. 89 f.; *Haft*, S. 95 ff.

Eine Fallskizze ist wie das Datenmodell aus |Objekten und Beziehungen zwischen Objekten aufgebaut. Die Fallskizze gibt dem Benutzer die Möglichkeit, beliebig Objekte zu instantiieren oder auf instantiierte Objekte zuzugreifen. Weiterhin können beliebige Beziehungen zwischen den Objekten eingerichtet werden. Das folgende Schaubild zeigt die Darstellung einer Ehe von Peter und Tina:

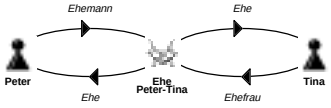


Abbildung 43: Darstellung eines Szenarios *Ehe* in der Anwendung *Fallskizze*.

Dargestellt werden die Objekte *Peter*, *Tina* und *Ehe Peter–Tina*. Die Visualisierung erfolgt über Symbole wie Spielfiguren für Personen oder einen Handschlag für Verträge etc. Weiterhin werden durch Pfeile die Beziehung *Ehe Peter-Tina* $= Ehe(Peter, Tina)$ sowie die komplementären Beziehungen *Peter* $= Ehemann(Ehe\ Peter\text{-}Tina)$ und *Tina* $= Ehefrau(Ehe\ Peter\text{-}Tina)$ dargestellt. Dabei zeigt der Pfeil immer vom Hauptobjekt zu den jeweiligen weiteren Objekten einer Beziehung.

aa) Typisierte Objekte

Bereits die Wahl der Symbole zeigt an, daß zwischen mehreren Typen von Objekten unterschieden wird. Dies geschieht zunächst aus Gründen der Übersichtlichkeit. So muß eine Person von einem Vertrag oder einem anderen gedanklichen Objekt unterscheidbar sein, damit die Skizze ausreichend aussagekräftig ist.

Das System Atlas kennt hingegen keine Typisierung von Objekten. Es erlaubt prinzipiell sogar kollidierende Eigenschaften. So können einem Objekt gleichzeitig die Eigenschaften *Mensch*, *Tier* und etwa *Auto* zugewiesen werden, ohne daß das System die tatsächliche Unvereinbarkeit moniert. Lediglich über zusätzliche Erweiterungsmodule ist es denkbar, derartige Plausibilitätskontrollen einzuführen. Der Grund für die hohe Flexibilität des Grundsystems liegt darin, daß prinzipiell jede Plausibilitätskontrolle auch eine juristische Regel in sich birgt. Juristische Regeln sind ihrerseits Veränderungen unterworfen, die ein System flexibel abbilden muß. So ist es heute ausgeschlossen, daß ein Mensch gleichzeitig eine Sache ist. Rechtssysteme, die den Sklavenhandel kannten, mögen diese Regel $\forall x(Mensch(x) \rightarrow \neg Sache(x))$ nicht für selbstverständlich angesehen haben. Die Überprüfung von Kollisionen der dargestellten Art überläßt Atlas den zugreifenden Programmen. Dabei ist es durchaus denkbar, ein Programm zu entwickeln, das zu definierten Ereignissen alle Fakten auf Plausibilität prüft und dem Anwender entsprechende Fehler mit Korrekturvorschlägen vorlegt.

Die Anwendung *Fallskizze* verwendet Eigenschaften, um Objekte zu typisieren bzw. den Typ eines Objekts zu erkennen. Instantiiert der Benutzer ein neues Objekt, so wählt er zunächst aus einer Palette von Objekttypen einen Typ aus. Dem instantiierten Objekt wird sofort eine dem Typus entsprechende Eigenschaft wie *Mensch*, *Sache* etc. zugeordnet. Ist ein Objekt nicht typisierbar, weil etwa die entsprechende Eigenschaft gerade im Streit steht, so stellt das Programm ein typenloses Objekt zur Verfügung, das lediglich die systeminterne Eigenschaft *OBJEKT* besitzt.

bb) Stammdaten

Mit einem Objekttypus wird ein bestimmter Satz von Eigenschaften und Beziehungen verknüpft. Sie verkörpern die Stammdaten eines Objekts. Bei Personen sind dies etwa Angaben wie Name, Vorname oder Geburtsdatum. Bei Sachen kann dies der Wert, eine Zuordnung zu einer Gattung etc. sein. Bei Verträgen wiederum ist es der Zeitpunkt des Vertragsschlusses oder des Vertragsablaufs. Instantiiert der Benutzer ein neues Objekt in der Fallskizze, so erhält er die Möglichkeit, diese

Stammdaten in einer Maske einzugeben. Während der Arbeit, kann er diese Stammdaten jederzeit ändern.

Weiterhin kann der Anwender angeben, ob das instantiierte Objekt auch tatsächlich existiert. Denkbar ist es, auch Objekte in eine Fallskizze aufzunehmen, die zumindest von einem Beteiligten bestritten werden.

cc) Beziehungen

Das Programm bietet die Möglichkeit, instantiierten Objekten neben den Stammdaten beliebige Beziehungen zueinander zuzuweisen. Hierzu markiert der Benutzer die an einer Beziehung beteiligten Objekte. Sodann wählt er aus einer Liste aller in dem konkreten System installierten Relationen die gewünschte aus. Er kann nun die Relationsparameter mit den markierten Objekten belegen.

Neben der gewählten Relation legt das Programm automatisch die entsprechenden komplementären Beziehungen an. Der Benutzer instantiiert beispielsweise die Objekte *Peter*, *Tina* und *Ehe Peter-Tina*. Er wählt die Relation *ehe = Ehe*(*ehemann, ehefrau*) und instantiiert hiermit die konkrete Beziehung *Ehe Peter-Tina = Ehe*(*Peter, Tina*). Die Anwendung *Fallskizze* übernimmt nun die Instantiierung der komplementären Beziehungen *Peter = Ehemann*(*Ehe Peter-Tina*) sowie *Tina = Ehefrau*(*Ehe Peter-Tina*) automatisch.

b) Realisation

aa) Oberfläche

Die folgende Abbildung 44zeigt das Schema der Oberflächen einer Fallskizzenanwendung:

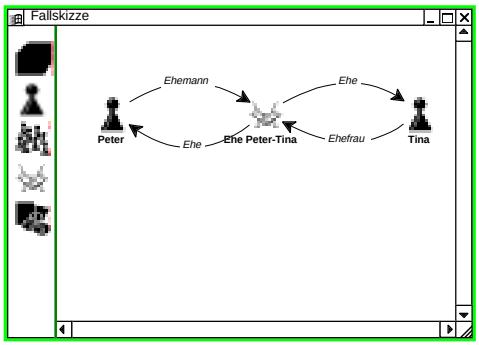


Abbildung 44: Schema für die Oberfläche des Programms *Fallskizze*.

Ähnlich wie bei einem Standardgraphikprogramm[380] besteht die Oberfläche aus einer Werkzeugleiste (links im Bild) und einem Zeichenbereich [381]. Die Werkzeugleiste enthält eine Liste von Symbolen. Jedes Symbol repräsentiert einen definierten Objekttyp. Um ein neues Objekt auf der Zeichenfläche zu plazieren, nimmt der Benutzer ein Objektsymbol aus der Werkzeugleiste und zieht es auf die Zeichenfläche. Hier legt er das Objekt an einer beliebigen Position ab. Die Position und das verwendete Symbol kann er später ändern.

---

[380] Die Fallskizzen wurden hier mit einem Graphikprogramm zur Erstellung von Flußdiagrammen (Corel FLOW) entworfen. Programme dieser Art arbeiten nach einem ähnlichen Prinzip wie die vorgestellte Fallskizze. Sie erlauben das Verbinden von beschrifteten Symbolen mit beschrifteten Pfeilen. Der Unterschied liegt darin, daß die Programme nur graphische Informationen verwalten. In einigen Fällen wird aus den Graphiken ein Programmcode generiert oder eine Datenbank entworfen (Software Engineering).

[381] Auf die Darstellung des Menüs wurde verzichtet.

bb) Anlage von Objekten

Sobald der Benutzer ein Objekt plaziert hat, erhält er ein Dialogfeld folgender Art:

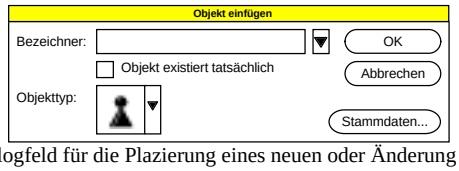


Abbildung 45: Dialogfeld für die Plazierung eines neuen oder Änderung eines vorhandenen Objektsymbols.

Der Benutzer erhält eine Liste der in einem Fall bereits instantiierten Objekte. Er kann hieraus eines auswählen oder einen noch unbekannten Objektbezeichner eingeben. Im letzten Fall wird ein neues Objekt instantiiert. Weiterhin kann der Benutzer eingeben, ob das Objekt tatsächlich existiert oder die Existenz bezweifelt wird. Hinzu kommt die Möglichkeit, den Objekttypus durch die Wahl eines Symbols zu ändern.

Zuletzt bietet das Dialogfeld die Option, die Stammdaten des Objekts in einer weiteren Maske zu verändern. Die Maske hängt von dem angegebenen Objekttyp ab. Sofern das Objekt neu instantiiert wird, zeigt das Programm die Stammdatenmaske automatisch nach Verlassen des Dialogfeldes an. Das Programm bietet jederzeit die Möglichkeit, die Grunddaten wie Typus, zugehöriges Objekt und dessen Existenz für ein plaziertes Objektsymbol zu ändern.

cc) Kalender

Eine alternative Möglichkeit, Objekte zu instantiieren bietet der Kalender. Er wird als *Ereignisübersicht* bezeichnet[382]. Die Ereignisübersicht zeigt eine Liste aller Objekte, die einen Datumswert besitzen (Ereignisse). Die Liste wird nach diesem Wert sortiert. Das entsprechende Dialogfeld bietet die Möglichkeit, neue Ereignisse hinzuzufügen. Hierdurch werden neue Objekte instantiiert, denen sofort ein Datumswert zugewiesen wird. Das Abbild zeigt ein Dialogfeld *Ereignisse*, das diese Funktionalität besitzt:

Abbildung 46: Das Dialogfeld *Ereignisse*.

Das Dialogfeld erlaubt es, Daten von Ereignissen zu ändern und neue Ereignisse hinzuzufügen. Die angezeigten Daten entsprechen dem eingestellten Filter. Innerhalb des Filters wird von Atlas automatisch der zuletzt angegebene Wert übermittelt. Dieser wird auch den Angaben der Fallskizze zugrundegelegt. Bei der Änderung eines Datums wird, wie in Atlas nicht anders vorgesehen, dem Objekt ein neuer Datumswert hinzugefügt.

[382] S. z.B. *Tettinger*, S. 89; *Schumacher*, S.12.

dd) Eingabe von Beziehungen

Die Anwendung ermöglicht es, mehrere Objekte auf der Zeichenfläche zu markieren und zwischen Ihnen mit einer Relation eine Beziehung herzustellen. Hierzu dient folgendes Dialogfeld:

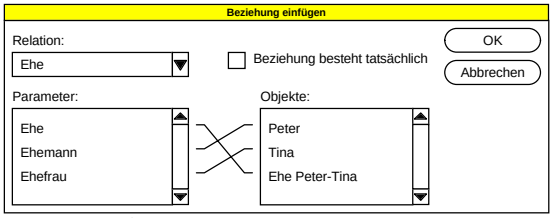


Abbildung 47: Dialogfeld zum Editieren von Beziehungen zwischen Objekten.

In der aufklappbaren Liste *Relation* wählt der Benutzer zunächst die Relation, die der Beziehung zugrunde liegen soll. Das Programm zeigt in dieser Liste alle Relationen an, die auf dem konkreten System in Atlas zur Verfügung stehen. Dies führt zu einer erheblich höheren Flexibilität als sie bei Programmen gegeben ist, die von festen Szenarien ausgehen.

In der Liste *Parameter* werden jeweils die Parameter der gewählten Relation angezeigt. Der Name der Relation wird als erster Parameter selbst auch in die Liste eingestellt. Die Liste *Objekte* enthält alle auf der Zeichenfläche markierten Objekte. Durch Markieren eines Parameters in der linken Liste und nachfolgend eines Objekts in der rechten Liste wird das Objekt als Belegung des Parameters angegeben. Die Belegungen müssen eineindeutig erfolgen. Jeder Parameter muß belegt sein. Mit der Bestätigung der Eingaben wird die Beziehung instantiiert. Zudem prüft das System die korrespondierenden Relationen (im Beispiel sind dies *Ehemann* und *Ehefrau*). Sind diese Relationen Eigenschaften, so werden diese Eigenschaften den entsprechenden Objekten (*Peter* und *Tina*) zugewiesen. Sind es wie im Beispiel Relationen mit einem weiteren Parameter **und** ist dieser Parameter wiederum die Ausgangsrelation (im Beispiel *Ehe*), so werden die komplementären Beziehungen angelegt und somit das Gesamtszenario vervollständigt.

Im Graphikfeld werden die neuen Beziehungen durch Pfeile dargestellt.

ee) Filter und Legenden[383]

Eine interessante Möglichkeit bietet eine Fallskizze dann, wenn Sie ein- und denselben Sachverhalt in einer Skizze aus unterschiedlichen Perspektiven darstellt. Sie kann dabei sowohl den zeitlichen Ablauf eines Geschehens verdeutlichen als auch die unterschiedlichen Vorträge der Beteiligten. Dies kann durch unterschiedlich farbige Linien oder Hintergründe von Symbolen erfolgen. Diese Darstellungsvarianten werden letztlich spezifischen Atlas-Filtern zugewiesen. Eine derartige Definition soll hier als *Sicht* bezeichnet werden. Das folgende Abbild zeigt ein Dialogfeld, das die Bearbeitung einer solchen Sicht ermöglicht:

[383] Diese Option ist im Beispiel nur sehr rudimentär realisiert.

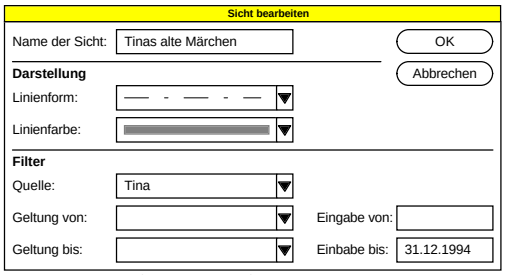


Abbildung 48: Dialogfeld zur Konfiguration unterschiedlicher Sichten.

Jede Sicht erhält einen Namen. Die Darstellungsform der Sicht wird aus einer Liste ausgewählt. Für die Bestimmung des entsprechenden Atlas-Ausgabefilters werden in der Liste *Quelle* alle verfügbaren Objekte und in den Listen *Geltung von* und *Geltung bis* die verfügbaren Ereignisse (das sind Objekte mit Datumswert) angeboten. Der Eingabezeitraum wird im Klartext eingegeben.

Mehrere Sichten werden zunächst in einer Skizze zusammengefaßt. Die Skizze erhält eine Legende, um die Farbbedeutungen klarzustellen. Die Zusammenfassung mehrerer Sichten soll hier als Sichtrahmen[384] bezeichnet werden. Ein Sichtrahmen kann wiederum einen Namen erhalten. Durch eine zeitgesteuerte Abfolge derartiger Rahmen können Prozesse dargestellt werden. Hierbei kann zum einen der zeitliche Ablauf des Sachverhaltes selbst, aber auch die zeitliche Wandlung der Sachverhaltsvorträge der Beteiligten dargestellt werden. Eine Gegenüberstellung zweier Sichtrahmen ermöglicht etwa den Vergleich unterschiedlicher Ansichten oder Parteivorträge.

ff) Assistent[385]

Die Komplexität des beschriebenen Programms sowie der denkbaren Sachverhalte macht die Implementierung von Modulen sinnvoll, die bestimmte immer wieder auftretende Prozesse durch eine Automatik unterstützen.

aaa) Automatische Sachverhaltsskizze

Eine Hilfe stellt eine Automatisierung einer Sachverhaltsskizze dar. Angenommen, der Anwender hat für eine Ehesache bereits den beschriebenen Aufnahmebogen ausgefüllt. Im Zuge dieser Tätigkeit wurden bereits mehrere Objekte instantiiert und zwischen diesen wurden Beziehungen hergestellt.

Ein einfacher Assistent bietet beispielsweise die Möglichkeit, daß er alle Objekte eines Sachverhalts nach vorgegebenen Regeln auf der Zeichenfläche anordnet und die Beziehungen zwischen ihnen darstellt. Er kann die Objekte sortiert nach ihrem Bezeichner, ihrem Auftreten in der Datenbasis oder etwa ihrem Wert wie Kacheln von links oben nach rechts unten auf der Zeichenfläche verteilen.

Ein intelligenterer Assistent erkennt dagegen das Hauptszenario des Falls oder fragt danach bei dem Benutzer. Er stellt dann nur die für das Szenario typischerweise interessanten Beziehungen dar und plaziert hierzu die benötigten Objekte in einer für das Szenario typischen Darstellung. Im Grundszenario *Ehe* wird der Assistent zum Beispiel die Objekte, die die Relationen *Ehemann*, *Ehe* sowie *Ehefrau* repräsentieren, in dieser Reihenfolge in der ersten Zeile und alle Objekte für die Relation

---

[384] Der englische Ausdruck Frame (Rahmen) wird insbesondere in der Videotechnik verwendet. Er bezeichnet ein Bild eines Videos. Dieser Ausdruck ist jedoch im bereiche der künstlichen Intelligenz anders belegt.

[385] Im Beispielsprogramm nicht implementiert.

eheliches_Kind in der zweiten Zeile auf der Breite der Fläche verteilen. Dieses Verfahren erlaubt nicht nur eine optisch ansprechendere Plazierung der Objekte, es führt aufgrund der Selektion auch zu einer besseren Übersicht. Es bleibt dem Benutzer letztlich immer noch überlassen, weitere Objekte und Beziehungen hinzuzufügen.

bbb) Automatische Generierung von Sichten

Die Darstellung der zeitlichen Abfolge eines Geschehens aus unterschiedlichen Sichten kann bei komplexen Sachverhalten sehr aufwendig werden. Ein Assistent hilft hier, indem er für jedes Ereignis, das in einem Filter als Geltungsgrenze angegeben ist, und jeden Beteiligten, der als Quelle einer Information angegeben wird, entsprechende Sichten oder Sichtrahmen erstellt. Ebenso übernimmt er die Anordnung der Sichten in Sichtrahmen sowie Sequenzen. Auf diese Weise kann sich der Anwender schnell einen Überblick über die Vorgänge und Zeitabläufe innerhalb eines Sachverhalts verschaffen.

c) Realisierung

Eine Fallskizzenanwendung kann mit unterschiedlichen Entwicklungsumgebungen realisiert werden. Die Beispielanwendung[386] wurde in Pascal programmiert. Sie genießt den Vorteil einer offenen DLL-Schnittstelle, die die dynamische Einbindung von Objekttypen erlaubt. Die Schnittstelle enthält Funktionen zur Anzeige des Objektsymbols, zur Anlage einer dem Typus zugeordneten Eigenschaft sowie zur Anzeige und Ausführung der Stammdatenmaske. Auf diese Weise können für unterschiedliche Arbeitsplätze die verschiedensten Objekttypen entwickelt werden. So mag an einem allgemeinen Arbeitsplatz ein Objekttyp *Sache* für alle Belange ausreichen. Besondere Eigenschaften werden von Hand nachgetragen. An einem Arbeitsplatz in einem Notariat wird hingegen ein eigener Typ *Grundstück*, *Wohnung Haus* oder *bewegliche Sache* vernünftig sein, da hier die typspezifischen Stammdaten meist relevant sind und deshalb optimal sofort erfaßt werden. Ein anderer Arbeitsplatz unterscheidet etwa zwischen unterschiedlichen Automobilklassen etc.

Im folgenden sollen die grundsätzlichen Merkmale in einem hypothetischen Programm beschrieben werden und zwar wie bisher auch in Visual–BASIC[387].

aa) Datenmodell

Die Anwendung greift teilweise auf Daten aus dem System Atlas zu, teilweise verwendet sie aber darüber hinausgehende Informationen. Entsprechend speichert die Anwendung Fallskizzen primär in einem eigenen Datenformat, das hier jedoch nicht detailliert zu beschreiben ist. Die Skizzendatei enthält voran einen Verweis auf die der Skizze zugrundeliegenden Akte. Hierher werden auch alle Informationen übermittelt, die von allgemeinem juristischen Interesse sind. Dazu gehören etwa die Existenz von Objekten, ihre Eigenschaften, Beziehungen und Werte.

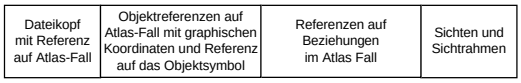
Abstrahiert davon werden alle Informationen, die sich nicht aus den Angaben des Systems Atlas ermitteln lassen. Mithin sind dies Informationen, die über die reinen Fakten des Falls hinausgehen. Hierzu gehören die Informationen, welche Objekte an welcher Position im Zeichenbereich dargestellt wurden. Weiterhin wird gesondert gespeichert, welche Beziehungen zwischen den Objekten dargestellt und welche weggelassen werden. Ebenso gehört zu den Programminterna die Information, mit welchem Objektsymbol ein Objekt dargestellt wird. Diese Ablage von Daten in

---

[386] S. S. 232.

[387] Ein funktional ähnliches Programm ist die Objektansicht *ATLEXPL.EXE*, die in Visual–BASIC 4.0–32 entwickelt wurde.

einer separaten Datei führt einerseits zu Dubletten, andererseits wie jede Dublette zur Gefahr von Widersprüchen zwischen der Information in Atlas und derjenigen in der Fallskizze. Sie ist dennoch notwendig, da nicht alle Fakten aus Atlas in der Fallskizze ausgewertet werden und somit Angaben über die Selektion getrennt abgelegt werden müssen. Das folgende Schaubild zeigt ein Schema einer Datei für eine Fallskizze:



Um Dubletten gering zu halten, wird bei der Speicherung von Objekten und Beziehungen lediglich mit Verweisen auf die entsprechenden Informationen in Atlas gearbeitet.

bb) Überprüfung der vorausgesetzten Relationen

Für die Typisierung von Objekten und die Erfassung ihrer Stammdaten setzt das Programm bestimmte Relationen in der globalen Atlas-Datenbasis voraus. Für die Darstellung eines Menschen wird beispielsweise die Relation $x = Mensch()$ vorausgesetzt. Für dessen Stammdaten werden etwa die Relationen $x = Geburtstag(m)$ oder $x = Nachname(m)$ etc. gefordert. Atlas benötigt diese Relationen zwar nicht, um eine Fallskizze anzufertigen, es kann jedoch in Ermangelung dieser Relationen nicht alle erfaßten, juristisch relevanten Fakten an das System übermitteln.

Es gibt mehrere Möglichkeiten, auf das Fehlen solcher Relationen zu reagieren. So könnte das Setup–Programm bereits bei der Installation die notwendigen Relationen in der Atlas–Datenbasis anlegen[388]. Sofern der Benutzer nicht bewußt an der Datenbasis Änderungen vornimmt, kann sich das System dann auf die Existenz der Relationen verlassen. Das Programm könnte auch beim Programmstart die Existenz der entsprechenden Relationen abprüfen. Fehlen Relationen, so könnte es den Programmablauf anhalten oder die entsprechenden Objekttypen oder Stammdatenmasken nicht anzeigen.

Um das Programm einfach und zugleich flexibel zu halten, prüft es zwar beim Programmstart die Existenz der notwendigen Relationen ab, es zeigt dem Benutzer allerdings nur eine Warnung, daß nicht alle erfaßten Daten später in anderen Anwendungen zur Verfügung stehen. Das Programm sperrt zudem Eingabefelder, deren Inhalte in keiner Weise mehr gespeichert werden können. Dies sind regelmäßig keine Informationen, die einen korrekten Programmablauf generell verhindern. Vielmehr können nicht alle Stammdaten erfaßt werden oder die einem Objekttyp zugeordnete Eigenschaft kann nicht an Atlas übermittelt werden.

Eine derartige Prüfung wird zweckmäßigerweise in der Load–Methode des Hauptfensters plaziert. Der folgende Code zeigt beispielhaft eine derartige Passage:

```
Sub Form_Load()
' Generelle Initialisierung des Programms
  Dim iFehler As Integer  ' Wahr, falls Relation fehlt
' Gehen wir davon aus, daß alle Relationen implementiert sind.
  iFehler = False
```

[388] S. S. 168.

```
' Prüfung der Eigenschaft x = Mensch
  If Anfrage("@Relationsindex(Mensch)") = "" Then
    iFehler = True
  End If
' Prüfung der Relation x = Geburtstag(y)
  If Anfrage("@Relationsindex(Geburtstag)") = "" Then
    iFehler = True
'   Sperre die Eingabe für den Geburtstag bei Menschen
'   im Stammdatenfenster "DlgMensch"
    DlgMensch.EdGeburtstag.Enabled = False
  End If
' Prüfung der Relation x = Nachname(y)
  If Anfrage("@Relationsindex(Nachname)") = "" Then
    iFehler = True
    DlgMensch.EdNachname.Enabled = False
  End If
' Prüfung der Relation x = Vorname(y)
  If Anfrage("@Relationsindex(Vorname)") = "" Then
    iFehler = True
    DlgMensch.EdVorname.Enabled = False
  End If
' Weitere Prüfungen
' Weitere Initialisierungen
' Fehlermeldung
  If iFehler Then
    MsgBox "Es stehen nicht alle Fakten in späteren Anwendungen zur Verfügung"
  End If
End Sub
```

cc) Werkzeugleiste

Die Werkzeugleiste wird wie in Visual–BASIC üblich mit Hilfe von dreidimensionalen Radioschaltern realisiert, von denen jeweils genau einer gedrückt (der Wert ist dann TRUE), die übrigen ungedrückt sind. Neben jeweils einem Schalter für jedes Objektsymbol existiert ein Schalter für die Selektion und Verschiebung von Objekten auf der Zeichenfläche. Die Werkzeugleiste entspricht damit der Werkzeugleiste der Visual–BASIC Entwicklungsumgebung selbst.

dd) Positionieren von Objekten

Ein Objekt wird dadurch auf dem Zeichenfeld positioniert, daß der Benutzer zunächst den Objekttypus in der Werkzeugleiste wählt und dann an die entsprechende Position auf dem Zeichenfeld klickt. Auf die Click–Methode des Zeichenfeldes reagiert dieses zunächst durch die Anzeige des Standarddialogfelds zur Objektbearbeitung und Eingabe. Bestätigt der Benutzer die Eingaben in dem Dialogfeld, so wird das Objektsymbol auf dem Zeichenfeld positioniert und gegebenenfalls in Atlas instantiiert. Daraufhin wird das jeweilige Stammdatendialogfeld für den gewählten Objekttyp angezeigt.

Der folgende Code zeigt die Reaktion auf den Mausklick in der Click–Methode des Zeichenfeldes:

```
Sub HauptForm_Click()
  Dim iObjTyp As Integer  '                          Gibt den gewählten Objekttyp an.
  Dim x As Integer  ' Laufvariable
  Dim sObjX As String  '                             Angabe, ob Objekt existiert
```

```
'    Auf das Click-Ereignis wird nur reagiert, wenn einer der Objekttypen
'    ausgewählt ist. Ansonsten wird auf das MouseUp und MouseDown-Ereignis,
'    oder auf das DragDrop-Ereignis reagiert, um Objekte zu verschieben oder zu
'    markieren.
'    ToolBox ist ein Fenster, das die beschriebenen Schalter enthält.
'    BnPfeil ist ein Schalter, der die Markierung von Objekten ermöglicht.
'    BnKreuz ist ein Schalter, der das Verschieben von Objekten ermöglicht.
  If ToolBox.BnPfeil Or ToolBox.BnKreuz Then Exit Sub '                    BnPfeil oder BnKreutz ist gedrückt
'    Zunächst wird geprüft, welcher Objekttyp-Schalter gewählt ist
'    iZObjTyp ist eine Konstante, die die Anzahl der verfügbaren Objekttypen angibt.
  For x = 0 To iZObjTyp - 1
    If ToolBox.BnObjekt(x) Then iObjTyp = x
  Next
    Jetzt wird der Dialog zur Anlage eines Objekts (ObjDlg) angezeigt.
'    Die Initialisierung des Dialoges wird in seiner Load-Methode vorgenommen (s.u.).
  Load ObjektDlg ' Dieser Aufruf hat deklaratorischen Charakter
'    In der Objekttypliste wird der gewählte Objekttyp als Standard voreingestellt.
  ObjektDlg.ObjTypListe.ListIndex = iObjTyp
'    Das Fenster wird modal angezeigt.
  ObjektDlg.Show 1
'    Falls im ObjDlg OK gedrückt wurde, wird in dessen Tag-Eigenschaft der Bezeichner
'    des Objekts abgelegt. Steht hier nichts, hat der Benutzer Abbrechen gedrückt.
  If ObjektDlg.Tag = "" Then Exit Sub
'    Jetzt wird geprüft, ob der Existenzwert des Objekts
‘    mit dem gewählten übereinstimmt.
'    Dies ist insbesondere dann nicht der Fall,
‘    wenn das Objekt noch gar nicht existiert.
'    Stimmt der Existenzwert nicht überein, so wird der neue Wert übermittelt.
'    BnObjX ist eine Auswahlbox, in der der Benutzer angibt, ob ein Objekt existiert.
  sObjX = Anfrage("@Existiert($" + ObjektDlg.Tag + ")")
  If ObjektDlg.BnObjX = 0 And sObjX <> "Nein" Then '                    möglichen Antworten s.S. 138
    Übermitteln "@Existiert($" + ObjektDlg.Tag + ")", "Nein"
  ElseIf ObjektDlg.BnObjX = 1 And sObjX <> "Ja" Then
    Übermitteln "@Existiert($" + ObjektDlg.Tag + ")", "Ja"
  ElseIf ObjektDlg.BnObjX = 2 And sObjX <> "Unklar" Then
    Übermitteln "@Existiert($" + ObjektDlg.Tag + ")", "Unklar"
  End If
'    Nun wird das Objekt auf dem Zeichenfeld positioniert. Zunächst wird das
'    Kontrollelement eingerichtet, das das Symbol enthält.
'    iZObjekte ist eine globale Variable,
‘    die die Anzahl der bereits auf der Zeichenfläche
'    befindlichen Objekte enthält. Diese wird erst hochgezählt.
  iZObjekte = iZObjekte + 1
  Load Objekt(iZObjekte)
'    Der Einfachheit halber wird das Bild des Schalters einfach in das
'    Kontrollelement übernommen.
  Objekt(iZObjekte).Picture = ToolBox.BnObjekt(iObjTyp).PictureUp
'    Nun wird der Bezeichner aus dem Objektdialog entnommen
'    und als Beschriftung des Bildes eingesetzt.
  Objekt(iZObjekte).Caption = ObjektDlg.ObjektListe.Text
'    MausX und MausY sind globale Variablen und enthalten die Koordinaten
'    des Mauszeigers. Mit ihrer Hilfe wird das Objekt positioniert.
  Objekt(iZObjekte).Left = MausX
  Objekt(iZObjekte).Top = MausY
  Objekt(iZObjekte).Visible = True
```

```
'    Zuletzt wird individuell auf den gewählten Objekttyp reagiert.
'    Hierbei wird das entsprechende Stammdatendialogfenster angezeigt.
'    Dieses Fenster erledigt alle notwendigen Aktionen selbst.
'    Es erhält in der Tag-Eigenschaft den Objektbezeichner übergeben.
  Select Case iObjTyp
    Case 0:
'       Ein Objekt des Typs "Ding" hat keine besondere Eigenschaft.
'       Deshalb geschieht hier gar nichts.
    Case 1:
'       Objekt vom Typ Mensch
      DlgMensch.Tag = ObjektDlg.Tag
      DlgMensch.Show 1
    Case 2:
'       Objekt vom Typ Vertrag
      DlgVertrag.Tag = ObjektDlg.Tag
      DlgVertrag.Show 1
'    etc.
  End Select
End Sub
```

Der folgende Codeausschnitt zeigt die Initialisierung des Dialogfeldes, das bei der Positionierung eines Objekts auf dem Zeichenfeld angezeigt wird. Hierbei wird eine Liste aller in einem Fall bereits instantiierten Objekte und eine Liste der vom Programm vorgesehenen Objekttypen angelegt.

```
Sub Form_Load()
  Dim iZObj As Integer '                                    Anzahl der im Fall instantiierten Objekte
  Dim sAnfr As String '                                      Anfrage nach einem Objektbezeichner
  Dim x As Integer '   Laufvariable

'    Auffüllen der Liste der Objektbezeichner
'    Zunächst wird die Anzahl der verfügbaren Objekte erfragt.
  iZObj = Val(Anfrage("@Anzahl(@Objekt())")

'    Nun wird jeder Objektbezeichner erfragt und in die Liste eingetragen.
  If iZObj > 0 Then
    For x = 1 To iZObj
'       Zusammenstellen der Anfrage
      sAnfr = "@Objekt(;@Pos(" + Trim$(Str$(x)) + "))" '                    Trim$(Str$(x)) s. VB-Handbuch
'       Anfragen und Anfügen an die Liste in einem Schritt
      ObjektListe.AddItem Anfrage(sAnfr)
    Next
  End If

'    Auffüllen der Liste der Objekttypen. Der Klartextbezeichner des Typs befindet sich
'    in der Tag-Eigenschaft des entsprechenden Schalters.
  For x = 0 To iZObjTyp - 1
    ObjTypListe.AddItem ToolBox.BnObjekt(x).Tag
  Next
End Sub
```

ee) Stammdatenerfassung

Nach der Instantiierung und Positionierung eines Objekts auf dem Zeichenfeld wird ein Dialogfeld angezeigt. Dieses ist abhängig vom gewählten Objekttyp. Lediglich bei dem neutralen Objekttyp *Ding* entfällt dieser Dialog.

Das Dialogfeld *Stammdaten* dient der Aufnahme gewisser Fakten, die abhängig sind vom Objekttyp und regelmäßig bei derartigen Objekten relevant sind. Welche Daten hier als relevant angesehen werden, hängt einzig am Programm *Fallskizze*. Das Dialogfeld enthält nicht alle relevanten Fakten. Das Ausfüllen des Dialogfeldes ist zudem optional. Die Daten können zu einem späteren Zeitpunkt in dieser oder einer anderen Anwendung geändert werden. Sobald das Dialogfeld angezeigt wird, werden die bereits verfügbaren Daten in die entsprechenden Felder der Maske eingetra-

gen. Der Anwender hat so schnell einen Überblick über die neuesten bekannten Stammdaten eines in der Fallskizze positionierten Objekts. Dies ist zum Beispiel dann besonders interessant, wenn die Stammdaten in einer Kanzlei zunächst mit Hilfe eines Aufnahmebogens im Vorzimmer erfaßt werden. Der eigentliche Bearbeiter kann dann quasi durch eine Skizze auch auf diese Informationen zugreifen.

Klassische Stammdaten sind etwa Name, Geburtsdatum und Anschrift beim Menschen, Wert, Flurnummer und Adresse beim Grundstück oder auch Standesamt, Registernummer und Heiratsdatum bei der Ehe etc. Neben diesen Angaben sollte das Stammdatenfeld regelmäßig eine Entscheidung vom Anwender verlangen, ob das konkrete Objekt dem entsprechenden Typus tatsächlich entspricht. Es mag zwar nur selten streitig sein, ob ein Mensch auch wirklich ein Mensch ist. Bei einem Objekt des Typs *Vertrag* hingegen kann dies nicht selten ein Problem sein. Insbesondere wird die Relevanz der Frage dann um so größer, je spezieller ein Objekttypus tatsächlich definiert ist. Ist der Typus unsicher, so mag es für den Benutzer dennoch interessant sein, die entsprechenden Stammdaten zu erfassen und das Objekt zunächst entsprechend dem Typ zu behandeln, selbst wenn sich später herausstellt, daß das Objekt einen anderen Typ hat.

Das Stammdatendialogfeld arbeitet technisch ähnlich wie die Masken des Aufnahmebogens[389]. Jedes Kontrollelement (d.h. jedes Feld der Maske) erhält eine Atlas-konforme Beschreibung des zugehörigen Faktums in seiner LinkItem–Eigenschaft. Nachdem jedoch keine festen Szenarien vorliegen, muß diese Beschreibung beim Laden des Dialogfeldes dadurch angepaßt werden, daß der Name des konkreten Objekts in die Beschreibung eingesetzt wird. Angenommen, beim Kontrollelement *Nachname* der Stammdatenmaske *Mensch* ist die LinkItem–Eigenschaft Nachname(;$). Wird nun das Dialogfeld angezeigt, um die Daten für das Objekt *Peter* einzugeben oder zu ändern, so wird der Objektbezeichner in die LinkItem–Eigenschaft des Eingabeelements eingesetzt. Hier steht nun Nachname(;$Peter). Will man die Anpassung der Felder dem jeweiligen Dialogfeld überlassen, so richtet man ein unsichtbares statisches Kontrollelement ein, in das die Prozedur, die den Dialog aufruft, den Namen des Objektbezeichers einsetzt. Der Dialog selbst reagiert auf die Change–Methode des statischen Elements mit der gewünschten Anpassung.

Die folgende Change–Methode des Elements StObjektName zeigt eine sehr allgemeine Technik, die LinkItem–Eingenschaften aller Kontrollelemente eines Dialogfeldes anzupassen. Gleichzeitig wird das Dialogfeld dazu aufgefordert, die entsprechenden Daten bei Atlas anzufragen:

```
Sub StObjektName_Change()
  Dim ix As Integer  '   Laufvariable
  Dim ip As Integer  '   Position des $ in der LinkItem-Eigenschaft
  Dim sItem As String  '                                    Kopie von LinkItem der Übersichtlichkeit halber
'    Die folgende Schleife behandelt alle Kontrollelemente der Form.
  For ix = 1 to Me.Controls.Count
'      Die LinkItem-Eigenschaft enthält per Definition dort ein $, wo der Bezeichner
'      des Objekts zu plazieren ist. Beispiel: Nachname($)"
    sItem = Me.Controls(ix - 1).LinkItem
    ip = Instr(sItem, "$")
'      Ist kein $ in der Eigenschaft, wird sie nicht berücksichtigt.
    If ip > 0 Then
```

[389] S. S. 177.

```
'       Hier wird der Objektbezeichner in die Eigenschaft eingefügt.
'       Beispiel: Nachname($)" wird zu Nachname($Peter)"
      sItem = Left$(sItem, ip) + StObjektName+ Mid$(sItem, ip + 1)
      Me.Controls(ix - 1).LinkItem = sItem
'       Nun wird die in Atlas vorhandene Information abgefragt.
      Me.Controls(ix - 1).LinkMode = 2
      Me.Controls(ix - 1).LinkRequest
      Me.Controls(ix - 1).LinkMode = 0
    End If
  Next ix
End Sub
```

Die Übertragung der Daten erfolgt, sobald der Benutzer den Bestätigungsschalter des Dialogfeldes betätigt. Betätigt er hingegen den Abbruchschalter, werden die Daten nicht übertragen. Die Art der Übertragung entspricht exakt derjenigen, die bereits bei der Anwendung *Aufnahmebogen* beschrieben wurde. Insbesondere achtet die Anwendung auch hier darauf, nicht Daten zu übermitteln, die bereits so vom System als Antwort auf eine identische Anfrage ausgegeben werden.

ff) Werte, Eigenschaften und Beziehungen

aaa) Benutzerschnittstelle

Sind Objekte auf dem Zeichenfeld plaziert, so können diesen zusätzlich zu den Stammdaten weitere Eigenschaften zugeordnet werden. Weiterhin können mehrere Objekte durch Beziehungen miteinander verknüpft werden. Hierzu markiert der Benutzer ein oder mehrere Objekte. Dies kann durch Anklicken der Objekte oder durch Aufziehen eines Markierungsbereiches erfolgen. Ein Klick mit der rechten Maustaste auf ein Objekt öffnet ein kontextsensitives Menü. Hier können die wesentlichen Funktionen aufgerufen werden. Dazu gehört der Zugriff auf die generellen Objektdaten, die Stammdaten, die Werte des Objekts und dessen Eigenschaften. Sind mehrere Objekte markiert, so enthält das Menü auch eine Option zur Anlage von Beziehungen[390].

Ähnlich verhält sich das Programm, wenn durch Aufziehen eines Markierungsrechtecks mehrere Objekte markiert wurden. Es zeigt dann sofort ein Menü an, das Optionen zum Kopieren, Löschen sowie zur Eingabe von Beziehungen enthält.

bbb) Bearbeiten von Objektwerten

Der Benutzer kann neben den Stammdaten jederzeit in einem Dialogfeld die vier denkbaren Werte eines Objekts (Existenz, Zeichenkette, Zahl und Datum) einsehen. Das Dialogfeld ist äußerst einfach konstruiert. Es erhält den Objektbezeichner in einem statischen Element StObjektName übergeben. In der Change–Methode des Elements werden die Eingabeelemente mit den Werten des Objekts gefüllt. Hierzu wird die LinkItem–Eigenschaft so geändert, daß der konkrete Objektbezeichner eingetragen wird:

```
Sub StObjektName_Change()
'   Abfrage des logischen Wertes
  EdExistiert.LinkItem = "Existiert($" + StObjektName + ")"
  EdExistiert.LinkRequest
```

[390] Obwohl diese Form der Benutzerschnittstelle für Windows 3.x entwickelt wurde, gewinnt sie unter Windows 95 besondere Aktualität. Die beschriebenen kontextsensitiven Menüs sind hier zum Standard insbesondere für die Behandlung von Objekten erhoben worden. Jedes Objekt hat dabei ein eigenes Eigenschaftsfenster, das prinzipiell dieselbe Aufgabe erfüllt, wie das Stammdatenfenster der hier beschriebenen Objekte.

```
'    Abfrage des Zeichenkettenwertes                    EdName.LinkItem = "Name($" + StObjektName + ")"
  EdExistiert.LinkRequest
'    Abfrage des Zahlwertes
  EdZahl.LinkItem = "Zahl($" + StObjektName + ")"
  EdExistiert.LinkRequest
'    Abfrage des Datumswertes
  EdDatum.LinkItem = "Datum($" + StObjektName + ")"
  EdExistiert.LinkRequest
End Sub
```

In dem Codebeispiel wurde der Lesbarkeit halber auf die Umstellung der LinkMode–Eigenschaft verzichtet. Die Übertragung der neuen Daten an Atlas erfolgt, sobald der Benutzer den Bestätigungsschalter des Dialogfeldes betätigt:

```
Sub OkBN_Click()
'    Übertragen der Informationen mit Dublettenkontrolle
  ÜbermittleElement(EdExistiert)
  ÜbermittleElement(EdName)
  ÜbermittleElement(EdZahl)
  ÜbermittleElement(EdDatum)
'    Entfernen des Dialoges
  Me.Hide
  Unload Me
End Sub
```

ccc) Bearbeiten von Eigenschaften.

Ähnlich wie auf die Werte eines Objekts hat der Benutzer auch Zugriff auf dessen Eigenschaften. Im Unterschied zu der limitierten Anzahl von Wertetypen ist die Anzahl der Eigenschaften eines Objekts grundsätzlich offen. Sie ergibt sich aus den im System des Benutzers installierten Relationen, die genau einen Parameter haben. Möchte der Benutzer die Eigenschaften eines Objekts einsehen oder ändern, so wird ihm auf Anforderung im kontextsensitiven Menü ein Dialogfeld angezeigt, welches diese Änderung ermöglicht[391]. Das Dialogfeld füllt kurz vor der Anzeige eine Liste aus Auswahlelementen aus. Für jede verfügbare Eigenschaft des Systems steht ein Auswahlelement zur Verfügung. Es kann die Werte **falsch** (0), **wahr** (1) oder **unentschieden** (2) annehmen. Zusätzlich wird über eine fette Darstellung des Textes signalisiert, daß überhaupt eine Information darüber vorliegt, ob das Objekt diese Eigenschaft hat. Dies ist besonders dann interessant, wenn der Wert unentschieden ist, da Atlas zwischen einer bewußten unentschieden (fett) und einer fehlenden Information (mager) unterscheiden kann.

[391] Modernere Dialogfelder besitzen teilweise mehr als eine Seite. Die einzelnen Seiten werden über Reiter wie in einem Karteikasten angesprochen. Unter der Verwendung dieser Technik ist auch an eine Zusammenfassung der Dialoge „Werte" und „Eigenschaften" zu denken. Der Benutzer hat so alle Informationen über ein Objekt in einem Dialog, ohne die Struktur zu verlieren.

Im folgenden Beispiel wird die Liste als eine Collection von Checkboxen realisiert. Dies sind Unterelemente zu einem Rahmen mit einer Bildlaufleiste. Die Change–Methode des statischen Elements StObjektName füllt diese Liste auf.

```
Sub StObjektName_Change()
  Dim iRelZ As Integer  '                                          Anzahl der verfügbaren Relationen
  Dim ix As Integer  '    Laufvariable
  Dim sAnfr As String  '                                  Anfrage der Übersichtlichkeit halber in eigener Variablen
  Dim sEX As String  ' Angabe, ob die Beziehung vorliegt.
  iEZ = 0  '            Anzahl der Eigenschaften im System (globale Variable)
'    Die folgende Schleife durchläuft alle Relationen, die in Atlas vorliegen.
  iRelZ = Val(Anfrage("@AnzahlRelationen())")
  if iRelZ = 0 Then Exit Sub
  For ix = 1 To iRelZ
'      Zu Beginn der Schleife wird geprüft, wieviele Parameter die Relation hat.
'      Nur Relationen mit einer Parameterzahl von 0 sind Eigenschaften.
    sAnfr = "@AnzahlParameter(#" + Format$(ix) + ")"
    If Val(Anfrage(sAnfr)) = 0 Then
'        Nun wird die Checkbox instantiiert
      iEZ = iEZ + 1
      Load CBEigenschaft(iEZ)
'        Hier wird der Bezeichner der Eigenschaft ermittelt und als Überschrift
'        der Checkbox eingesetzt.
      sAnfr = "@Relationsbezeichner(#" + Format$(ix) + ")"
      CBEigenschaft(iEZ).Caption = Anfrage(sAnfr)
'        Jetzt wird ermittelt, ob das Objekt die Eigenschaft hat.
'        Beispiel: @Beziehung(#2($Peter)) fragt an, ob Peter die Eigenschaft
'        mit dem Atlas-Index 2 besitzt.
      sAnfr = "@Beziehung(#" + Format$(ix) + "($" + StObjektName + "))"
      sEX = Anfrage(sAnfr)
'        Hier wird der Wert der Checkbox entsprechend eingestellt.
      CBEigenschaft(iEZ).FontBold = True '                                  Standardeinstellung ist fett
      Select Case sEX
        Case "Nein"
          CBEigenschaft(iEZ) = 0
        Case "Ja"
          CBEigenschaft(iEZ) = 1
        Case "Unklar"
          CBEigenschaft(iEZ) = 2
        Case Else
'            Es wurde keine Information über das Vorliegen der Eigenschaft bei dem
'            behandelten Objekt gefunden. Das Element wird als unentschieden und
'            mager dargestellt.
          CBEigenschaft(iEZ) = 2
          CBEigenschaft(iEZ).FontBold = False
      End Select
'        Für die spätere Aktualisierung wird die Anfrage
'        in der LinkItem-Eigenschaft hinterlegt.
      CBEigenschaft(iEZ).LinkItem = sAnfr
    End If
  Next ix
'    Zum Abschluß der Methode werden der Anfang der Liste auf das erste Element
'    gesetzt und die Checkboxen angeordnet.
  ErsteCBEigenschaft = 1  '                          Globale Variable gibt die erste sichtbare Checkbox an.
  OrdneEigenschaftenAn  '                  Diese Funktion soll die Checkboxen in der Liste positionieren.
End Sub
```

Die geänderten Werte werden als Reaktion auf die Click–Methode des Bestätigungsschalters wieder zurückübertragen:

```
Sub BnOK_Click()
  Dim ix As Integer  ' Laufvariable
  For ix = 1 To iEZ
'       Nur Werte von Elementen mit fetter Schrift werden zurückübertragen.
'       Die Checkbox stellt jedoch die Schrift bei jeder Änderung automatisch
'       auf fett.
    If CBEigenschaft(ix).FontBold = True Then
'         ÜbermittleCheckbox ähnelt ÜbermittleElement.
'         Es wird jedoch der Wert einer Checkbox übertragen, falls er nicht
'         von Atlas auf Anfrage so geliefert wird.
      ÜbermittleCheckbox(CBEigenschaft(ix))
    End If
  Next iX
' Der Dialog wird entladen.
  Me.Hide
  Unload Me
End Sub
```

ddd) Anlegen von Beziehungen

Die Anlage von Beziehungen erfolgt in einem Dialogfeld, das bereits skizziert wurde[392]. Es besteht aus drei Listen. Die Liste RelationenListe enthält alle Relationen, die in Atlas auf dem konkreten System zur Verfügung stehen. Die Liste Parameterleiste enthält die Parameter der aktuell markierten Relation. Die Liste ObjektListe enthält die auf dem Zeichenfeld markierten Objekte. Letztere Liste wird noch vor der Anzeige durch das Zeichenfeld aufgefüllt. Die Relationenliste wird in der Load–Methode des Dialogfensters gefüllt. Die Parameterleiste wird bei jeder Änderung der Selektion in der Relationenliste, d.h. in deren Click–Methode neu gefüllt.

Das folgende Codebeispiel zeigt zunächst den Aufruf des Dialogfensters und das Auffüllen der Objektliste. Dabei gilt ein Objekt auf dem Zeichenfeld dann als markiert, wenn es einen Rand besitzt:

```
Sub MnuBeziehung_Click()
  Dim ix As Integer ' Laufvariable
'   Der Beziehungsdialog wird geladen.
  Load DlgBeziehungen
'   Die Schleife durchläuft alle Objekte und stellt die markierten in die Liste ein.
  For ix = 1 To iZObjekte
'     Selektierte Objekte haben BorderStyle = 1
    If Objekt(ix).BorderStyle = 1 Then
      DlgBeziehungen.ObjektListe.AddItem Objekt(ix).Caption
    End If
  Next ix
'   Das Fenster wird modal angezeigt.
  DlgBeziehungen.Show 1
'   Da das Fenster alle Aktionen selbständig erledigt, ist nach der Anzeige kein
'   weiterer Funktionsaufruf notwendig.
End Sub
```

[392] S. S. 194.

Das nächste Beispiel zeigt das Auffüllen der Relationenliste in der Load–Methode des Beziehungsfensters. Die Prozedur ist eine Abwandlung der Methode StObjektName_Change des Eigenschaftsfensters, da beide in einer Schleife alle auf dem System verfügbaren Eigenschaften abarbeiten:

```
Sub Form_Load()
  Dim ix As Integer  '   Laufvariable
  Dim iRelZ As Integer  '                                   Anzahl der verfügbaren Relationen
  Dim sAnfr As String  '                          Anfrage der Übersichtlichkeit halber in eigener Variablen
  Dim iParZ As String  '                                    Anzahl der Parameter einer Relation
'    Die folgende Schleife durchläuft alle Relationen, die in Atlas vorliegen.
  iRelZ = Val(Anfrage("@AnzahlRelationen())")
  if iRelZ = 0 Then Exit Sub '                                          Es gibt keine Relationen
  For ix = 1 To iRelZ
'      Hier wird geprüft, wieviele Parameter die Relation hat. Nur Relationen
'      mit einem Parameter sind Eigenschaften.
    sAnfr = "@AnzahlParameter(#" + Format$(ix) + ")"
'      Im Gegensatz zum Eigenschaftsfenster werden hier alle Relationen aufgezählt,
'      die Beziehungen zwischen allen oder einem Teil der markierten Objekte
'      herstellen können. Eigenschaften werden allerdings ausgeschlossen.
    iParZ = Val(Anfrage(sAnfr))
    If iParZ < ObjektListe.Count And iParZ > 0 Then
      sAnfr = "@Relationsbezeichner(#" + Format$(ix) + ")"
      RelationenListe.AddItem Anfrage(sAnfr)
    End If
  Next ix
End Sub
```

Wählt der Benutzer in der Liste der Relationen eine bestimmte aus, so wird die Parameterleiste mit den Parametern der gewählten Relation gefüllt. Diese Aktion erfolgt im Programm in der Click–Methode der Relationenliste:

```
Sub RelationenListe_Click()
  Dim ix As Integer  '   Laufvariable
  Dim sAnfr As String  '                          Anfrage der Übersichtlichkeit halber in eigener Variablen
  Dim iParZ As String  '                                    Anzahl der Parameter einer Relation
'    Zuerst wird der Inhalt der Parameterleiste entfernt.
  Parameterleiste.Reset
'    Als erster Paramter wird der Relationsbezeichner selbst eingetragen.
  Parameterleiste.AddItem Relationenliste
'    Für den Einstieg in eine Schleife wird die Anzahl der Parameter erfragt:
  sAnfr = "@AnzahlParameter(" + RelationenListe + ")"
  iParZ = Val(Anfrage(sAnfr))
'    Die Schleife trägt nun die restlichen Parameter in die Liste ein.
'    Aufgrund der Vorselektion muß iParZ größer als 0 sein, wenn die
'    globale Datenbasis nicht inzwischen geändert wurde.
  For ix =1 to iParZ
'      Aufbau einer Anfrage nach dem Bezeichner des ix-ten Parameters.
    sAnfr = "@ParameterBezeichner(" + RelationenListe +";"+Format$(ix) +  ")"
'      Eintragen des Anfrageergebnisses in die Liste.
    Parameterleiste.AddItem Anfrage(sAnfr)
  Next ix
End Sub
```

Der Benutzer hat nun die Möglichkeit, durch Auswahl der Einträge in der Parameterleiste und nachfolgend in der Objektliste, den Parametern entsprechende Objekte zuzuweisen. Da dies keine Vorgänge sind, die einen Zugriff auf Atlas benötigen,

wird hier auf eine Darstellung des Codes verzichtet. Weiterhin kann der Anwender in einem Optionsfeld angeben, ob die Beziehung tatsächlich existiert. Hat er seine Auswahl getroffen, so betätigt er den Bestätigungsschalter, um die Beziehung an Atlas zu übermitteln. Weiterhin wird hierdurch eine interne Information abgelegt, die die graphische Anzeige der Beziehung ermöglicht.

Zusätzlich zur Anlage der eigentlichen Beziehungen werden unter bestimmten Umständen auch die komplementären Beziehungen angelegt. Das folgende Codebeispiel zeigt die wesentlichen Bestandteile dieses Vorgangs in der Click–Methode des Bestätigungsschalters:

```
Sub BnOK_Click()
  Dim ix As Integer  '    Laufvariable
  Dim sAnfr As String  '                                  Anfrage der Übersichtlichkeit halber in eigener Variablen
  Dim sAnfr2 As String  '                                 Anfrage der Übersichtlichkeit halber in eigener Variablen
  Dim sVal As String  ' Erwarteter Wert einer Anfrage.
  Dim iParZ As String  '                                                Anzahl der Parameter einer Relation
'    Zunächst sollte überprüft werden, ob jedem Parameter ein Objekt zugewiesen
'    wurde. Andernfalls sollte eine Fehlermeldung erfolgen.
  For ix = 0 To Parameterleiste.ListCount - 1
'       Jeder Parameter muß ein ItemData > 0 haben
    If Parameterleiste.ItemData(ix) <= 0 Then
'          Ausgabe der Fehlermeldung und Beenden der Funktion.
      MsgBox "Nicht alle Parameter sind mit einem Objekt belegt"
      Exit Sub
    End If
  Next ix
'    Nun wird die Übermittlung zusammengestellt. Die Informationen, welches Objekt
'    welchem Paremter zugeordent wurde, legt das Programm in der ItemData-Eigenschaft
'    der Einträge der Parameterleiste ab. Deshalb durchläuft eine Schleife alle diese
'    Einträge. Zunächst wird jedoch der Beziehungsparameter eingetragen.
  sAnfr = "@Beziehung(" + Parameterleiste.List(0) + "($"
'    Es folgt die Schleife durch die Parameter.
  For ix = 0 To Parameterleiste.ListCount - 1
'       Alle Elemente werden durch Kommata voneinander getrennt
    If x > 0 Then sAnfr = sAnfr + "; $" ' Das erste $-Zeichen steht bereits.
'       Der Index des Objekts steht in der ItemData-Eingenschaft -1 des Paramters.
    sAnfr = sAnfr + ObjektListe.List(Parameterleiste.ItemData(ix) - 1)
  Next ix
'    Zuletzt werden die Klammern geschlossen.
  sAnfr = sAnfr + "))"
'    Aus dem Wert des Auswahlelements "Beziehung existiert"
'    ergibt sich der Wert der Beziehung.
  Select Case BnBeziehungExistiert
    Case 0: sVal = "Nein"
    Case 1: sVal = "Ja"
    Case 2: sVal = "Unklar"
  End Select
'    Zur Vermeidung von Redundanzen wird zunächst abgefragt, ob die Beziehung so
'    in Atlas eingetragen ist. Sofern dies nicht der Fall ist,
'    wird die Beziehung übermittelt.
  If Anfrage(sAnfr) <> sVal Then Übermittlung(sAnfr, sVal)
'    Auf die Ablage der internen graphischen Informationen wird hier verzichtet.
'    Nun werden die korrespondierenden Beziehungen aufgebaut.
'    Dazu werden alle Parameter als Relationen behandelt.
  For ix = 1 To Parameterleiste.ListCount - 1
```

```
'       Im ersten Schritt wird die Anzahl der Paramter ermittelt.
    sAnfr = "@AnzahlParameter(" + Parameterleiste.List(ix) + ")"
    iParZ = Val(Anfrage(sAnfr))
'       Bei 0 Parametern wird eine Eigenschaft angelegt.
'       Bei einem Parameter wird eine Beziehung zum Hauptobjekt angelegt.
    If iParZ < 2 Then
'         Der jew. Parameter bezeichnet die Relation der korr. Beziehung.
      sAnfr = "@Beziehung(" + Parameterleiste.List(ix)
'         Dessen Objekt ist das Hauptobjekt.
      sAnfr = sAnfr +"($" + ObjektListe.List(Parameterleiste.ItemData(ix) - 1)
'         Falls es einen weiteren Paramter der Relation gibt,
'         ist eine korrespondierende Relation anzulegen.
      if iParZ = 1 Then
'           Dabei ist zunächst zu prüfen, ob dieser Paramter mit der Hauptrelation
'           übereinstimmt. Nur dann entspricht der Parameter einer
'           korrespondierenden Relation.
        sAnfr2 = "@ParameterBezeichner(Parameterleiste.List(ix), 1)"
        If Anfrage(sAnfr2) = Parameterleiste.List(0) Then
          sAnfr = sAnfr + ";$" + _
            ObjektListe.List(Parameterleiste.ItemData(0) + "))"
          If Anfrage(sAnfr) <> sVal Then Übermittlung(sAnfr, sVal)
        End If  '   Anfrage(sAnfr2) = Parameterleiste.List(0)
      Else  '       iParz = 1. Es wird also eine Eigenschaft angelegt.
        sAnfr = sAnfr + "))"
        If Anfrage(sAnfr) <> sVal Then Übermittlung(sAnfr, sVal)
      End If  '     iParz = 1
    End If  '           iParZ < 2
  Next ix  '            For ix = 1 To Parameterleiste.ListCount - 1
End Sub
```

Die Realisation der Eingabe beliebiger Beziehungen zeigt die Flexibilität des Systems. Sie stellt jedoch nicht den Hauptanwendungsfall dar, sondern ist eine der aufwendigeren Anwendungen, die mit Atlas denkbar sind.

gg) Ein- /Ausgabefilter

Eingabefilter dienen vor allem dazu, allen Eingaben, die der Benutzer tätigt, die entsprechenden Zusatzinformationen beizufügen. Ausgabefilter werden, wie im Konzept der Fallskizze bereits beschrieben, zunächst durch Sichten realisiert. Der Benutzer kann also mehrere Ausgabefilter einrichten und die unterschiedlichen Anfrageergebnisse durch unterschiedliche Farben etc. kennzeichnen lassen.

Hinzu kommt jedoch, daß bei der Anzeige etwa einer Stammdatenmaske angegeben werden muß, mit welchem Filter die Felder zunächst aufgefüllt werden sollen und mit welchem Filter die Speicherung erfolgt. Gibt der Anwalt beispielsweise aufgrund einer Replik die Daten aus der Sicht des Gegners ein, so sollte er in einer einfachen Maske zunächst die Daten aus der Sicht seines Mandanten sehen. Er kann die Daten dann bestätigen oder ändern. Mit der Bestätigung dieser Angaben werden sie an Atlas übermittelt. Dabei erhalten Sie die Zusatzinformation, daß ihre Quelle nun der Gegner ist. Der Benutzer sollte deshalb die Möglichkeit haben, an jeder Stelle, an der er Daten eingeben kann, die Einstellungen der beiden Filter (Ein- bzw. Ausgabe) zu verändern.

Die konkrete Realisierung der Dialogfelder zur Einstellung der Filter entspricht wiederum der Anwendung *Aufnahmebogen* und bedarf deshalb hier keiner genaueren Erklärung.

hh) Assistenten

Assistenten dienen der graphischen Aufarbeitung von Sachverhalten, die bereits auf anderem Wege erfaßt wurden und somit im System vorhanden sind. Die einfachste

Form eines Assistenten positioniert zunächst alle Objekte auf der Zeichenfläche. Sie wird diese hierzu in einer gekachelten Anordnung von oben links nach unten rechts verteilen. Danach arbeitet der Assistent alle Relationen des Systems ab, und überprüft, ob hierzu Beziehungen vorhanden sind. Er bedient sich zu diesem Zweck der Funktion @AlleParameter. Der folgende Code zeigt das Vorgehen bei der Analyse des Falls:

```
Sub AssistentAllegemein()
  Dim iAnzObj  '     Anzahl der Objekte eines Falls
  Dim iAnzBez  '     Anzahl der Beziehungen eines Falls bzgl, einer Relation
  Dim iAnzPar  '     Anzahl der Parameter einer Relation
  Dim sAnfr As String  '                                  Anfrage der Übersichtlichkeit halber in eigener Variablen
  Dim sPar As String  '                                                                                  Parameter
  Dim ix As Long '  Laufvariable
  Dim iy As Long '  Laufvariable

'    Hier werden zunächst alle Objekte positioniert.
  iAnzObj = Anfrage("@Anzahl(@Objekt())")
  For ix = 1 To iAnzObj
    sAnfr = "@Objekt(#" + Format$(ix) + ")"

'      Die nachfolgend postulierte Funktion ObjektAutoPos(iPos, sBezeichner)
'      positioniert in dieser Anwendung das ix-te Objekt auf der Zeichenfläche.
'      Auf die Beschreibung der Realisierung wird hier verzichtet.
'      Dabei ermittelt die Funktion auch, mit welchem Sinnbild das Objekt
'      darzustellen ist.
'      Hierzu fragt sie in einer vorgegebenen Hierarchie ab,
'      welche Eigenschaften das Objekt besitzt. Die hochwertigste
'      Eigenschaft bestimmt das Symbol.
    ObjektAutoPos(ix, Anfrage(sAnfr))
  Next ix

'    Der folgende Abschnitt ermittelt und visualisiert die Beziehungen der
'    Objekte zueinander. Hierzu durchläuft das Programm zunächst alle Relationen
'    des Systems, um von dort aus auf die konkreten Beziehungen zuzugreifen.
  iAnzRel = Anfrage("@AnzahlRelationen()")
  For ix = 1 to iAnzRel

'      In einer verschachtelten Schleife werden alle Beziehungen,
'      die auf einer Relation beruhen, abgefragt.
'      Angefragt wird zunächst die Anzahl der Beziehungen.
    sRel = "#" + Format(ix) '                                  Darstellung der Relation in Index-Syntax
    sAnfr = "@Anzahl(" + sRel + "())"
    iAnzBez = Anfrage(sAnfr)

'    Nur wenn es Beziehungen gibt, wird fortgefahren.
    if iAnzBez > 0 Then
'        Ermitteln der Anzahl der Parameter einer Relation
      sAnfr = "@AnzahlParamter(" + sRel + "())"
      iAnzPar = Anfrage(sAnfr)
'      Abfragen der Daten aller Beziehungen.
      For iy = 1 to iAnzBez
'          Die Abfrage der konkreten Beziehung erfolgt über die Fiktion
'          @AlleParameter.
'          Die hochgezählten Relationen werden über die Index-Syntax identifiziert.
'          Die hochgezählten Beziehungen werden über @Pos identifiziert.
'          Beispiel: @AlleParameter(#3(;;;@Pos(1)))
'          Die erste Beziehung, die auf der Relation mit dem Index 3 beruht.
'          Die Parameterleiste enthält soviele Semikola wie (sekundäre) Parameter.
'          Sie kann leicht mit der BASIC-Funktion String$() erzeugt werden.
        sAnfr = "@AlleParameter(" + sRel + "(" _
          + String$(iAnzPar, Asc(";")) _
          + "@Pos(" + Format$(iy) + ")))"
```

```
'           Die nachfolgend postulierte Routine "BeziehungAuto(sRel, sParameter)"
'           erzeugt die Graphik für die angegebene Beziehung. Der Code wird hier
'           nicht dargestellt, da es ein interner Vorgang unabhängig von Atlas ist.
'           Zunächst wird noch der Name der Relation ermittelt, da dieser statt des
'           Index in der Graphik als Beschriftung der Pfeile eingesetzt werden soll.
          sRel = Anfrage(("@Objekt(" + sRel + ")")
          BeziehungAuto(sRel, Anfrage(sAnfr))
        Next iy
      End If
    Next ix
End Sub
```

Ein Assistent, der im Gegensatz zu diesem allgemeinen Verfahren auf Basis von Szenarien arbeitet, ist effektiver und einfacher. Während der vorangegangene Assistent alle Objekte quasi wahllos auf der Zeichenfläche postiert, hat ein Szenariogestützter Assistent die Möglichkeit, die Objekte graphisch übersichtlich anzuordnen. Die folgenden Abbildungen stellen beispielhaft eine Ehe und die Kinder der Eheleute dar. Im ersten Abbild wird ein allgemeiner Assistent eingesetzt, im zweiten ein Assistent, der das Szenario *Familie mit Kindern* zugrunde legt:

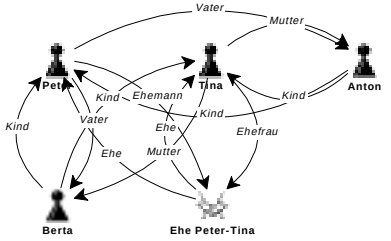


Abbildung 49: Automatisch erstellte Fallskizze ohne Berücksichtigung eines Szenarios. Der Assistent ordnet die Objekte von links oben nach rechts unten in der Reihenfolge ihrer Instantiierung an.

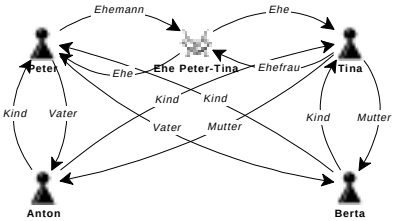


Abbildung 50: Automatisch erstellte Fallskizze mit Berücksichtigung eines Szenarios. Der Assistent ordnet die Objekte nach einem vorgegebenen Schema an.

Die zweite Darstellung ist übersichtlicher als die erste. Der Assistent ordnet hier die Objekte nach einem festen Schema. Ehemann, Ehefrau und das Objekt für die Ehe werden auf vorgegebenen Punkten positioniert. In der zweiten Reihe werden die Kinder abhängig von ihrer Anzahl gleichmäßig angeordnet.

Abschließend wird durch ein Codebeispiel ein solcher Assistent demonstriert. In diesem Fall wird das Szenario im Programm hart codiert. Die Vorgehensweise für ein spezifisches Szenario (hier *Familie mit Kindern*) ist also unmittelbar im Programmcode fixiert. Eine flexiblere Variante würde die Beschreibung solcher Szenarien mit Hilfe einer einfachen Skriptsprache erlauben. Diese Vorgehensweise ermöglicht dann eine leichte Pflege solcher Szenarien durch Fachexperten. Sie ist jedoch aufwendiger zu implementieren. Nachdem an dieser Stelle lediglich das Prinzip, nicht jedoch die optimale Realisierung dargelegt werden soll, wird auf eine skriptgesteuerte Codierung verzichtet.

```
Sub AssistentFamilie()
  Dim iAnzKinder As Integer  '                                   Anzahl der Kinder zweier Personen
  Dim sEhe As String  '                                                    Bezeichner der Ehe
  Dim sEhemann As String  '                                          Bezeichner des Ehemanns
  Dim sEhefrau As String  '                                           Bezeichner der Ehefrau
  Dim sKind As String  '                                                Bezeichner eines Kindes
  Dim sPx As String  '  horizontale Position eines Objekts
  Dim sAnfr As String  '                          Anfrage der Übersichtlichkeit halber in eigener Variablen
  Dim ix As Long '  Laufvariable
  Dim iRx As Long '  Linker und rechter Rand
  Dim iRy As Long '  Zeilenabstand sowie oberer Rand
  Dim iy As Long '  Laufvariable

'    Setzen der Ränder
  iRx = 10
  iRy = 10

'    Im ersten Schritt wird ein Objekt des Typs Ehe gesucht und positioniert.
  sEhe = Anfrage("@Objekt(Ehe())")
'    Der Assistent bricht ab, falls keine Ehe im aktuellen Fall instantiiert ist.
  if sEhe = "" Then Exit Sub

'    Nun werden die Objekte für Ehemann und -frau ermittelt.
  sEhemann = Anfrage("Objekt(Ehemann(;$" + sEhe + "))")
  sEhefrau = Anfrage("Objekt(Ehefrau(;$" + sEhe + "))")
'    Der Assistent bricht ab, falls Ehemann oder Ehefrau
'    im aktuellen Fall nicht instantiiert sind.
  if sEhemann = "" Or sEhefrau = "" Then Exit Sub

'    Die nachfolgende postulierte Routine "ObjektNeu(sObj, x, y, Symbol)" positioniert ein
'    neues Objekt am angegebenen Punkt mit dem angegebenen Symbol.
'    Falls das Objekt bereits auf der Zeichenfläche existiert, wird es nur verschoben.
'    "ZeichenFenster" bezeichnet das Fenster der Zeichenfläche.

'    "SYMBOL_VERTRAG" ist eine Konstante, die für das Symbol "Vertrag" steht.
'    Der Vertrag wird horizontal zentriert.
  ObjektNeu(sEhe, ZeichenFenster.ScaleWidth / 2, iRy, SYMBOL_VERTRAG)

'    "SYMBOL_MENSCH" ist eine Konstante, die für das Symbol "Mensch" steht.
'    Der Ehemann wird am rechten Rand dargestellt.
  ObjektNeu(sEhemann, iRx, iRy, SYMBOL_MENSCH)
'    Die Ehefrau wird am linken Rand dargestellt.
  ObjektNeu(sEhefrau, ZeichenFenster.ScaleWidth - iRx, iRy, SYMBOL_MENSCH)

'    Nun werden die Beziehungen graphisch dargestellt. Hierzu werden die
‘    einzelnen Pfeile mit der hier nicht weiter beschriebenen Funktion
'    "PfeilNeu(sUrsprung, sZiel, sBeschriftung)" in die internen
'    Zeicheninformationen aufgenommen.
  PfeilNeu(sEhe, sEhemann ,"Ehe")
  PfeilNeu(sEhe, sEhefrau ,"Ehe")
  PfeilNeu(sEhemann, sEhe ,"Ehemann")
  PfeilNeu(sEhefrau, sEhe ,"Ehefrau")

'    Nach der Darstellung der Ehe werden die einzelnen Kinder erfragt.
  iAnzKinder = Anfrage("@Anzahl(Kind(;$" + sEhemnann + ";$" + sEhefrau + "))")
  For ix = 1 To iAnzKinder

'      Die Objekte werden gleichmäßig zwischen den Rändern des
‘      Zeichenbereiches verteilt.
    iPx = (ZeichenFenster.ScaleWidth - 2 * iRx) / (ix -1) + iRx

'      Hier wird der Objektname angefordert.
    sAnfr = "@Objekt(Kind(;$" + sEhemann + ";$" + sEhefrau + _
      ";@Pos(" Format$(ix) + ")))"
    sKind = Anfrage(sAnfr)
```

```
'       Positionierung des Objekts sowie Anlage der Pfeile.
‘       Bei diesem Vorgang erfolgt hier keine Kontrolle,
‘       ob sEhemann auch wirklich der Vater von sKind ist.
    ObjektNeu(sKind, iPx, iRy * 2)
    PfeilNeu(sKind, sEhemann, "Kind")
    PfeilNeu(sEhemann, sKind, "Vater")
    PfeilNeu(sKind, sEhefrau, "Kind")
    PfeilNeu(sEhefrau, sKind, "Mutter")

  Next ix
End Sub
```

d) Ausblick

In den vorangegangenen Ausführungen wurde ein Programm beschrieben, das eine für den Computer neuartige Eingabeform von Faktenwissen beschreibt. Dabei lehnt sich das Programm einerseits an eine klassische Arbeitstechnik des Juristen an. Andererseits bildet es fast identisch die interne Darstellung der hier vorgestellten Schnittstelle ab. Diese Koinzidenz ist nicht zufällig, da sowohl der klassischen Fallskizze als auch der Schnittstelle ein juristisch orientiertes Modell der Wirklichkeit zugrunde liegt. Aufgrund der Ähnlichkeit der Ansätze ist eine programmtechnische Implementierung der Fallskizze auf der Grundlage von Atlas mit überschaubarem Aufwand verbunden.

Der Einsatzbereich für graphisch orientierte Programme wird insbesondere durch Tendenzen in der Hardwareindustrie interessant. Hier werden verstärkt Computer vorgestellt, die in handlicher Größe eine Dateneingabe mit einem Stift erlauben. Diese Eingabeform erlaubt primär eine komfortable Eingabe von Graphiken. Sie erlaubt jedoch darüber hinaus auch die Interpretation der Eingaben. So kann einerseits Handschrift in Text umgewandelt werden. Es können andererseits auch gezeichnete Symbole als sogenannte Gesten für bestimmte Aktionen interpretiert werden.

Denkbar ist es beispielsweise, daß typisierte Objekte einfach durch definierte Gesten am Bildschirm positioniert und in Atlas instantiiert werden[393]. Dabei würde der Name des Objekts in einer für Schrifterkennung optimierten Maske abgefragt werden. Andere Gesten können als Beziehungen zwischen den Objekten interpretiert werden oder sie starten einen komplexen Assistenten. So könnte ein Endloszeichen ∞ als Geste für eine Ehe zwischen den nebenstehenden Objekten interpretiert werden. Die folgende Zeichnung reicht dementsprechend aus, um die Ehe zwischen zwei Menschen im System einzugeben:

Abbildung 51: Beispiel für die Kombination dreier Gesten zur Instantiierung einer Ehe zwischen zwei Objekten.

Systeme wie OLE 2 dienen dazu, daß mehrere Programme quasi in einem Rahmen gestartet werden können. Führt man diesen Gedanken weiter, so reicht auch eine Geste aus, um etwa das Programm *Fallskizze* in ein allgemeines Arbeitsblatt einzubetten. Eine andere Geste startet ein Berechnungsprogramm und eine dritte Geste eine Checkliste. Auf diese Weise wird eine Art globales Skizzenpapier denkbar, das dem Benutzer völlig intuitiv die Instrumente zur Verfügung stellt, die er für einen bestimmten Vorgang benötigt.

---

[393] Vgl. *Kraft*, jur-pc 93, 2331, 2333.

3. Sachverhaltsstrukturierung

In den vorangegangenen Abschnitten wurden zwei Programme mit ausführlichen Codebeispielen dargestellt. Eines der Programme ging von einer relativ starren Vorgabe eines typischen Szenarios aus, wohingegen das andere die vorhandenen Relationen nutzte, um sehr flexibel mit Sachverhalten umzugehen. Die im folgenden vorgestellten Programme werden sich technisch gesehen nicht erheblich außerhalb dieser Bandbreite bewegen. Auf die Darstellung von Programmcodes wird deshalb nachfolgend verzichtet. Statt dessen werden lediglich die Konzepte für einige weitere praktische Anwendungen vorgestellt. Sie dienen gleichzeitig für einen weiteren theoretischen Abgleich der Bedürfnisse juristischer Tätigkeiten mit den Möglichkeiten des beschriebenen Systems.

Atlas ermöglicht die Verwaltung unterschiedlichen Sachvortrags mehrerer Parteien anhand juristischer Anknüpfungspunkte. Die bisher gezeigten Darstellungsformen des in Atlas verfügbaren Inhalts erlaubten jedoch keine Ordnung nach juristischen Kriterien. Der folgende Abschnitt stellt eine Anwendung vor, die eine derartige Strukturierung des Sachverhalts anhand eines praxiserprobten Verfahrens ermöglicht.

a) Ansatz

Die wohl gängigste Form der Sachverhaltsstrukturierung in der Praxis ist die sogenannte *Relation*[394,395]. Die Identität des Wortes Relation mit dem in dieser Arbeit verwendeten Relationsbegriff ist rein zufällig. Bei der Relation in der meist richterlichen Praxis handelt es sich um eine tabellarische Gegenüberstellung des streitigen und unstreitigen Vortrags der Parteien. Dabei erfolgt die Ordnung der einzelnen Punkte anhand der Tatbestandsmerkmale der zugrundeliegenden Normen. Das folgende Abbild zeigt eine derartige Relation

Tatbestand	*AS*	*AG*	*Beweise*
1) § 1569 BGB	+	-	
a) Scheidung	+	+	
b) AS kann nicht selbst für ihren Unterhalt sorgen.	+	-	
aa) § 1570 BGB	+	-	Kind A als Zeuge
bb) § 1573 BGB	+	-	
cc) AS ist bereits erwerbstätig.	-	+	Kind B, Herr C als Zeugen

Wesentliches Ziel einer elektronischen Unterstützung der Sachverhaltsstrukturierung ist die Ordnung bekannter Fakten nach juristischen Kriterien[396]. Dabei ist es wünschenswert insbesondere bei komplexen Sachverhalten die einzelnen Fakten über einen hierarchischen Zugang zu ordnen[397]. Hieraus ergibt sich dann die Möglichkeit, den Sachverhalt in unterschiedlicher Detaillierung zu überblicken. Im Idealfall beherrscht diese Ordnungsfunktion bereits die logischen Zusammenhänge der Merkmale, und die zugrundeliegenden Beweisregeln, so daß relevante und irrelevante non liquet Lagen leicht auffindbar sind.

---

[394] Für die Umsetzung in EDV vgl. *Kienbaum-Gutachten*, 2.4.3.; Hierzu *Herberger*, jur-pc 91, 1259.

[395] Vgl. *Schneider*, **Arbeitstechnik**, S80 ff. mit einer dringenden Mahnung, daß die Relationentechnik auch in der richterlichen Praxis von großer Bedeutung ist.

[396] Vgl. *Kienbaum-Gutachten*, 2.4.3.

[397] Zur Bildung von Begriffsstrukturen vgl. *Haft*, S. 79 ff.

b) Grundzüge einer Realisierung

Eine derartige Ordnungsfunktion ist nicht Bestandteil des Grundprogramms Atlas. Die Bandbreite der juristischen Grundvoraussetzungen, die einer solchen Ordnungsfunktion zugrunde liegt, hat eine andere Qualität als die Bildung von Relationen. Deshalb ist es auch wenig sinnvoll, die juristische Strukturierung zum unmittelbaren Bestandteil einer Schnittstelle zu machen, die sich lediglich mit dem Austausch von Fakten beschäftigt. Dennoch ist es denkbar, daß ein solches System, dem juristisches Strukturwissen zugrunde liegt, auf die Relationen von Atlas aufsetzt. Hierzu reicht es prinzipiell aus, wenn die sekundäre Datenquelle ähnlich wie in der beschriebenen Anwendung *Fallskizze* auf die Relationen in der globalen Atlas-Datenbasis verweist. Die sekundäre Datenbasis enthält nun neben den Indizes der bezeichneten Relationen noch Informationen über deren hierarchische Einordnung und eventuelle logische Verknüpfungen.

Das folgende Abbild stellt einen rein hierarchischen Zusammenhang einiger Unterhaltsvorschriften dar:

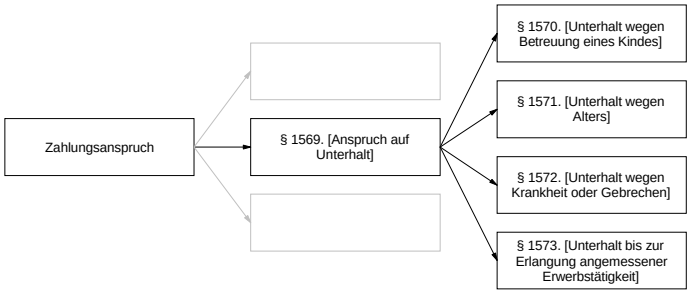


Abbildung 52: Vereinfachtes Schema der Hierarchie der Unterhaltsvorschriften.

Die Frage, wie die zugrundeliegenden Relationen zu bilden sind, sei hier nur kurz gestreift. Denkbar ist beispielsweise, die Vorschriften als Eigenschaften eines Anspruchs zu definieren. Ausgehend von einer allgemeinen Relation $Anspruch(anspruch, gläubiger, schuldner)$ mag der Unterhaltsanspruch durch die Eigenschaft $Unterhaltsanspruch(anspruch)$ beschrieben werden. Genausogut ist aber auch die Bildung einer Relation $Unterhaltsanspruch(anspruch, gläubiger, schuldner)$. Wesentlicher als eine Klärung der Einzelfragen bei jeder der Relationen mag die Herausarbeitung grundsätzlicher Problembereiche und die Formulierung von Regeln für die Lösung derartiger Probleme sein. So wurde in dieser Arbeit bereits klargestellt, daß jedenfalls nicht alle anspruchsbegründenden oder auch vernichtenden Faktoren als Parameter einer Relation dienen müssen. Ähnliche Grundregeln mögen auch für Probleme wie das hier kurz angerissene formulierbar sein. Solcherlei Fragen sollten Gegenstand der bereits postulierten Normung von Relationen sein. Sie sind von einer Vielzahl von Faktoren abhängig, die an dieser Stelle nicht abschließend diskutiert werden können.

Sie sind für die Gestaltung der hier behandelten Anwendung auch nicht zwingend. In einer sekundären Datei ließe sich die gezeigte Hierarchie etwa wie folgt abbilden:

```
1,1;...2,4;3,5;3,6;3,7;3,8
```

Der gezeigte Text besteht aus einer Liste von Zahlenpaaren. Die Paare sind durch Semikola, die einzelnen Zahlen durch Kommata getrennt. Die erste Zahl eines Paares bezeichnet die hierarchische Ebene (im Beispiel 1-3). Die zweite Zahl bezeichnet den Index der bezogenen Relation.

Eine Anwendung ist nun in der Lage, aus diesen Informationen einen hierarchischen Baum aufzubauen, ihn mit den Bezeichnern der Relationen zu füllen und so lesbar zu machen. Zuletzt kann sie die unterschiedlichen Behauptungen der Parteien und deren Beweismittel bezogen auf die Relationen sammeln und in eine Tabelle einstellen. Behauptungen sind dabei die Existenzwerte der dargestellten Beziehungen sowie die Werte der beteiligten Objekte. Die Beweismittel werden über die Atlas-Funktion *@Beweisobjekt*(x) und die entsprechende Beweisrelation (zum Beispiel *Zeuge*$(z,\ behauptung)$ oder etwa *Urkundsbeweis*$(u,\ behauptung)$) zugänglich. Dabei muß das Programm selbst wissen, welche Beweismittel es bei der aktuellen Rechtslage gibt und durch welche Relation sie im System dargestellt werden.

Die technische Vorgehensweise entspricht etwa derjenigen des Assistenten der Fallskizze. Im Unterschied zu diesem Assistenten werden jedoch die Relationen nicht in der Reihenfolge abgearbeitet, in der sie in der globalen Datenbasis vorliegen, sondern anhand der Strukturinformationen. Das folgende Abbild zeigt schematisch eine solche Anwendung:

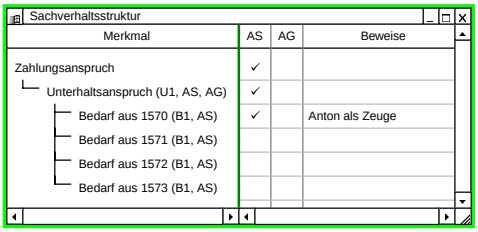


Abbildung 53: Schema für eine Anwendung zur Sachverhaltsstrukturierung.

Die Ebenen eines hierarchischen Baumes, wie er auf der linken Seite des Fensters dargestellt ist, können typischerweise vom Benutzer ein- und ausgeblendet werden. Sie werden in modernen Computeranwendungen vielfach zur Strukturierung von Informationen eingesetzt[398].

c) Konzeptionen zur Strukturierung

Die zunächst recht einfach wirkende Lösung wird bereits dann problematisch, wenn im Beispiel zwei Unterhaltsansprüche im Streit stünden, etwa weil der Antragsgegner seinerseits einen Anspruch gegen die Antragstellerin behauptet. Ein gänzlich anderes Problem stellt die Frage dar, wie die zugrundegelegten Strukturen in das System des Benutzers gelangen, genauer gesagt, wer sie entwirft und wie sie im System installiert werden. Nachfolgend werden zwei unterschiedliche Konzeptionen skizziert, die beide Fragen differenziert lösen.

aa) Klassischer Thesaurus

Bleibt man bei der klassischen hierarchischen Gliederung der Relationen, so handelt es sich hierbei faktisch um einen Thesaurus juristischer Begriffe. Es bestehen dabei keinerlei logische Zusammenhänge zwischen den Relationen, diese sind lediglich durch Achsen eines hierarchischen Baumes verbunden. Beruhen nun mehrere Beziehungen auf einer Relation so tritt spätestens dann ein Problem auf, wenn auch Beziehungen auf hierarchisch untergeordneten Relationen beruhen. Dem System ist es dann nicht mehr möglich, diese untergeordneten Beziehungen genau einer der beiden übergeordneten Beziehungen zuzuordnen. Damit die Zuordnung nicht völlig willkürlich erfolgt sind die tatsächlichen Beziehungen von der zugrundeliegenden Relation abzukoppeln und als eigene hierarchische Ebene darzustellen. Dies ent-

[398] In den 32-Bit Varianten von Windows sind sie bereits als eigenes Kontrollelement im Betriebssystem integriert.

spricht ansatzweise der Visualisierung des Reports *Beziehungen* des Programms Atlas[399]:

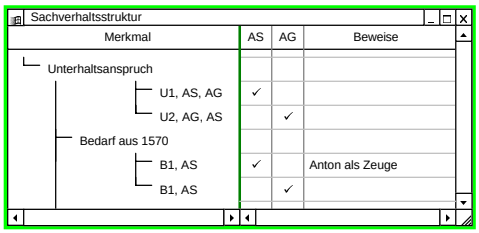


Abbildung 54: Schematische Darstellung des Programms zur Sachverhaltsstrukturierung bei mehreren Beziehungen, die auf einer Relation beruhen.

Diese Form der Strukturierung ist noch nicht geeignet, um die unterschiedlichen Ansprüche mit ihren Tatsachenbehauptungen übersichtlich zu differenzieren. Zu diesem Zweck müssen die Parameter der Relationen mitberücksichtigt werden. Existieren bei den verwendeten Relationen wie zum Beispiel bei *Unterhaltsanspruch*$(x, gläubiger, schuldner)$ und *Bedarf_aus_1570*$(x, gläubiger)$ identische korrelierende Relationen wie *Gläubiger*$(x, anspruch)$ so können diese als Anhaltspunkt für eine Gruppierung verwendet werden.

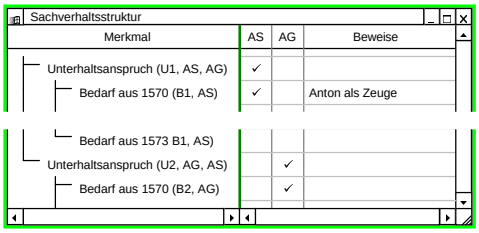


Abbildung 55: Schema mit gruppierten Beziehungen.

Das Verfahren versagt freilich, wenn in einer Relation mehrere Parameter mit derselben korrelierenden Relation wie *Mensch* oder *Ding* etc. beschrieben werden. Hier kann die Identität von Parametern, Objekten unterschiedlicher Beziehungen nur noch intellektuell zugewiesen werden.

Dies wirft die grundsätzliche Frage danach auf, wer für den Aufbau des Thesaurus und somit die Vorgabe der Struktur verantwortlich sein soll. Nachdem die Thesaurusinformationen an einer von Atlas unabhängigen Stelle gespeichert sein müssen, besteht grundsätzlich auch die Möglichkeit, die zugrundeliegenden Dateien von einem Experten aufbereiten zu lassen und getrennt an den Anwender zu liefern. Beim Setup müssen je nach Vorgaben des Benutzers alle Relationen oder nur bestimmte Stränge der Hierarchie in die globale Datenbasis des Benutzers übertragen werden. Die Ausarbeitung eines solchen Thesaurus wird zweckmäßigerweise vom gleichen Gremium übernommen, das für die Normierung der Relationen verantwortlich zeichnet. Letztlich sollte dieser Thesaurus ein Abfallprodukt der Normierungsbemühungen sein.

Eine womöglich in der Praxis zweckmäßigere Variante bestünde darin, daß der Anwender selbst einen entsprechenden Thesaurus aufbaut. Dies könnte quasi en passant geschehen. Ein entsprechendes Strukturierungsprogramm würde dann bei der ersten Anwendung keine externen Strukturinformationen vorfinden und die Relationen ungeordnet oder nach formalen Kriterien sortiert präsentieren. Das Pro-

[399] S. S. 129.

gramm kann nun dem Benutzer durch eine einfache Schalterleiste die Möglichkeit einräumen, die Fakten eines konkreten Falls hierarchisch anzuordnen. Eine für diese Zwecke gängige Schalterleiste zeigt der folgende Ausschnitt:

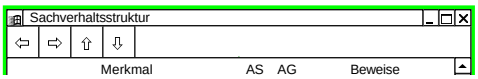


Abbildung 56: Benutzerschnittstelle zur individuellen Beeinflussung der Strukturierung von Fakten.

Mit Hilfe dieser Schalter oder ähnlicher Funktionalität lassen sich markierte Einträge vertikal verschieben und horizontal in eine andere hierarchische Ebene einordnen. Ein Programm, das diese Möglichkeit in einem konkreten Sachverhalt unterstützt, sollte nun in der Lage sein, aus den konkreten Veränderungen auch allgemeine Schlüsse auf die Einordnung der zugrundeliegenden Relationen zu ziehen. Verwendet der Anwender das Programm in einem späteren, ähnlich gelagerten Fall, so kann es die bereits bekannte Struktur auch zur Anordnung der Fakten des neuen Falls einsetzen.

Eine individuelle Erstellung von Strukturvorgaben schließt eine Vorgabe von Experten grundsätzlich nicht aus. Sie ist jedoch zum einen realistischer, da oftmals Normen lange auf sich warten lassen. Sie berücksichtigt zum anderen auch die Gewohnheiten des Anwenders. Sofern die allgemeinen Strukturinformationen durch die Arbeit an konkreten Fällen gewonnen werden, wird der Benutzer in der Erstellung dieser Vorgaben keine erhebliche Mehrbelastung sehen.

bb) Logische Verknüpfungen

Eine nahezu ideale Strukturierung von Sachverhalten ist möglich, wenn dem unterstützenden Programm echte juristische Intelligenz zur Verfügung steht[400]. Es könnte somit nicht nur für eine stets sinnvolle Anordnung der Fakten sorgen, sondern es bestünde auch die Möglichkeit, die juristischen Folgen der Sachlage zu visualisieren. Insbesondere könnte das Fehlen von Fakten für die Begründung eines Anspruchs oder das Fehlen von Beweisen für dessen Durchsetzung angezeigt werden. Geht man wiederum von einer hierarchischen Darstellung aus, so ist es bei der Verwendung von juristischen Regeln möglich, für jeden Knotenpunkt die aktuelle Beurteilung aus den unterschiedlichen Sichten anzuzeigen.

Was hier quasi unter dem Deckmantel der Sachverhaltsstrukturierung gefordert wird, ist tatsächlich ein System zur Unterstützung der Rechtsentscheidung. Es unterscheidet sich prinzipiell von bekannten subsumtionsunterstützenden Systemen nur durch die Darstellungsweise, die parallele Auswertung divergierender Aussagen und durch die Visualisierung der Beweismittel.

Viele der bisher veröffentlichten Systeme zur Subsumtionsunterstützung beruhen dabei auf einem aussagenlogischen Regelapparat[401]. Sie unterscheiden sich von einem Thesaurus lediglich durch die Verbindung der juristischen Begriffe mit logischen Operatoren. Der Benutzer tätigt eine Aussage dadurch, daß er dem Begriff einen logischen Wert zuweist. Die Konkretisierung der Aussage erfolgt dabei implizit in den Gedanken des Anwenders. Diese Systeme sind ähnlich wie der vorangehend beschriebene einfache Thesaurus nicht in der Lage, zwischen mehreren Beziehungen, die auf derselben Relation beruhen, zu differenzieren. Dieser Mangel ist zunächst unabhängig von der Aussagenlogik. Auch hier wäre es denkbar, die Aus-

[400] *Suhr*, a.a.O.; *Möller*, S. 105 ff.; *Kraft*, jur-pc 90, S. 496 ff.

[401] Vgl. FN. 400.

sagenvariablen etwa einer Regel $u \Rightarrow b0 \vee b1 \vee b2 \vee b3$ unterschiedlichen Beziehungen zuzuordnen. (Im System Atlas wurde die Behauptung der Existenz einer Beziehung oder Eigenschaft als Aussage bezeichnet.) Die Syntax der Regelbeschreibung ist jedoch weitaus ärmer als die Syntax eines prädikatenlogischen Ausdrucks. Sie reicht nicht aus, die unterschiedlichen Beziehungen automatisiert korrekt in das Regelwerk einzuordnen. Selbst wenn es möglich ist, der Aussagenvariablen u die Aussagen $Unterhaltsanspruch(U1, AS, AG)$ und $Unterhaltsanspruch(U2, AG, AS)$ zuzuordnen, ist es bereits nicht mehr möglich, die Aussage $Bedürfnis\_aus\_1570(B1, AS)$ zur Vervollständigung einer der Regeln automatisch zuzuordnen.

Zwar sind auch hier ähnliche Hilfslösungen denkbar, wie sie bereits im vorangegangenen Abschnitt beschrieben wurden. Sinnvolle und für alle Fallkonstellationen geeignete Strukturen lassen sich indes nur mit einem prädikatenlogischen Regelapparat erzielen. So bietet zum Beispiel der Ausdruck $\forall u \forall m \forall n(U(u, m, n) \Rightarrow \exists b(B0(b, m) \vee B1(b, m) \vee B2(b, m) \vee B3(b, m)))$ nicht nur durch seine unterschiedliche Quantifizierung die reichere logische Information. Aufgrund der Darstellung der Attribute als Variablen ist es nun möglich, die Fakten aus Atlas mit den entsprechend logischen Ausdrücken zur Deckung zu bringen. Die Probleme der Realisierung von Entscheidungshilfen auf Grundlage der Prädikatenlogik kann hier allerdings nicht abgearbeitet werden, so daß diese Variante lediglich als eine Anregung zu verstehen ist.

Grundlage der Strukturierung von Sachverhalten ist ein Strukturmodell, das nicht in Atlas unmittelbar, sondern als sekundäre Information gespeichert wird. Diese Information referenziert auf Relationen in der globalen Datenbasis des Anwenders. Atlas selbst liefert für die Sachverhaltsdarstellung die Fakten aus den unterschiedlichen Blickwinkeln und die Beweise für die entsprechenden Behauptungen.

4. Berechnungen

Wie bereits mehrfach dargelegt sind Berechnungsprogramme die wohl am häufigsten tatsächlich eingesetzten Anwendungen auf dem Tisch des Juristen. Allen voran im Familienrecht Programme zur Berechnung des Unterhalts. An dieser Stelle soll deshalb auf die Anbindung derartiger Programme an eine zentrale Datenschnittstelle eingegangen werden.

a) Maskenorientierte Programme

Wenn auch die praktische Bedeutung von Berechnungsprogrammen sehr hoch sein mag, so ist die technische Realisierung prinzipiell wenig spektakulär. Dem Benutzer präsentieren sich die Programme als eine Kollektion von Masken, die sowohl der Dateneingabe als auch der Ausgabe von Zwischen– und Endergebnisse dienen[402]. Sie unterscheiden sich damit zunächst nicht von der Maskeneingabe im beschriebenen Aufnahmebogen[403]. Für die technische Realisierung sind jedoch einige Unterschiede beachtenswert, die im folgenden aufgeführt werden:

aa) Kleine Szenarien

Berechnungsprogramme gehen oftmals von kleineren Detailszenarien aus, als dies beispielsweise der Aufnahmebogen tut. Legt dieser das Szenario eines kompletten familienrechtlichen Falls zugrunde, so mag ein Berechnungsprogramm nur für einen kleinen Ausschnitt, etwa der Brutto-Netto-Berechnung aller denkbarer Einkommen bestimmt sein. Ein Berechnungsprogramm wird also in einem aktuellen Fall nicht

---

[402] Eine Ausnahme bildet hier das Programm von Gutdeutsch. S.u.S. 219.

[403] S. S.177.

selten mehrere Konstellationen vorfinden, die seinem Grundszenario entsprechen. Das angeführte Programm zur Berechnung des Nettoeinkommens wird soviele Szenarien vorfinden, wie es Personen mit eigenem Einkommen in einem Fall gibt.

Wird das Programm einzeln gestartet, so wird es zunächst die passenden Konstellationen erfassen und dann dem Benutzer die Möglichkeit geben, eine der Konstellationen auszuwählen oder eine weitere hinzuzufügen. Das Programm sollte jedoch auch eine Möglichkeit vorsehen, daß ihm ein anderes Programm mitteilt, welche Konstellation abzuarbeiten ist. Dies ist dann sinnvoll, wenn ein Programm ein anderes zur Lösung eines Detailproblems aufruft. Dies kann auch dadurch geschehen, daß eine entsprechende Erweiterung in Atlas implementiert wird, die das Programm zur Lösung definierter Probleme aufruft. Technisch kann die Information, welche Fallkonstellation durch ein Programm bearbeitet werden soll, einfach durch einen Parameter in der Kommandozeile geschehen. Angenommen, ein Programm *NETTO.EXE* dient der Nettoeinkommensberechnung. In diesem Fall könnte der Aufruf NETTO Bruttoeinkommen_Peter das Programm beim Start darauf hinweisen, welches Bruttoeinkommen umzurechnen ist. Es wird nun anhand dieser Information die bereits bekannten Fakten zusammensuchen. Die Funktion *@AlleParameter*(*ausdruck*)[404] ist ein Weg für das Programm, weitere Objekte des Szenarios ausfindig zu machen. So wäre zum Beispiel eine Anfrage *@AlleParameter*(*Bruttogehalt*(*Bruttogehalt_Peter;*)) geeignet, den Bezieher des Gehalts festzustellen. Von hieraus können dann die Anzahl der Kinder, die weiteren Abzugsmöglichkeiten und so das gesamte Szenario ermittelt werden.

bb) Divergenzen in den Ergebnissen

Startet die Anwendung oder werden die Fakten etwa aufgrund einer neuerlichen Aktivierung des Programms aus Atlas aktualisiert, so muß dies in der Regel in einer definierten Reihenfolge geschehen. Diese Reihenfolge entspricht prinzipiell auch dem Datenstrom, den ein Berechnungsprogramm bei konventionellen Speichermethoden aus einer Datei ausliest. Eine Differenz zur individuellen Speichertechnik eines Programms entsteht indes daraus, daß in Atlas auch alle Ergebnisse und sogar Zwischenergebnisse einer Rechnung verfügbar sind. Bei einer Speicherung von Daten für lediglich ein Programm ist es unnötig, diese Ergebnisse zu speichern, da sie jederzeit wieder berechnet werden können. Hingegen werden im hier vorgestellten Konzept gerade diese Ergebnisse anderen Anwendungen zugänglich gemacht, die sie selbständig nicht berechnen können.

Dieser Zugang anderer Programme zu den Ergebnissen einer Berechnung beinhaltet auch die Möglichkeit, daß andere Programme diese Ergebnisse ändern. Das ist beispielsweise dann der Fall, wenn der Anwender zwei Programme einsetzt, die mit unterschiedlichen Algorithmen dasselbe Faktum zu ermitteln suchen. Er wird dies aus denselben Gründen tun, aus denen er auch sonst mehrere Rechtsmeinungen prüft. Ein Programm, das Daten aus Atlas bezieht, muß deshalb jederzeit damit rechnen, daß die eigenen Berechnungsergebnisse nicht mehr mit den von Atlas gelieferten Fakten übereinstimmen. Es wird also zweckmäßigerweise auch die eigentlich zu berechnenden Daten aus Atlas entnehmen und mit den eigenen Rechenergebnissen vergleichen. Diskrepanzen muß es dem Benutzer mitteilen. Hierzu wird es über die Funktion *@Info*(*ausdruck*)[405] die Quelle der abweichenden Fakten und die zugrundeliegende Anwendung ermitteln. Der Benutzer benötigt diese Angaben, um die Glaubhaftigkeit der einen oder anderen Information beurteilen zu können. Er

[404] S. S. 159.

[405] S. S. 160

muß die Möglichkeit haben, eines der beiden Ergebnisse anzunehmen. Lehnt der Benutzer das vom aktuellen Programm berechnete Ergebnis ab, so muß das Programm entweder beendet werden oder es paßt die Ausgangsfakten an das Ergebnis an oder es ändert eventuell die Berechnung der zugrundegelegte Parameter so, daß es nun ein korrektes Ergebnis erzielt.

Dieser Vorgang mag dann besonders kompliziert sein, wenn die Differenzen bereits im Zwischenergebnis liegen oder wenn aufgrund geänderter Fakten ein ganz anderer Rechenweg einzuschlagen ist. Angenommen der Benutzer steht bei der Berechnung des Unterhalts vor der Frage, wie hoch das Nettogehalt des Anspruchsgegners ist. Das aktuelle Programm öffnet eine eigene Maske, die zur Brutto-Netto-Berechnung dient. Der Benutzer hat jedoch eine Vorliebe für ein eigens erworbenes wesentlich spezialisierteres Programm. Er startet diese Anwendung und berechnet hiermit das gewünschte Nettogehalt. Bei der Rückkehr zum Unterhaltsprogramm muß dieses erfassen, daß die benötigten Zwischenwerte nun bekannt sind. Es muß die angezeigte Berechnungsmaske wieder entfernen und möglicherweise in einen ganz anderen Teil des Programmablaufs verzweigen. Das Programm sollte deshalb auf Änderungen des Sachverhalte auch von außen leicht reagieren können. Es bedarf hierzu möglicherweise einer intensiveren Führung als dies von maskengesteuerten Systemen allgemein bekannt ist.

b) Unterhaltsberechnung von Gutdeutsch[406]

Das Unterhaltsberechnungsprogramm von Gutdeutsch verfolgt einen anderen Weg als die meisten marktüblichen Programme. Es erfreut sich trotz seiner etwas antiquiert wirkenden Benutzeroberfläche großer Beliebtheit. Systembedingt wird es sich in der derzeit bekannten Fassung an einem Datenaustauschsystem der hier beschriebenen Art nicht beteiligen können. Dennoch ist es vom Konzept her letztlich benutzerfreundlicher als manch ein modernes Programm. Genau dieses Konzept ist es, das eine Betrachtung lohnt:

Im Gegensatz zu einem maskenorientierten System werden bei Gutdeutsch die Fakten in einer Sequenz abgefragt. Der Anwender hat zwar die Möglichkeit, die Sequenz zurückzuverfolgen, er kann jedoch nicht vorweg Angaben machen, die in der Abfolge zu einem späteren Zeitpunkt erfolgen. Die Besonderheit des Systems liegt in seiner Flexibilität und Effektivität. Das Programm entscheidet nämlich nach jeder Abfrage individuell, welches Faktum als folgendes relevant ist. Auf diese Weise können auch relevante sekundäre Informationen eingeholt werden, ohne daß der Benutzer jemals mit Fragen konfrontiert wird, die für die Problemlösung nicht zielführend sind. Die Ablage der erfaßten Fakten auf dem Massenspeicher erfolgt bei Gutdeutsch in derselben Sequenz, in der sie im Programm abgefragt werden. Werden die Fakten aus der Datei gelesen, so simuliert das Programm den Dialog genau bis zu dem Punkt, an dem die Abspeicherung erfolgte.

Diese Tugenden sind eine gute Voraussetzung für die Anbindung des Programms an Atlas[407]. Das Programm kann letztlich sehr einfach einen Dialog mit dem Benutzer führen und vor jeder Frage zunächst prüfen, ob das gesuchte Faktum im aktuellen Sachverhalt bereits bekannt ist. In diesem Fall verzichtet es auf die Interaktion mit dem Benutzer. Berechnet das Programm einen Zwischenwert, so hat es auch hier die Möglichkeit, einen eventuell im System verfügbaren Wert mit dem Rechenergebnis zu vergleichen. Bei einer Diskrepanz kann das Programm den Benutzer an der sy-

---

[406] Bilbl. Angaben.

[407] Wie bereits bemerkt, ist dies letztlich nicht möglich, da das Programm nur als DOS-Version verfügbar ist.

stematisch günstigsten Stelle informieren und den nachfolgenden Dialog an dem Ergebnis ausrichten.

Aktualisiert die Anwendung ihre Daten, etwa weil der Benutzer zwischendurch ein anderes Programm geöffnet hatte, so beginnt sie den Dialog im Hintergrund abermals. Sie setzt dort mit der Benutzerinteraktion wieder ein, wo ihr eine Information fehlt oder nicht den berechneten Erwartungen entspricht. Beantwortet der Benutzer die gestellte Frage, wird die Anwendung weiterhin zunächst versuchen die folgenden Informationen aus den bekannten Fakten zu ermitteln. Auf diese Weise ist es einfach, auf eine geänderte Sachlage jederzeit zu reagieren und jeweils eine optimale Dialogführung mit minimalem Aufwand für den Benutzer zu erzielen.

Berechnungsprogramme unterscheiden sich nur unwesentlich von elektronischen Formularen. Die Möglichkeit, Teile des Formularinhalts zu errechnen birgt die Gefahr, daß zwischen Rechenergebnis und bereits verfügbaren Fakten Divergenzen auftreten. Ein Programm muß diese Divergenzen lösen können. Programme, die anstelle von direkt und in beliebiger Reihenfolge zu manipulierenden Eingabefeldern ihre Fakten durch eine sequentielle Abfrage primär aus Atlas und sekundär beim Benutzer erfragen, können flexibler und effektiver auf geänderte oder individuelle Sachlagen reagieren, ohne den Benutzer zu überfordern.

5. Textverarbeitung

Im Endeffekt ist das Ziel der meisten juristischen Verarbeitungsprozesse die Erstellung eines Textdokuments. Es gehört dabei zu den Hauptanwendungsfällen des Computers, diese Texterstellung durch Textverarbeitungsprogramme zu unterstützen. Eine noch weitergehende Assistenz erfährt der juristische Anwender durch am Markt verfügbare Sammlungen von Mustertexten oder Textbausteinen, die einfach in ein Textverarbeitungsdokument übernommen werden können. Nachfolgend werden auf der Basis des wohl gängigsten Windows-Textverarbeitungsprogramms Microsoft Word für Windows Beispiele für die Einbindung von Fakten aus Atlas in ein Textverarbeitungsdokument gezeigt.

a) DDE-Feld

Die einfachste Form, Fakten aus Atlas in die Textverarbeitung zu übernehmen, erfolgt über ein DDE-Feld. WinWord erlaubt es, über sogenannte *Felder* Informationen in einem Text unterzubringen, die über den Feldtyp definiert werden. Dies können das Tagesdatum, der Autor oder Dateiname des Textes etc. sein. Ändern sich die Informationen, so wird der Inhalt des Feldes, d.h. der Text der anstelle des Feldes angezeigt wird, mit geändert. Das DDE-Feld erlaubt es nun, Informationen, die aufgrund einer innerhalb des Feldes definierten DDE-Anfrage übermittelt werden, im Text zu plazieren.

Ausgehend von den Relationen *Kläger*$(x, \mathit{Verfahren})$ und *Name*(x, Mensch) und der Relation kann man den Namen eines Klägers aus Atlas als DDE-Feld **{**DDE Atlas System Name(;Kläger())**}** im Text einfügen. Felder werden dabei in fetten geschweiften Klammern dargestellt. Der Feldtyp steht am Anfang des Feldes. Das DDE-Feld enthält Informationen über den Server, das Thema und das Element der Konversation. Als Element wird die entsprechende Atlas-Anfrage eingegeben. WinWord baut zur Aktualisierung des Feldinhaltes kurzzeitig eine DDE-Verbindung zur Anwendung Atlas auf. Es setzt die Anfrage ab und wartet auf die Antwort, um sie im Text darzustellen.

Wird dasselbe Faktum an mehreren Stellen im Text benötigt, mag es sinnvoll sein, die Information einmal von Atlas zu erfragen und sie dann an den unterschiedlichen Textstellen einzusetzen. Zu diesem Zweck wird die Antwort der Anfrage in eine Variable, eine sogenannte Textmarke geschrieben. Im Text wird nun eine Referenz

auf die Variable positioniert. Eine Kombination von Feldern wie **{**BESTIMMEN NameKläger **{**DDE Atlas System Name(;Kläger())**}}** weist der Textmarke *Kläger* den ermittelten Wert zu. Das Feld bleibt im Text unsichtbar. Innerhalb des Textes reicht nun ein Feld **{**NameKläger**}** aus, um den Namen als Feldergebnis einzusetzen. (Beim Referenzfeld **{**REF *Textmarke***}** kann der Ausdruck *REF* entfallen.) Dieses Verfahren erhöht die Lesbarkeit des Textes, da es störende technische Elemente fast völlig entfernt. Es verbessert zudem die Geschwindigkeit, da für jedes Faktum nur eine Anfrage an Atlas abgesetzt wird.

b) Textbausteine

Die Automation der Informationsübernahme ist nur dann wirklich sinnvoll, wenn sie Arbeit spart und Fehler vermeidet. Letzteres ergibt sich daraus, daß Fakten aus einer einheitlichen Quelle ohne menschliches Eingreifen übernommen werden. Eine Arbeitsersparnis ergibt sich bei einer Textverarbeitung meist dann, wenn oft benötigte Texte vorgefertigt abgelegt und einfach abrufbar sind. Beispielsweise ist es zu aufwendig, das oben beschriebene Feld in jedem Schriftsatz von Hand zu schreiben. Statt dessen könnte es durch eine Tastenkombination wie etwa *KL* leicht abgerufen werden.

Zur Verwaltung vorgefertigter Textpassagen dienen prinzipiell zwei Techniken, nämlich die Verwendung kompletter vorgefertigter Textmuster aus entsprechenden Textdateien und der Einsatz von einzelnen Textelementen, aus denen ein Schriftsatz zusammengestellt werden kann. Das hier eingesetzte Programm Word bietet mit sogenannten Dokumentvorlagen die Möglichkeit, vollständige Texte oder Textmuster so in einer Datei abzuspeichern, daß sie als Vorlage für einen neuen Text dienen. Beim Anlegen eines neuen Textes wird der Benutzer regelmäßig aufgefordert, eine Vorlage auszuwählen. Beim Abspeichern kann die Vorlage nicht versehentlich überschrieben werden. Eine solche Vorlage enthält die typischen einheitlichen Bestandteile eines Schriftsatzes wie Briefkopf und generelle Floskeln. Mit der Technik der sogenannten Textbausteine[408] hat der Anwender die Möglichkeit, ständig verwendete Floskeln zu sammeln, sie jeweils einem Schlüsselbegriff zuzuordnen und bei der Texterstellung in beliebiger Reihenfolge zu kombinieren. Dabei können Textbausteine entweder global, das heißt für Texte mit einer beliebigen Dokumentvorlage oder auch spezifisch für Texte mit einer bestimmten Vorlage definiert werden. Bei der Organisation der Textautomation sind diese Instrumente bedacht einzusetzen. So werden beispielsweise für die unterschiedlichen Rechtsgebiete unterschiedliche Dokumentvorlagen anzulegen sein. Die jeweils identischen Textbestandteile wie Briefkopf, Anrede- und Grußfloskel, Vertretungsberechtigung etc. werden in globalen Textbausteinen abgelegt. Textelemente, die rechtsbereichsspezifisch sind, also etwa nur bei Familienrechtsfällen benötigt werden, werden zusammen mit der entsprechenden Dokumentvorlage gespeichert.

Informationen aus Atlas können nun dazu dienen, Fakten wie bereits gezeigt, im Text zu plazieren. Sie können indes auch, wie schon bei der Analyse gedruckter Lösungen dargelegt[409], zur Auswahl der eingesetzten Textbausteine oder zur Bestimmung der Anzahl eines wiederholenden Bausteins eingesetzt werden. Technisch wird der Einsatz eines Textbausteins durch das Feld **{**BAUSTEIN *Textbaustein***}** in einen Text fest eingefügt. Für die Textautomation kann das Feld insbesondere in

---

[408] In der aktuellen Version 7 des Programms WinWord wird diese Funtkion als Autotext bezeichnet. Diese neue Bezeichnung scheint mir mehr marktpolitische als tatsächlich sachliche Gründe zu haben.

[409] S. S. 65 ff.

Verbindung mit einem Bedingungsfeld **{**WENN *Bedingung DannText SonstText***}** eingesetzt werden[410]. Der Inhalt dieses Feldes ist abhängig von der gesetzten Bedingung. Als Bedingung kann etwa eine Information aus Atlas dienen, die je nach ihrem Wert einen anderen Textbaustein aufruft. Zu Beginn eines Textes seien beispielsweise die Textmarken *Kläger* und *Mandant* mit den Bezeichnern der entsprechenden Atlas-Objekte belegt worden: **{**BESTIMMEN Kläger **{**DDE Atlas System @Bezeichner(;Kläger())**}}**, **{**BESTIMMEN Mandant **{**DDE Atlas System @Bezeichner(;Mandant())**}}**. Nun kann in Abhängigkeit davon, ob der Mandant der Kläger oder der Beklagte ist, ein anderer Textbaustein für die Einleitung des Schriftsatzes gewählt werden: **{**WENN **{**Mandant**}** = **{**Kläger**}** **{**BAUSTEIN KopfKl**}** **{**BAUSTEIN KopfBkl**}}**. Im Text erscheint an der Stelle dieses Feldes der Inhalt des Textbausteins *KopfKl*, wenn der Mandant der Kläger ist, ansonsten erscheint der Inhalt des Bausteins *KopfBkl*. Ein solches Feld kann nun entweder fest in einer Dokumentvorlage plaziert werden oder in einen weiteren Textbaustein eingebettet sein. Ebenso können in den angesprochenen Bausteinen *KopfKl* und *KopfBkl* wiederum Verweise auf Fakten aus Atlas eingebettet sein. Eine etwas einfachere Entscheidung zeigt der folgende Textbaustein:

zeige ich an, daß ich den **{**WENN **{**Mandant**}** = **{**Kläger**}** "Kläger" "Beklagten"**}** anwaltlich vertrete.

Wollte man nun das Geschlecht korrekt wiedergeben, so müßte in einem weiteren Feld, eine entsprechende Bedingung vor- oder nachgeschaltet werden. Im familienrechtlichen Szenario reicht regelmäßig als Anhaltspunkt für das Geschlecht die Relation *Ehemann*(*x,ehe*) beziehungsweise *Ehefrau*(*x,ehe*) aus. In allgemeiner gelagerten Fällen sollte eine eigens benötigte Eigenschaft *Mann*(*x*) oder *Frau*(*x*) ausgewertet werden:

zeige ich an, daß ich **{**WENN **{**Mandant**}** = **{**Ehefrau**}** **{**WENN **{**Mandant**}** = **{**Kläger**}** "die Klägerin" "die Beklagte"**}** **{**WENN **{**Mandant**}** = **{**Kläger**}** "den Kläger" "den Beklagten"**}}** anwaltlich vertrete.

Hierzu muß vorab die Textmarke *Ehefrau* mit dem Bezeichner der Ehefrau des Falls belegt werden (**{**BESTIMMEN Ehefrau **{**DDE Atlas System @Bezeichner(;Ehefrau())**}}**). Das folgende Muster zeigt einen etwas längeren Ausschnitt des bereits beschriebenen Musterformulars[411].

{BESTIMMEN Mandant **{**DDE Atlas System @Bezeichner(;Mandant())**}}**
{BESTIMMEN Ehemann **{**DDE Atlas System @Bezeichner(;Ehemann())**}}**
{BESTIMMEN Eheschließung **{**DDE Atlas System @Bezeichner(;Eheschließung())**}}**
{BESTIMMEN Gericht **{**DDE Atlas System @Bezeichner(zust_Gericht())**}}**
{BESTIMMEN GerichtOrt **{**DDE Atlas System Ort(;zust_Gericht())**}}**
{BESTIMMEN MBeruf **{**DDE Atlas System Beruf(;Mandant())**}}**
{BESTIMMEN MVorname **{**DDE Atlas System Vorname(;Mandant())**}}**
{BESTIMMEN MNachname **{**DDE Atlas System Nachname(;Mandant())**}}**
{BESTIMMEN MAnschrift "**{**DDE Atlas System Strasse(;Adresse(;Mandant()))**}**, **{**DDE Atlas System PLZ(;Adresse(;Mandant()))**}** **{**DDE Atlas System Ort(;Adresse(;Mandant()))**}**"**}**

---

[410] Eine bedingte Textbausteinkombination mittels Steuerung durch ein externes Programm bei *Kraft*, jur-pc 93, 1977 ff. Ein Ansatz mittels eines Parsings eines Textes bei *Herberger*, jur-pc 89, 116 ff.

[411] *Vespermann* S. 6 ff.

{BESTIMMEN GBeruf **{**DDE Atlas System Beruf(;Gegner())**}}**
{BESTIMMEN GVorname **{**DDE Atlas System Vorname(;Gegner())**}}**
{BESTIMMEN GNachname **{**DDE Atlas System Nachname(;Gegner())**}}**
{BESTIMMEN GAnschrift "**{**DDE Atlas System
Strasse(;Adresse(;Gegner()))**}**, **{**DDE Atlas System
PLZ(;Adresse(;Mandant()))**}** **{**DDE Atlas System Ort(;Adresse(;Gegner()))**}**"**}**
{BESTIMMEN RAGNachname **{**DDE Atlas System
Nachname(;Anwalt(;Gegner()))**}}**
{BESTIMMEN RAGOrt **{**DDE Atlas System
Ort(;Adresse(;Anwalt(;Gegner())))**}}**

Amtsgericht
- Familiengericht -
{GerichtOrt**}**

ANTRAG (SCH)

In Sachen

{WENN **{**Mandant**}**=**{**Ehemann**}** "des" "der"**}** **{**MBeruf**}**, **{**MVorname**}**
{MNachname**}**, **{**MAnschrift**}**

antragstellender Ehegatte

Verfahrensbevollmächtigter: RA Mustermann, Neustadt

gegen

{WENN **{**Mandant**}**=**{**Ehemann**}** "der" "des"**}** **{**GBeruf**}**, **{**GVorname**}**
{GNachname**}**, **{**GAnschrift**}**

gegnerischer Ehegatte

Verfahrensbevollmächtigter: RA **{**RAGNachname**}**, **{**RAGOrt**}**

wegen streitiger Scheidung der Ehe,
vorläufiger Streitwert: DM **{**DDE Atlas System Streitwert()**}**
wird in **{**WENN **{**DDE Atlas System @Bezeichner(Vollmacht())**}** = ""
"nachzureichender" "anliegender"**}** besonderer Vollmacht i.S. des § 609 ZPO
beantragt,

> Die vor dem Standesbeamten in **{**DDE Atlas System
> Ort(;$**{**Eheschließung**})}** am **{**DDE Atlas System
> @Datum($**{**Eheschließung**})}** unter der Registernummer **{**DDE
> Atlas System Registernummer(;$**{**Eheschließung**})}** geschlossene
> Ehe der Parteien wird geschieden.

Begründung:

Die Eheleute haben, wie im Antrag ausgeführt, die Ehe geschlossen.
Beweis: **{**DDE Atlas System Beweisurkunde(;@Beweisobjekt(Ehe()))**}**

{BAUSTEIN F6N03**}**

{WENN **{**DDE Atlas System Staatsangehörigkeit(;Ehemann())**}** = "deutsch"
{WENN **{**DDE Atlas System Staatsangehörigkeit(;Ehefrau())**}** = "deutsch"
{BAUSTEIN F6N04**}** **{**BAUSTEIN F6N06**}}** **{**BAUSTEIN F6N06**}}**

{WENN **{**DDE Atlas System "@Bezeichner(zust_Gericht_gem_606_1_1())"**}**
= **{**Gericht**}** **{**BAUSTEIN F6N08**}** **{**WENN **{**DDE Atlas System
@Bezeichner(zust_Gericht_gem_606_1_2())**}** = **{**Gericht**}** **{**WENN **{**DDE
Atlas System @Anzahl(ehel_Kind())**}** = "1" **{**BAUSTEIN F6N09**}** **{**BAUSTEIN
F6N010**}}** **{**BAUSTEIN F6N011**}}}**

...

Das Beispiel enthält zu Demonstrationszwecken einige unterschiedliche Lösungsansätze. So wird ein großer Teil der benötigten Fakten zu Beginn des Dokuments in entsprechende Textmarken eingetragen. Einige Fakten werden erst innerhalb des Textes ermittelt. Für die Ermittlung der zusätzlichen Angaben zur Eheschließung wird dabei in die Anfragen das Ergebnis einer vorangegangenen Anfrage als Textmarke eingetragen. Die Angabe des Beweises der Ehe erfolgt über die Atlas-Funktion *@Beweisobjekt*. Hier hätte auch eine Relation wie *Heiratsurkunde*(*x, ehe*) eingesetzt werden können, da es nur wenige sinnvolle Beweise für eine Ehe gibt.

Die Auswahl der Begründungsfloskeln erfolgt mit Hilfe komplex verschachtelter Bedingungsausdrücke. Dabei wird für die Begründung der örtlichen Zuständigkeit darauf aufgebaut, daß bereits im Vorfeld bei der Prüfung der Relation *zust_Gericht*(*x*) abgeklärt wurde, auf welcher Vorschrift die Zuständigkeit beruht. Insofern weichen die Bedingungen für die Auswahl der Bausteine vom zugrundeliegenden Druckwerk ab.

c) Wiederholende Fakten

Wenn auch das gezeigte Schriftsatzmuster bereits recht individuelle Zusammenhänge abzudecken vermag, so sind Grenzen dort gesetzt, wo bestimmte Fakten mit beliebiger Wiederholung auftreten können. Ein einfaches Beispiel wäre die Aufzählung der Kinder aus einer Ehe mit ihren Namen und womöglich auch einer Forderung, die ein Elternteil in gesetzlicher Prozeßstandschaft geltend macht. Hier ist es sinnvoll, sich der Programmiersprache der Textverarbeitung zu bedienen. Ein solches Programm (Bei Standardanwendungen wird meist noch der Name *Makro* verwendet. Tatsächlich handelt es sich aber um ganz gewöhnliche BASIC-Programme.) muß zunächst die Anzahl der Wiederholungen ermitteln und danach im einfachsten Fall eine entsprechende Anzahl von Textbausteinen einfügen. Damit jeder Textbaustein in der Lage ist, zu erkennen, für welches Kind er die Fakten benötigt, muß ihm eine Ordnungsziffer dieses Kindes mitgeteilt werden. Diese setzt er ein, um eine Relation *Eheliches_Kind*(*x, Ehe*) über die Positionsparameter zu präzisieren (*Eheliches_Kind*(*;;@Pos*(*n*))). Der folgende Textbaustein nutzt das Feld **{**SEQ *Name***}**, um zu Beginn des Bausteins die laufende Nummer festzulegen. Dieses Feld erhöht seinen Wert jedesmal, wenn es im Text mit demselben Sequenznamen auftritt.

{Bestimmen KindNr **{**SEQ AdrKind**}}{**Bestimmen AktuellesKind "**{**DDE Atlas System "@Bezeichner(Eheliches_Kind(;;@Pos(**{**KindNr**}**)))"**}**"**}**Das **{**KindNr**}**. Kind aus der Ehe, **{**DDE Atlas System "Vorname(;$**{**AktuellesKind**}**)"**} {**DDE Atlas System "Nachname(;$**{**AktuellesKind**}**)"**}**, wohnaft in **{**DDE Atlas System "Ort(;Adresse(;(;$**{**AktuellesKind**}**)))"**}**...

Die Nummer des im folgenden Text gemeinten Kindes wird in der Marke *KindNr* eingefroren. Über sie wird zunächst der Bezeichner des Kindes abgerufen und in die Textmarke *AktuellesKind* eingetragen. Der Inhalt dieser Textmarke wird nun für die weiteren Anfragen nach Vor- und Nachname sowie Adresse eingesetzt. Schreibt man denselben Textbaustein für jedes Kind einmal in den Text, so werden die Fakten aller Kinder abgerufen und dargestellt. Dabei ist wichtig, daß dieses Verfahren nur auf der Eigenart des zugrundeliegenden Textverarbeitungsprogramms beruht, die Bestimmen-Felder sequentiell zu aktualisieren. Das heißt, der Inhalt einer mit einem Bestimmen–Feld definierten Textmarke gilt nur für Referenzen rechts von diesem Feld und links von einem womöglich folgenden Feld, das den Inhalt wieder verändert. Dies vereinfacht die Arbeit erheblich. Ein Programm, daß ein Muster auf die individuelle Anzahl der Kinder anpaßt, beschränkt sich hierdurch auf wenige

Zeilen, in denen eine zu definierende Anzahl von Textbausteinen hintereinander eingefügt wird:

```
' Gehe zunächst zu dem Punkt, an dem die Adressen plaziert werden sollen.
BearbeitenGeheZu "AdressenKinder"

' Die Anzahl der Kinder sollte im Text stehen.
' Der entsprechende Text muß mit der Textmarke "AnzahlKinder" markiert sein.
' Alternativ kann das Programm auch über DDE selbst nach der Anzahl der Kinder fragen!
AnzKinder = Val(AbrufenTextmarke("AnzahlKinder"))

Nun werden die Textbausteine AnzKinder-mal hintereinander im Text eingefügt.
For x = 1 To AnzKinder
  BearbeitenTextbaustein "AdresseKind", .Einfügen
Next x
```

Ein solcher Makro kann etwa beim Erstellen eines neuen Dokuments (*AutoNew*) oder durch Auslösen einer Schaltfläche gestartet werden. Eine derartige Schaltfläche kann auch Bestandteil des Textes selbst oder eines Textbausteines sein. Sollten bei jedem Kind andere Texte benötigt werden, so kann dies der Textbaustein selbst über Bedingungsfelder steuern oder der Makro sammelt noch einige weitere Fakten und wählt anhand derer einen entsprechenden Textbaustein aus.

Fakten aus dem entwickelten System lassen sich auf unterschiedliche Weise in Texte einer Textverarbeitung einbinden. Neben einer Referenz auf die Fakten selbst und die einschlägigen Beweisangebote können Atlas–Fakten beispielsweise in Bedingungsfeldern individuelle, auf die Sachlage angepaßte Formulierungen auswählen. Über die den meisten Textverarbeitungen immanente Makroprogrammierung können zudem Textbausteine in Abhängigkeit von Fakten auch wiederholend in einem Text plaziert werden. So lassen sich praktisch anhand entsprechender, intelligent aufbereiteter Textmuster sehr sachverhaltsspezifische Texte erstellen.

Teil IV. Schlußwort und Ausblick

In den vorangegangenen Teilen wurde über die Grundlagen des Austausches von Fakten unter besonderer Beachtung juristischer Spezialprobleme gesprochen. Dieser letzte Teil wird zunächst kurz den Stand darlegen, wie er sich anhand der im Rahmen dieser Arbeit entwickelten Programme ergibt. Er wird des weiteren eine Vision entwickeln, wie weitere Schritte der Integration eines juristischen Arbeitsplatzes aussehen können, nicht ohne eine Prognose der Realisierbarkeit zu wagen.

A. Technische Ergebnisse der Arbeit

Nicht alle der beschriebenen Programme sind überhaupt oder wenigstens nicht in der Form realisiert worden, in der sie im vorangegangenen Teil beschrieben wurden. Es mag auch nicht Sinn und Zweck der Arbeit sein, ein Ergebnis zu liefern, das sofort praxistauglich einsetzbar ist. Vielmehr handelt es sich um Prototypen, die einen Eindruck von den Möglichkeiten vermitteln und empirisch darlegen, daß die Forderungen der Arbeit nicht völlig aus der Luft gegriffen sind. Die folgenden Abschnitte beschreiben kurz den Ist-Stand der Programme immer als Differenz zum postulierten Soll.

1. Atlas 1

Anfängliche Experimente erfolgten mit einer Zentraleinheit *Atlas*, die keine Parameter zuließ. Die einzelnen Informationsträger werden hier als *Begriffe* bezeichnet. Eine globale Datenbasis enthält alle Begriffe. Die Begriffe haben wie klassische Variablen alternativ einen Typ aus den möglichen Wertetypen Boole, Zahl, Datum oder Zeichenkette. Der Zugriff erfolgt ausschließlich über DDE mit einem dem Grundprinzip angemessenen eingeschränkten Befehlssatz. Die Namen der Befehle können frei konfiguriert werden, so daß sie auch in andere Sprachen zu übersetzen sind[412]. Das System besteht aus einzelnen Programmen: Der DDE-Server dient als Schnittstelle zu anderen Programmen. Ein eigenes Programm ermöglicht die Pflege der globalen Datenbasis. Ein weiteres Programm bietet über DDE einen allgemeinen Zugriff auf die Fakten des Falls. Die Programme sind in Turbo-Pascal für Windows entwickelt.

Auf die Schnittstelle zugegriffen wurde experimentell mit dem DDE-Terminal und einigen Standardanwendungen wie Word für Windows, Excel sowie Object Vision[413]. Für das innerhalb eines Projektes zum UN-Kaufrecht entwickelte Programm *Checkliste* dient Atlas 1 als primärer Datenserver.

2. Atlas 2

Da ein System ohne Parameter nicht einmal eindimensionale Datenfelder zuläßt, wie dies z.B. beim Vorhandensein von mehreren Kindern in einem Sachverhalt benötigt wird, ist das System *Atlas 1* letztlich nur sehr eingeschränkt einsatzfähig. Dies führte zur Entwicklung des Systems *Atlas 2*. Es entspricht weitgehend den Postulaten dieser Arbeit.

Das System erlaubt die Beschreibung von Fakten anhand einer globalen Liste von Relationen mit mindestens einem und bis zu 32 Parametern. Als Datenträger dienen

---

[412] Diese Option wurde nötig, da das System in einem Projekt zum gemeinsamen UN-Kaufrecht *CISG* eingesetzt wurde.

[413] Dieses an sich recht leistungsfähige Entwicklungssystem der Firma Borland war zwischen Tabellenkalkulationsprgrammen und visuellen Entwicklungssystemen positioniert. Es ist inzwischen nicht mehr verfügbar und kann leicht durch Visual BASIC ersetzt werden.

beliebig instantiierbare Objekte. Ihnen können beliebig viele Werte der beschriebenen Datentypen zugeordnet werden. Weiterhin lassen sich anhand der Relationen beliebig viele Beziehungen zwischen Objekten instantiieren. Werte von Objekten und ihre Beziehungen untereinander werden als Fakten angesehen. Zu jedem Faktum werden Informationen über Quelle der Behauptung, Geltungsbereich und Eingabezeitpunkt abgelegt. Die Programmquelle, die das Faktum übermittelt hat, wird im Gegensatz zum Postulat nicht abgelegt[414]. Ebenfalls nicht möglich ist die Behandlung eines Faktums als Beweisobjekt[415].

Grundlage von Atlas 2 ist das Programm ATLASSV2.EXE. Das Programm wurde in Turbo-Pascal für Windows entwickelt. Es ermöglicht die Pflege der globalen Datenbasis und die Überwachung der einzelnen Datenbestände. Weiterhin stellt es die DDE-Schnittstelle für andere Programme zur Verfügung. Das Modul zur physikalischen Verwaltung der Datenbasis ist wie gefordert austauschbar. Das Beispiel verwaltet die Daten im Arbeitsspeicher. Dabei sind die einzelnen Datenbereiche auf eine Größe von jeweils ca. 65 KBytes begrenzt. Für die Praxis ist dies untauglich, zu Demonstrationszwecken allerdings vollkommen ausreichend. Am Ende einer Sitzung oder auf DDE–Befehl hin werden die globalen und die fallbezogenen Daten in einer Datei abgespeichert. Der Dateiname der globalen Datenbasis kann in eine Initialisierungsdatei eingetragen werden.

Abbildung 57: Das Programm *ATLASSV2*.

Syntax, Funktionsumfang und Verhalten entsprechen im wesentlichen ebenfalls den Beschreibungen der Spezifikation. Die Namen der Systemfunktionen sind in einer Initialisierungsdatei konfigurierbar. Die Systemeigenschaft *@Objekt*(x) ist nicht vollständig implementiert. Nicht realisiert ist die Funktion *@Beweisobjekt*(x). Beweise müssen statt dessen in Beziehung zu den Objekten gesetzt werden, deren Werte zu beweisen sind.

Der Funktionsumfang von Atlas 2 ist über die beschriebene Schnittstelle erweiterbar. Als Beispiel steht ein Modul zur Verfügung, das alle Anfragen, die es erhält, in einer Botschaft am Bildschirm anzeigt.

[414] Vgl. S. 109.

[415] Vgl. S. 156.

3. Eingabeassistent

Der Eingabeassistent *ASS01.EXE* ist ein Programm zur Eingabe von Daten eines Falls. Die Daten werden in mehreren Datenblättern sequentiell abgefragt. Jedes Datenblatt enthält eine Gruppe von Eingabemöglichkeiten, wie etwa die Grunddaten eines Szenarios oder die persönlichen Daten eines Beteiligten. Letztlich handelt es sich um eine Adaptation des Gedankens des *Aufnahmebogens* allerdings in einer bildschirmgerechteren Gestaltungsweise.

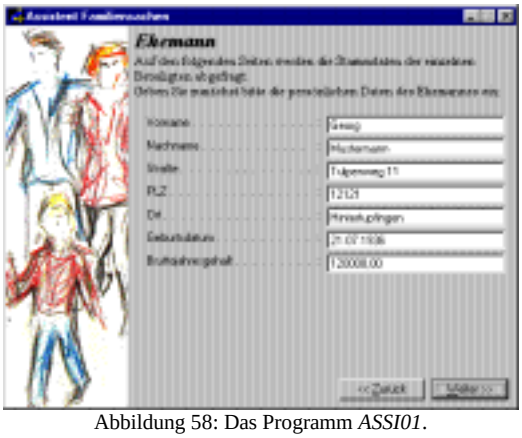


Abbildung 58: Das Programm *ASSI01*.

Das Programm ist skriptgesteuert. Das Skript ist in einer Datei abgelegt. Der Name des Skripts kann dem Programm beim Start übergeben werden. Das folgende Beispielskript zeigt 6 Erfassungsmasken für die Aufnahme der einfachen Daten bei einem Familienrechtsfall:

```
[Ablauf]
Titel=Assistent Familiensachen
Titelbild=famr.bmp
1=Szenario
2=Akte
3=Ehemann
4=Ehefrau
5=Sache
6=Ehe
7=Eheliches Kind

[Szenario]
Blatt=Makro
Erklärung={\rtf1\ansi{\b\fs24 Szenario}\par Bitte warten, das Grundszenario wird aufgebaut . . .}
1=Text\ja\Ehe (ID)\@ObjektID(Ehe())\Ehe\
2=Text\ja\Ehemann (ID)\@ObjektID(Ehemann())\Ehemann\
3=Text\ja\Ehefrau (ID)\@ObjektID(Ehefrau())\Ehefrau\
4=Auswahl\ja\%1 ist Ehe zwischen %2 und %3\@Beziehung(Ehe($%1;$%2;$%3))\ja
5=Auswahl\ja\%2 ist Ehemann\@Beziehung(Ehemann($%2;$%1))\ja
6=Auswahl\ja\%3 ist Ehefrau\@Beziehung(Ehefrau($%3;$%1))\ja

[Akte]
Blatt=Einfach
Erklärung={\rtf1\ansi{\b\fs24 Daten der Akte}\par Geben Sie im folgenden die Stammdaten der Akte ein:}
1=Text\ja\Aktenzeichen\Aktenzeichen(;Sache())
2=Gruppe\ja\Mandant ist die Ehefrau\@Beziehung(Mandant(Ehefrau();Sache()))\ja
3=Gruppe\ja\Mandant ist der Ehemann\@Beziehung(Mandant(Ehemann();Sache()))

[Ehemann]
Blatt=Einfach
Erklärung={\rtf1\ansi{\b\fs24 Ehemann}\par Geben Sie im folgenden die Stammdaten des Ehemannes ein:}
Objekttyp=Mensch\Ehemann()
```

```
[Ehefrau]
Blatt=Einfach
 Erklärung={\rtf1\ansi{\b\fs24 Ehefrau}\par Geben Sie im folgenden die Stammdaten der Ehefrau ein:}
Objekttyp=Mensch\Ehefrau()

[Sache]
Blatt=Einfach
Erklärung={\rtf1\ansi{\b\fs24 Sachlage}\par Geben Sie im folgenden die Daten des Verfahrens ein:}
1=Gruppe\ja\Mandant ist der Antragsteller\@Beziehung(Antragsteller(Mandant(;Sache());Sache()))\Ja
2=Gruppe\ja\Mandant ist der Antragsgegner\@Beziehung(Antragsgegner(Mandant(;Sache());Sache()))

[Kind]
Blatt=Wiederholen\@Anzahl(EhelichesKind())
NeuesObjekt=Möchten sie die Daten für ein (weiteres) eheliches Kind eingeben?
Erklärung={\rtf1\ansi{{\fs30\b\i Kind}\par{\fs20 Geben Sie hier Informationen zu den einzelnen Kindern aus der
streitgen Ehe ein.}}}
Objekttyp=Mensch\EhelichesKind(;Ehe();@Pos(%S))
I1=Text\ja\Kind %S (ID)\@ObjektID(EhelichesKind(;Ehe;@Pos(%S)))\Kind %S\
I2=Auswahl\ja\\@Beziehung(EhelichesKind($%1;Ehe()))\ja[Mensch]
1=Text\nein\Vorname\Vorname(;%O)
2=Text\ja\Nachname\Nachname(;%O)
3=Text\nein\Straße\Strasse(;Adresse(;%O))
4=Text\nein\PLZ\PLZ(;Adresse(;%O))
5=Text\nein\Ort\Ort(;Adresse(;%O))
```

Im Abschnitt *Ablauf* werden die einzelnen Eingabemasken - hier als *Blätter* bezeichnet - angegeben. Diese Masken werden der Reihe nach dem Benutzer zur Eingabe vorgelegt. Jedes Blatt bildet einen weiteren Abschnitt. Der Eintrag *Blatt* definiert die Art des jeweiligen Blattes. *Makros* werden nicht angezeigt. Sie dienen der Initialisierung von Szenarien. Blätter vom Typ *Einfach* werden genau einmal angezeigt. Blätter vom Typ *Wiederholen* ermöglichen die beliebig häufige Wiederholung des selben Blattes bezogen auf unterschiedliche Objekte. Hierzu stehen zwei weitere Schalter zur Verfügung. Das System ermittelt beim Anzeigen eines Blattes die Anzahl der entsprechenden Objekte durch die Abfrage nach `Wiederholen` (im Beispiel `@Anzahl(EhelichesKind())`). Wird ein Blatt mehr als die ermittelte Anzahl aufgerufen, so wird der Benutzer gefragt, ob er ein neues Objekt anlegen möchte. Der Platzhalter *%S* wird in allen Anweisungen durch die Nummer des Blattes in der Folge ersetzt.

Die Eingabeelemente werden durch Ziffern 1-8 gesteuert. Die folgende Zeichenkette enthält Informationen über die Art und Beschriftung des Eingabeelements, die Notwendigkeit der Eingabe, die Beschreibung des Faktums nach Atlas-Konventionen sowie den Vorgabewert. Platzhalter *%1* bis *%7* erlauben die Übernahme eines zuvor ermittelten Wertes in eine nachfolgende Zeichenkette. Zur Vermeidung von Wiederholungen kann von einem Blatt auf eine generelle Definition der Eingabeelemente verwiesen werden. Der Eintrag *Objekttyp* bezeichnet dazu einen weiteren Abschnitt sowie die Beschreibung des jeweiligen Objekts nach Atlas-Konventionen (Im Beispiel wird mehrfach auf den Abschnitt *Mensch* zurückgegriffen.). Der Platzhalter *%O* in den Angaben des verwiesenen Abschnitts nimmt die im Eintrag *Objekttyp* des verweisenden Abschnitts angegebene Beschreibung des konkreten Objekts auf.

Die Mischung aus Maskenabfrage und sequentieller Abfolge der Einzelmasken bietet einen Kompromiß zwischen direkter, quasi nicht geführter Datenmanipulation durch den Benutzer und gezielter Benutzerführung. Dabei wird berücksichtigt, daß komplexe Sachverhalte meist ohne eine Benutzerführung kaum vollständig zu erfassen sind. Andererseits führt eine zu strenge sequentielle Abfrage zu oftmals langatmigen Abfragefolgen, die das vorausschauende Fachwissen des Anwenders unberücksichtigt lassen.

Die Skriptsteuerung erleichtert nicht nur den Entwurf des Programmes, sie ermöglicht auch dem Leser dieser Arbeit, experimentell weitere Eingabefolgen zu schreiben und diese auch auf andere Szenarien anzupassen. Das Programm wurde in der 32-Bit Version von Visual–BASIC 4.0 entwickelt. Die Programmquellen befinden sich auf der Diskette.

4. Schnellübersicht

Das Programm *SZENARIO.EXE* dient der hierarchischen Darstellung von Fakten eines Szenarios. Dabei können zwei unterschiedliche Ansichten über denselben Fall gegenüber gestellt werden.

Abbildung 59: Das Programm *SZENARIO*.
Zu sehen sind die unterschiedlichen Behauptungen zur Frage des Einkommens der Ehefrau.

Das Programm ermöglicht die Anordnung von Objektbeschreibungen in einem hierarchischen Baum. Der Benutzer hat die Möglichkeit, in diesem Baum durch die einzelnen Objekte des Szenarios zu manövrieren. Zu jedem Eintrag erhält er Informationen darüber, welchen Bezeichner das entsprechende Objekt hat und welche Werte ihm zugeordnet sind. Die Darstellung erfolgt in zwei Spalten. Jeder Spalte kann ein beliebiger Ausgabefilter zugeordnet werden. Auf diese Weise nimmt das Programm auch rudimentäre Aufgaben der weiter oben postulierten Relation wahr.

Abbildung 60: Optionsdialog zum Einstellen zweier Ansichten.

SZENARIO wird wie der im vorigen Abschnitt dargestellte Eingabeassistent durch ein Skript gesteuert. Die Skriptsprache ist jedoch erheblich einfacher. Sie stellt lediglich einen linear konstruierten Baum dar, der das Navigieren in der Fakten ermöglicht. Das folgende Skript beschreibt ein Szenario *Ehe* mit den Personen und ihren Adreßdaten (Die Einrückungen dienen nur der besseren Lesbarkeit):

```
1,"Sache","Sache()",""
2,   "Aktenzeichen","Aktenzeichen()",""
2,   "Mandant","Mandant()",""
3,      "Vorname","Vorname(;Mandant())",""
3,      "Nachname","Nachname(;Mandant())",""
3,      "Geburtsdatum","Geburtsdatum(;Mandant())",""
3,      "Adresse","Adresse(;Mandant())",""
4,         "Straße","Strasse(;Adresse(;Mandant()))",""
4,         "PLZ","PLZ(;Adresse(;Mandant()))",""
4,         "Ort","Ort(;Adresse(;Mandant()))",""
2,   "Gegner","Gegner()",""
3,      "Vorname","Vorname(;Gegner())",""
3,      "Nachname","Nachname(;Gegner())",""
3,      "Geburtsdatum","Geburtsdatum(;Gegner())",""
3,      "Adresse","Adresse(;Gegner())",""
4,         "Straße","Strasse(;Adresse(;Gegner()))",""
4,         "PLZ","PLZ(;Adresse(;Gegner()))",""
4,         "Ort","Ort(;Adresse(;Gegner()))",""
1,"Ehe","Ehe()",""
2,   "Ehemann","Ehemann()",""
3,      "Vorname","Vorname(;Ehemann())",""
3,      "Nachname","Nachname(;Ehemann())",""
3,      "Geburtsdatum","Geburtsdatum(;Ehemann())",""
3,      "Adresse","Adresse(;Ehemann())",""
4,         "Straße","Strasse(;Adresse(;Ehemann()))",""
4,         "PLZ","PLZ(;Adresse(;Ehemann()))",""
4,         "Ort","Ort(;Adresse(;Ehemann()))",""
2,   "Ehefrau","Ehefrau()",""
3,      "Vorname","Vorname(;Ehefrau())",""
3,      "Nachname","Nachname(;Ehefrau())",""
3,      "Geburtsdatum","Geburtsdatum(;Ehefrau())",""
3,      "Adresse","Adresse(;Ehefrau())",""
4,         "Straße","Strasse(;Adresse(;Ehefrau()))",""
4,         "PLZ","PLZ(;Adresse(;Ehefrau()))",""
4,         "Ort","Ort(;Adresse(;Ehefrau()))",""
2,   "Eheliche Kinder","@Anzahl(EhelichesKind())",""
3,      "Kind %i","EhelichesKind(;;@Pos(%i))","%i"
4,         "Vorname","Vorname(;EhelichesKind(;;@Pos(%i)))",""
4,         "Nachname","Nachname(;EhelichesKind(;;@Pos(%i)))",""
4,         "Adresse","Adresse(;EhelichesKind(;;@Pos(%i)))",""
5,            "Straße","Strasse(;Adresse(;EhelichesKind(;;@Pos(%i))))",""
5,            "PLZ","PLZ(;Adresse(;EhelichesKind(;;@Pos(%i))))",""
5,            "Ort","Ort(;Adresse(;EhelichesKind(;;@Pos(%i))))",""
```

Jede Zeile enthält Angaben über die Ebene der Information im hierarchischen Baum, ihre Beschriftung sowie die Beschreibung des entsprechenden Objekts im Atlas-Format. Mit Hilfe des Platzhalters *%i* kann ein Zähler gesetzt werden, der den jeweiligen Zweig so oft wiederholt, bis an der entsprechenden Verzweigung kein weiteres Objekt ermittelt werden kann.

In der Praxis soll das Programm schnell über den Sach- und Streitstand in einem Fall Auskunft geben. Im übrigen eignet sich auch dieses Programm gut zum Experimentieren mit Atlas. Es wurde ebenfalls in der 32-Bit Version von Visual–BASIC 4.0 entwickelt. Auch diese Programmquellen befinden sich auf der Diskette.

5. Fallskizze

Das Programm CASE.EXE enthält eine Anwendung mit ähnlicher Grundfunktionalität wie das beschriebene Programm *Fallskizze*. Es ermöglicht das Positionieren von neuen und bereits instantiierten Objekten auf dem Arbeitsfeld. Objekte können dabei unterschiedliche Typen besitzen, die durch unterschiedliche Symbole dargestellt werden. Die Anzahl der Symbole kann durch Zusatzmodule erweitert werden.

Weiterhin können Ereignisse als Objekte mit Datumswert auch in einen Kalender eingetragen werden. Zwischen mehreren Objekten können wie beschrieben Beziehungen hergestellt werden. Diese werden durch entsprechende Pfeile angezeigt.

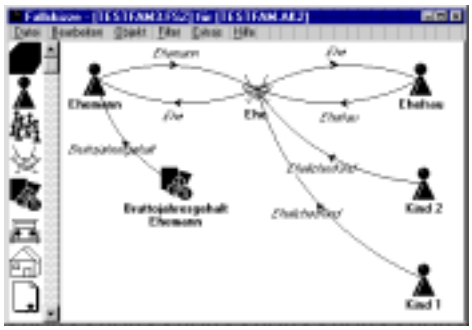


Abbildung 61: Das Programm *CASE*.
Die abgebildete Skizze beschränkt sich auf die Darstellung der Kinder als *eheliche* Kinder.

Nicht oder nicht vollständig implementiert sind einige Detailfunktionen So sind Stammdatenmasken für die unterschiedlichen Objekttypen zwar vorgesehen, aber in den entsprechenden Modulen nicht funktionsfähig. Assistenten sind nicht eingebaut und eine Gegenüberstellung von Fakten nach unterschiedlichen Behauptungen ist ebenfalls nicht möglich. Dennoch ist das Programm recht gut geeignet, individuelle Fälle zu gestalten oder Szenarien zu modifizieren. Zudem kann man sich die computergestützte Fallskizze als alternatives Eingabemedium recht gut vergegenwärtigen.

6. Objektübersicht

Das Programm *ATLEXPL.EXE* bietet eine moderne Ansicht auf die Objekte eines Falls. Es ist auf die Oberfläche von Windows 95 abgestimmt.

Abbildung 62: Das Programm *ATLEXPL*.
Dargestellt wird das Szenario *Ehe* in der Symbolansicht.

Objekte werden wie mit Hilfe einer unter Windows 95 standardisierten Anzeigetechnik wahlweise als frei plazierbare Symbole[416] oder als Elemente einer Liste dargestellt. Es stehen mehrere Sichten auf eine Akte zur Verfügung. Diese Sichten filtern die Objekte nach unterschiedlichen Kriterien und weisen ihnen entsprechende Symbole zu. Auf diese weise können Beteiligte verschiedener Szenarien wie *Ehe* oder *Prozeß* aber auch *alle Menschen*, *alle Sachen* etc. ausgefiltert und angezeigt

[416] Die Positionen der Objekte werden ungefragt in einer eigenen Datei abgelegt.

werden. Standardmäßig werden die Sichten *Alle Objekte* und *Ereignisse*[417] zur Verfügung gestellt. Alle Übrigen Sichten sind konfigurierbar.

Abbildung 63: Das Programm *ATLEXPL*.
Dargestellt wird das Standardszenario *Ereignisse* in einer sortierten Listenansicht.

Zu jedem Objekt können zudem *Eigenschaften* angezeigt und bearbeitet werden. Auch hier entspricht die Benutzung dem neuen Standard. Die Eigenschaften werden über ein *kontextsensitives Menü* aufgerufen und in einem Dialogfenster dargestellt. Die unterschiedlichen Registerkarten im Eigenschaftsfenster bestimmen sich nach den Grundeigenschaften des Objekts wie *Mensch*, *Sache*, etc. Lediglich die Karte *Allgemein* gehört zu jedem Objekt. Hier werden die Werte des Objektes und seine Kategorisierung angezeigt.

Abbildung 64: Das Eigenschaftsfenster von *ATLEXPL*.

Die einzelnen Sichten, die Symbole sowie die angezeigten Eigenschaften im Eigenschaftsfenster sind frei konfigurierbar. Die Konfiguration lehnt sich an diejenige des Eingabeassistenten[418] an. Für die Darstellung der Eigenschaftsfenster wird ein identischer Aufbau in der Konfigurationsdatei eingesetzt. Der folgende Auszug aus einem Skript zeigt unter anderem das Element *Mensch*, das die Ansicht aus Abbildung 62 und die Maske aus Abbildung 64 erzeugt:

```
[Ansichten]
1=Szenario Ehe
2=Alle Menschen
```

[417] Vgl. den Kalender in der Fallskizze (S. 193).

[418] S. 229

```
[Szenario Ehe]
Kriterium=@Beziehung(Ehe(;;))
1=Ehemann
2=Ehefrau
3=Ehekind
4=Ehe

[Ehemann]
Kriterium=@Beziehung(Ehemann($%O;))
Symbol=Ehemann
Typ=Mensch

[Mensch]
Kriterium=@Beziehung(Mensch($%O))
Symbol=Mensch
Typ=Mensch
Erklärung={\rtf1\ansi{\fs28\b Eigenschaften eines Menschen}\par An dieser Stelle werden die Stammdaten eines
Menschen angezeigt. Sie können die Daten auch ändern.}
1=Text\nein\Vorname\Vorname(;$%O)
2=Text\ja\Nachname\Nachname(;$%O)
3=Text\nein\Straße\Strasse(;Adresse(;$%O))
4=Text\nein\PLZ\PLZ(;Adresse(;$%O))
5=Text\nein\Ort\Ort(;Adresse(;$%O))
6=Text\nein\Geburtsdatum\Geburtsdatum(;$%O)
7=Text\nein\Bruttojahresgehalt\Bruttojahresgehalt(;$%O)
```

Der Abschnitt *Ansichten* bestimmt die Kategorisierung der Objekte. Die Priorität ergibt sich aus der Ordnungsummer. Die einzelnen Ansichten sind eigene Abschnitte des Skripts (im Beispiel *Szenario Ehe*), die auf Klassifikationen von Objekten (z.B. *Ehemann*) verweisen. Ansichten und Objektklassen besitzen ein *Kriterium*. Hiernach wird entschieden, ob ein entsprechendes Szenario verfügbar ist oder ein Objekt zu einer bestimmten Klasse gehört. Objektklassen werden mit Symbolen dargestellt. Erfüllt ein Objekt mehrere Kriterien, so entscheidet die Ordnungsziffer der Objektklasse in einer Ansicht, wie das Objekt dargestellt wird.

Objektklassen können auf übergeordnete Klassen verweisen. Einem Objekt der hierarchisch untergeordneten Klasse wird bei bedarf das Kriterium der Oberklasse vom Programm zugewiesen. Ein Objekt mit der Eigenschaft *Ehemann* wird im Beispiel also automatisch mit der Eigenschaft *Mensch* versehen, falls er sie noch nicht hat. Bei der Anzeige des Dialogfelds *Eigenschaften* werden für bestimmte Objektklassen Grundeigenschaften angezeigt. Hierzu werden die entsprechenden Registerkarten angelegt, sofern ein Objekt das Kriterium der Klasse erfüllt. In der ersten Registerkarte *Allgemein* kann der Benutzer zudem das Objekt weiteren Klassen zuordnen. Dem Objekt werden dann implizit vom Programm die entsprechenden Kriterien zugewiesen.

Das Programm ähnelt teilweise der Anwendung CASE. Der Stand der Realisierung ist jedoch unterschiedlich. Das Programm *ATLEXPL* besitzt keine Möglichkeit, Beziehungen graphisch durch Pfeile darzustellen. Hingegen sind Eigenschftsfenster besser realisiert. Die Anpassung des Programms erfolgt über Skripten und nicht wie bei CASE über DLLs. Das Programm dient weniger der Anlage neuer Fallkonstellationen als der Anzeige vorhandener Objekte. Mit der Entwicklung sollte vor allem die Flexibilität des Grundsystems unter Beweis gestellt werden. So ist es nicht nur unproblematisch, eine 32–Bit Anwendung mit Atlas zu kombinieren. Die Anwendung kann vielmehr sofort nahtlos in die objektorientierte Benutzungsphilosophie des neuen Betriebssystems integriert werden.

Das Programm ist unter den realisierten Anwendungen wohl am besten geeignet, neue Objekte in einem Fall anzulegen und ihnen Eigenschaften und Rollen in Szenarien zuzuweisen. Aufgrund der skriptgesteuerten Konfigurierbarkeit kann es leicht auf individuelle Aufgaben angepaßt werden.

B. Kritische Betrachtung und Ausblick

Abschließend wird die Frage behandelt, inwieweit ein Einsatz des beschriebenen Systems realistisch ist und welche weiteren Schritte noch zu gehen sind bzw. gegangen werden können.

1. Realisationsmöglichkeiten

Das vorgestellte Programm *Atlas 2* erhebt nicht den Anspruch auf echte Praxistauglichkeit. Es ist ein Prototyp, der die Verhaltensweise des beschriebenen Systems demonstrieren soll. Seine Grenzen liegen zum einen in der technisch unvollkommenen Realisierung und zum anderen in fehlenden sprachlichen Elementen der Schnittstelle. So arbeitet die eingesetzte Datenverwaltung lediglich im Arbeitsspeicher des Computers. Sie erfüllt damit weder die Anforderungen an die notwendige Kapazität aufgrund der zu erwartenden Datenmengen noch diejenigen an die notwendige Sicherheit. Beide Mängel sind jedoch mit einem begrenzten Aufwand behebbar. Eine Anbindung etwa an einen klassischen SQL–Datenserver würde das Programm sicherer machen, die Datenmenge quasi von jeder Begrenzung befreien und das System auch für den Einsatz in einer verteilten Umgebung vorbereiten (s. nächsten Abschnitt).

Das Verhalten des Systems erscheint noch nicht immer schlüssig und erwartungskonform. Probleme liegen bei der impliziten Instantiierung von Objekten. Diese kann nicht immer so vorgenommen werden, wie es für einen minimalen Aufwand auf der Seite des Anwendungsprogramms nötig scheint. Eine Beschreibung eines Faktums in einem Schritt, wie sie eigentlich angestrebt wurde, ist deshalb oftmals ausgeschlossen. Zudem kann die an sich wünschenswerte Versionsverwaltung aller an das System übermittelten Fakten in der Praxis eher befremdlich wirken. Probleme treten besonders dann auf, wenn der Benutzer Eingaben während eines Arbeitsganges korrigiert, das Anwendungsprogramm die vorher unrichtigen Daten jedoch bereits an das System übermittelt hat. Hier sind für die Praxis Möglichkeiten zu schaffen, alte Eingaben zu annullieren oder die additive Sammlung der Fakten wenigstens vorübergehend abzuschalten.

Ein weiterer kritischer Punkt für die Praxis mag die Wahl der technischen Schnittstelle, nämlich des DDE-Protokolls sein. Es erfüllt zwar alle theoretischen Anforderungen an die gesuchte Schnittstelle: So ist es einfach aufgebaut und eignet sich deshalb auch für eine Beschreibung unabhängig von der eigentlichen Softwareplattform. In der Praxis wird es von einer großen Basis von Anwendungen insbesondere auch von der wesentlichen Standardsoftware unterstützt. Für die Programmierung von Individualsoftware hingegen ist der Einsatz eines solchen Protokolls jedoch etwas umständlich. Insbesondere wirkt es störend, daß es zwingend eine laufende Serveranwendung voraussetzt. Für eine Realisierung im Praxisbetrieb scheint es deshalb vernünftiger, als unterste Ebene eine abstrakte Programmschnittstelle mittels einer Programmbibliothek, also eine API[419] zu realisieren. Individualprogramme können diese Schnittstelle für den Zugriff auf die gesuchten Fakten benutzen, als wären es Routinen des eigenen Programms. In den gezeigten Programmbeispielen wird ein entsprechendes Vorgehen bereits dadurch simuliert, daß die Aufrufe der DDE-Schnittstelle in Unterfunktionen isoliert werden. Die Einrichtung einer Programmierschnittstelle schließt die Möglichkeit nicht aus, eine Applikation zu entwickeln, die die Funktionalität der Programmschnittstelle auf das DDE-Protokoll überträgt. So können auch Anwendungen wie etwa WinWord auf die Daten zugrei-

[419] *Application Programmers Interface*.

fen, selbst wenn sie eine Programmschnittstelle nicht oder nicht im ausreichenden Maße unterstützen.

Diese Schwächen der Realisierung des Prototypen können letztlich jedoch vernachlässigt werden. Die Entwicklung eines serienreifen Produktes, das sich die hier gewonnenen Erkenntnisse zunutze macht, stellt kaum ein Problem dar, das einer praktischen Anwendung des Systems ernsthaft im Wege stehen könnte. Die vorgestellte Idee allerdings, beliebige Fakten unabhängig von starren Formaten klassischer Datenbanken zwischen Anwendungen auszutauschen, eröffnet in der Praxis sicherlich neue Perspektiven. Bereits die wenigen Beispielanwendungen lassen vermuten, daß die EDV-Unterstützung der Arbeit am juristischen Sachverhalt hierdurch eine neue Qualität gewinnen kann. Der Anspruch des Systems ist gleichwohl nicht so hoch, daß es aufgrund überhöhter Anforderungen sich selbst behindert. Zudem sind keine methodischen oder gar ethischen Bedenken gegen den Ansatz ersichtlich, die einen grundlegenden Zweifel am Sinn und Zweck des Systems aufkommen lassen.

Tatsächliches Realisierungsproblem kann die Notwendigkeit einer Normung der Relationen darstellen. Ohne eine solche Normung dieser Informationsträger ist ein entsprechender Datenaustausch nicht sinnvoll möglich. An dieser Stelle sollte man sich jedoch auf entsprechende Marktmechanismen verlassen. Sofern wirklich, wie hier vermutet wird, ein Bedarf an einer Integration mehrerer Arbeitsgänge und der damit verbundenen Software besteht, wird eine Einigung von den Teilnehmern an dem System herbeigeführt werden.

2. Austausch von Fakten in einem elektronischen Netz

Die vorliegende Konzeption beschränkte eine Integration juristischer Tätigkeit auf die Verwendung einmal ermittelter Fakten eines Falls in unterschiedlichen Anwendungen. Auf diese Weise wird nicht nur der Aufwand vermindert, dieselben Daten mehrfach manuell einzugeben, es wird zudem die Suche nach Informationen erleichtert. Letztlich werden vielerlei Fehler vermieden und die Konsistenz der Qualität verbessert.

a) Arbeiten im lokalen Netz einer Kanzlei

Die Ausführungen gingen dabei von der Arbeit an einem Einzelplatz aus. Die zugrundeliegende Client-Server–Architektur erlaubt jedoch von ihrer Natur her einen Einsatz in einer Arbeitsumgebung mit verteilten Aufgaben. In diesem Fall können möglicherweise Fakten von unterschiedlichen Anwendern gleichzeitig geändert werden. Das hat zur Folge, daß sich die Fakten eines Falls für einen Benutzer auch dann ändern können, wenn er selbst keine Änderungen vornimmt. Dies geschieht etwa dann, wenn im Sekretariat gerade die Stammdaten zu einem Fall erfaßt werden, während der Anwalt gleichzeitig den Fall in einer Fallskizze betrachtet.

Für den Einsatz in derartigen verteilten Umgebungen müssen Mechanismen hinzugefügt werden, die unterschiedliche Berechtigungen für den Zugriff der Benutzer auf die Daten ermöglichen. Es muß sichergestellt werden, daß nicht alle Arten von Fakten von jedem Benutzer eingesehen oder gar geändert werden können. Weiterhin müssen Mechanismen entwickelt werden, die ein gleichzeitiges Ändern derselben Fakten verhindern. Zuletzt ist eine Technik einzurichten, die alle beteiligten Anwendungen in einem Netzwerk über Änderungen an den Fakten informiert. Nur so kann gesichert werden, daß alle Anwendungen immer mit dem aktuellsten Bestand an Daten arbeiten.

b) Arbeiten im Verbund mit anderen Verfahrensbeteiligten

Beschränkt man die Integration auf die Fakten eines Falls, so ist es gerechtfertigt, daß den Fragen der Vernetzung von Computern keine überwiegende Bedeutung zugemessen wird. Die auftretenden Probleme lassen sich durch entsprechende Arbeitsabläufe und gängige Sicherungsmechanismen lösen. Etwas anderes kann sich dann ergeben, wenn die am juristischen Verfahren beteiligten Personen innerhalb eines Netzwerkverbundes ihre Informationen auf elektronischem Wege austauschen. Ziel eines solchen Austauschs kann es sein, daß alle Verfahrensbeteiligten auf einheitliche, objektiv verständliche Informationen zugreifen. Diese Informationen können zudem maschinell gelesen und verarbeitet werden. In stark formalisierten Verfahren wie dem Mahnverfahren wird ein entsprechender Effekt durch entsprechende - meist noch gedruckte - Formulare erreicht.

Für den Austausch solcher Fakten stellen die beschriebenen Ansätze eine Grundlage dar. Die technische Realisierung ist auf unterschiedlichen Wegen denkbar. Zum einen können derartige Informationen mittels konventioneller elektronischer Post ausgetauscht werden, zum anderen wäre die Einrichtung eines zentralen Faktenservers vorstellbar.

Die herkömmliche elektronische Post - sie gehört bisher weder zwischen mehreren Anwälten noch zwischen Anwalt und Gericht zum Alltag[420] - beschränkt sich zunächst auf den Austausch klassischer Schriftsätze über einen einfacheren und schnelleren Weg. Die Schriftsätze können als reiner Text oder bei moderneren Systemen als typographisch gestalteter Text übersendet werden. Die Verarbeitung der Information beim Empfänger geschieht vornehmlich durch Lesen, d.h. im Kopf des Empfängers. Es ist jedoch bereits heute ohne weiteres technisch möglich, komplexe Informationen so auszutauschen, daß sie automatisch und intellektuell ausgewertet werden können.

Beispielsweise kann der Absender Passagen in einzelnen Schreiben auszeichnen, so daß sie vom Empfänger automatisch interpretiert werden. Ein Standard im Bereich des Dokumentaustauschs ist SGML[421]. Dabei handelt es sich um eine Konvention, die eine Kennzeichnung von Textpassagen nach beliebigen logischen Kriterien erlaubt. Die Normierung erfolgt unabhängig von den verwendeten Hard- und Softwareplattformen. Deshalb können SGML-ausgezeichnete Schreiben auch problemlos über jedes Mailsystem verschickt werden. Die optional oder zwingend zu verwendenden logischen Elemente eines Textes werden je nach Textgattung und entsprechenden Anwenderbedürfnissen in einer sogenannten DTD[422] festgelegt. Bei einem Anwaltsschriftsatz könnten danach etwa die in der ZPO vorgeschriebenen Informationen als Pflichtinformationen bestimmt und entsprechend ausgezeichnet werden. Der Schriftsatz kann so automatisch einem Verfahren und dort der entsprechenden Partei zugeordnet werden.

Die in dieser Arbeit gewonnenen Erkenntnisse können ebenfalls bei der Gestaltung einer Normierung einfließen, sofern eine differenziertere Auszeichnung des Sachverhaltes gewünscht ist. Das folgende Beispiel zeigt eine Passage eines fiktiven SGML-ausgezeichneten Schriftsatzes. Sie ist als *Behauptung* gekennzeichnet. Zur

---

[420] Interesse an elektronischer Post wird regelmäßig bekundet (z.B. *Bauer*/*Lichtner*, S. 118 f. Durch die starke Expansion des Internet ist allerdings in Zukunft mit einer breiten Infrastruktur zu rechnen.

[421] *Standard Generalised Mark-up Language*.

[422] *Document Type Declaration*.

Präzisierung, was behauptet wird, sind die Attribute *Faktum* und *Wert* gesetzt. Als Faktum wird dabei eine Atlas-Faktenbeschreibung verwendet:

<BEHAUPTUNG Faktum="Einkommen(;Vater(;Kind()))" Wert="130000">Das Einkommen meines Mandanten liegt bei
<DM>130.000</DM>.<BEWEIS>Einkommensbestätigung des Arbeitgebers.</BEWEIS></BEHAUPTUNG>

Eine alternative Möglichkeit, automatisch zu verarbeitende Informationen mit elektronischer Post zu verschicken, ist die häufig praktizierte Versendung von binären Anlagen. Sie werden einem elektronischen Schreiben im Originalformat der Anwendung beigefügt, die die Daten erzeugt hat. Sie können beim Empfänger direkt von den zugehörigen Anwendungen weiterverarbeitet werden. Auf diese Weise könnten auch Daten im hier beschriebenen Format versendet und in die Systeme der anderen Verfahrensbeteiligten eingespielt werden. Atlas müßte um eine Funktion erweitert werden, die alle oder einen Teil der Fakten eines Falls zum Verschicken in eine Anlage verpackt. Beim Empfänger packt eine weitere Funktion die Anlage wieder aus und spielt die neuen Fakten in den Datenbestand des entsprechenden Falls ein.

Alternativ zum Austausch von Fakten per elektronischer Post ist eine Architektur, bei der die Beteiligten auf einem einheitlichen Atlas–Faktenserver aufsetzen. Ein solcher Server könnte etwa bei einem Gericht aufgestellt werden. Die Verfahrensbeteiligten greifen dann über eine Onlineverbindung auf die Daten zu. Prinzipiell entspricht das System dann einer lokalen Vernetzung mit dem einzigen Unterschied, daß Teile der Verbindung über öffentliche meist langsamere Leitungen erfolgten. Ein solches System unterläge jedoch von vornherein erheblichen Sicherheitsbedenken. Zum einen sind die öffentlichen Leitungen bis heute nicht über Zweifel an der Sicherheit vor Zugriffen anderer erhaben. Zum anderen bestehen bei einem direkten Zugriff aller Beteiligten auf dieselben Daten Bedenken gegenüber der Authentifizierung der Benutzer. Letztlich verlassen sich hier alle Anwender auf die Sicherheitssysteme des Betreibers des Servers, wohingegen beim Mailsystem jeder Anwender seine Daten mit der eigenen Sorgfalt wartet. Insbesondere die Angaben über die Informationsquelle und -zeit dürften beim Eingeben von Daten nicht ohne Sicherheitsmechanismen veränderbar sein.

Aber auch die einfachere Variante des Datenaustauschs über elektronische Post wird sich aufgrund praktischer Bedenken nicht allzu schnell durchsetzen: Zum einen bestehen auch hier Sicherheitsbedenken. Der Absender einer E-Mail muß beispielsweise mindestens mit der Sicherheit identifizierbar sein, die eine Unterschrift bietet[423]. Weiterhin müssen sich die Verantwortlichen auf einen Standard im Bereich der E-Mail zwischen den Justizorganen einigen.

Während jedoch Ressentiments, die auf technischen Unzulänglichkeiten beruhen, durch die fortschreitende Entwicklung auszuräumen sind, mögen psychologische Barrieren eine derartige Vernetzung langfristig gefährden. Sie treffen nämlich unmittelbar die Sinnfrage der Idee. Bedenklich ist genau gesagt, inwieweit die gegnerischen Parteien überhaupt ein Interesse daran haben, miteinander oder mit dem Gericht Fakten auf einem objektivierbaren Niveau auszutauschen. So mag die Kunst des Anwalts oft genug darin bestehen, einen Mangel an objektiven Fakten durch seine Eloquenz auszugleichen. Es mag aber auch sein, daß es gerade bei einer an

---

[423] *Bauer/Lichtner*, S. 118; umfassend *Kuhn*, **Rechtshandlungen mittels EDV und Telekommunikation**, München 1991. Womöglich bietet die Arbeit von *Britz* (a.a.O.) weitere Anhaltspunkte.

sich eindeutigen Sach- und Rechtslage auf eine besonders gute Argumentation ankommt. Liefert man nun die Fakten in einer technisch objektiven Form und koppelt sie so von der Darstellung und Argumentation im Schriftsatz ab, so birgt dies die Gefahr in sich, daß der objektiven Sachlage ein unverhältnismäßig hoher Stellenwert eingeräumt wird.

Zudem mag die Kommunikation mit automatisch zu verarbeitenden Fakten auch die oft gefürchtete Vision verstärken, Recht und Rechtsprechung könnten automatisierbar werden. Während die hier präsentierten Hilfsmittel noch zum Ziel hatten, dem entscheidenden Menschen die Entscheidungsfindung zu erleichtern, sind bereits seit vielen Jahren Forschungsansätze erkennbar, die die Entscheidung selbst jedenfalls teilweise dem Computer zu überlassen. Über Sinn und Unsinn solcher Tendenzen mag hier jedoch nicht weiter diskutiert werden. Entsprechende Stimmen scheinen auch eher verhaltener zu werden. Jedenfalls unterstützen eine Vernetzung der Verfahrensbeteiligten und ein Austausch der Fakten auf dem beschriebenen Niveau in erheblichem Maße die Möglichkeiten wenigstens teilweise automatisierter Verfahrensbearbeitung. Ein Computer mit juristischem Wissen könnte ohne menschliches Eingreifen Beweisbeschlüsse abfassen, Termine ansetzen und im äußersten Fall Entscheidungen fällen.

3. Regeln als Austauschobjekt

Ein letztes Thema wurde bislang bewußt stiefmütterlich behandelt, knüpft jedoch unmittelbar an das Gesagte an: der Austausch von Regeln zwischen Anwendungen und Anwendern. Das bisherige Szenario begrenzte sich darauf, die Fakten eines Falls zwischen mehreren Programmen auszutauschen. Die soeben entwickelte Vision zeigt, daß der Schritt zum Austausch der Fakten zwischen Benutzern, also zwischen den an einem Verfahren Beteiligten nicht weit ist. Wie aber ist es, wenn die Verarbeitungsregeln ebenfalls aufgrund einer definierten Beschreibung abrufbar werden. Die Frage lautet dann nicht, *gib mir das Nettogehalt von X*, sondern *gib mir eine Regel, die aus dem Bruttogehalt Y das Nettogehalt berechnet*.

Man stelle sich vor, ein Szenario aus Fakten wird in ein Netz (das kann das Spinnennetz der eigenen Festplatte genau so sein wie das internationale Internet) geschickt mit der Anforderung, eine oder mehrere Regel zu finden, die einen definierten Endzustand erzeugen: *Suche mir eine Regel, nach der ich mit einem Gehalt von 3000 nur 100 Steuern zahlen muß. Bruttogehalt*$(, \mathit{Anton}) = 3000 \Rightarrow$ *Nettogehalt*$(, \mathit{Anton}) = 2900)$. Die technische Form, solche Regeln letztlich zu übermitteln, ist kaum problematisch. Findet sich diese Regel in einer Gerichtsentscheidung, so mag der Austausch auf klassischem Wege durch Übermittlung des Textes geschehen. Handelt es sich um einen Algorithmus, so kann dieser entweder in einer im Netzwerk üblichen Programmiersprache[424] übermittelt werden oder der Einsatz erfolgt direkt an der gefundenen Stelle im Netz.

Wichtig ist der hierdurch in Reichweite gerückte Ansatz, Regeln durch Ausgangsszenarien und angestrebte Lösungen zu bescheiden. Dies ist nur in einer Sprache möglich, die eine objektive Beschreibung der Fakten ermöglicht. Eine Suche mit Hilfe einer konkreten Kollektion an Fakten in einer Menge von entsprechend beschriebenen Regeln ist sicherlich präziser als die klassische Suche etwa nach einer Norm oder einer Gerichtsentscheidung über Schlagworte oder die Suche nach jedem

[424] Ein genereller Bedarf, Algorithmen in Netzwerken auszutauschen und zu verbreiten, zeigt der überraschende Erfolg der Sprache *JAVA*. Sie ist als plattformunabhängige Programmiersprache speziell für das Internet gedacht und ermöglicht unter anderem die Publizierung von Algorithmen in Online-Diensten.

Wort im Text. Gewissermaßen als Pendant hierzu kann der herkömmliche Thesaurus durch eine Regel oder ein Netz von Regeln ersetzt werden. Hier wird ein Begriff nicht etwa nur in eine Begriffshierarchie eingeordnet, sondern es können komplexe Ausdrücke entwickelt werden, die die Vergleichbarkeit eines aktuellen Faktums mit dem Ausgangsfaktum einer Regel herstellen können.

Für Entscheidungen (Eine Entscheidung entwickelt meist de facto, oft auch de jure eine Bindungswirkung für die Zukunft und stellt mithin eine Regel dar.) etwa ergibt sich hier eine sehr einfache aber hoch präzise Möglichkeit der Archivierung. Verwendet man nämlich ein System wie das in dieser Arbeit entwickelte zur Erfassung des Sachverhaltes und zum Austausch der Fakten zwischen den einzelnen Vorgängen der juristischen Entscheidungsfindung, so hat man implizit die relevanten Suchkriterien vergeben. So ist das Ausgangsszenario mit allen entscheidungsnotwendigen Fakten erfaßt. Zudem werden durch die Subsumption auch die generellen Zwischenergebnisse sowie das Endergebnis ermittelt. Die Speicherung dieser Informationen und des Endergebnisses als Schlüssel für die Entscheidung selbst, bieten eine gute Grundlage, andere Sachverhalte mit dem Szenario abzugleichen und so relevante Entscheidungen mit hoher Präzision wiederzufinden.

Ähnlich lassen sich Regeln ablegen, die auf Algorithmen basieren. Hier werden das Ausgangsszenario und das Ziel bereits durch die definierten Zugriffe auf die Schnittstelle bestimmt. Lediglich im Bereich der Gesetzgebung gibt es kein Verfahren, das diese Form der Erschließung quasi en passant erledigt. Da das Gesetz jedoch das Rückgrat auch eines solchen Systems darstellt, wird es sinnvoll sein, die notwendigen Normungen intellektuell vorzunehmen. Dieser Aufwand unterscheidet sich allerdings nicht von demjenigen, der zur Normierung der Relationen selbst als Basis von Atlas aufgebracht werden muß.

Die Erschließung von Regeln über die Darstellung der Ausgangs- und Zielszenarien ist also eine konsequente Weiterentwicklung des beschriebenen Systems. Sie ermöglicht es dem Anwender, Regeln zu finden, die auf die Situation seines Falls zutreffen. Dabei hat er die Möglichkeit, nicht nur die Ausgangslage zur Grundlage der Suche zu machen, sondern er kann auch ein zu erreichendes Ziel vorgeben. Dies ermöglicht ein taktisches Vorgehen einer Partei. Wenn diese Informationen frei verfügbar sind, ist auch eine Vorhersage des Handelns des Gegners leicht.

Die globale Verfügbarkeit von Fakten in einer begrenzten Arbeitsumgebung eines Juristen ist also nur der erste Schritt auf dem Weg zu einem integrierten Arbeitsplatz. Weitere Schritte sind die Integration mehrerer Organe der Rechtspflege zur Erlangung einheitlicher Sachverhaltskenntnisse. In einem zusätzlichen Schritt werden Regelinformationen unterschiedlicher Art so abrufbar, daß die für einen Fall relevanten Lösungen mit Hilfe der vorhandenen Fakten ermittelt werden. Sofern die Regeln algorithmisiert sind, sind sie dann am Computer sofort einsetzbar.

Der Weg zu diesem Ziel ist wohl realistischer als manche der ganzheitlichen Ansätze der früheren Tage. Die technische Machbarkeit dürfte angesichts der steigenden Leistungsfähigkeit der Geräte und deren weltweiter Vernetzung in Zukunft auch nicht der begrenzende Faktor sein. Vor jeder Einführung des technisch Machbaren sollte jedoch die Frage geklärt werden, ob das Machbare auch das Wünschenswerte ist.

Teil V. Anhang

A. Inhalt und Installation der Diskettenbeilage

Die CD-ROM enthält das Atlas 2 Programmsystem sowie einige Anwendungsbeispiele. Zudem ist der Text dieser Arbeit in einer elektronischen Aufbereitung enthalten. Inhalt und Installation und Funktionsweise der CD-ROM–Beilage sind auf der CD-ROM selbst dokumentiert. Beachten Sie bitte folgendes:

1. In der Datei README.TXT befinden sich die neusten Hinweise über die CD-ROM.
2. Sie installieren das System Atlas 2 und alle weiteren Komponenten mit Hilfe des Programms SETUP.EXE. Die Installation einiger Programme ist optional.
3. Die Datei INHALT.HLP enthält Hinweise auf die Benutzung der Programme mit Verweisen auf die Hilfesysteme der einzelnen Programme.

B. Wahrheitswertetabelle der am häufigsten benötigten logischen Junktoren

		Konjunktion	Disjunktion	Kontravalenz	Exklusion	Implikation	Replikation	Äquivalenz	Negation
p	q	$p \wedge q$	$p \vee q$	$p \otimes q$	$p \times q$	$p \rightarrow q$	$p \leftarrow q$	$p \leftrightarrow q$	$\neg p$
w	w	w	w	f	f	w	w	w	f
w	f	f	w	w	w	f	w	f	f
f	w	f	w	w	w	w	f	f	w
f	f	f	f	f	w	w	w	w	w

C. Formular zur Aufnahme von Ehe und Familiensachen

D. Erklärung über die persönlichen und wirtschaftlichen Verhältnisse

E. Fragebogen zum Versorgungsausgleich

F. Source-Code des Atlas-Servers

Der folgende Abdruck des Source–Codes dient nur dem interessierten Leser. Es handelt sich dabei um den originalen Stand der aktuellen Version. Der Code ist nicht als Inhalt der Arbeit gedacht.

ATL2DKL.PAS

```
unit atl2dkl;  {Einheitlicher Deklarationsteil für Atlas2}

{Letzte Änderung: 20.06.93}

interface

uses WinTypes, ddeml, StrConv;

type
  tLine = array[0..255] of char;
  tSLine = array[0..127] of char;
  tFileName = array[0..255] of char;
  tPath = array[0..127] of char;

{ *** WinIni Informations *** }

{Zur flexibleren Systemverwaltung sind einige Konstanten manipulierbar}

const
  AtlIniFile = 'ATLAS2.INI'#0;
  AtlIniSection = 'General';          {Section für generelle Informationen über AtlasSV2}
  AtlIniConvTopic = 'ConvTopic'#0;    {Dateiname des Altlas2-Servers}
  AtlIniBaseFileName = 'BaseFile'#0;  {Dateiname der Repräsentationsbasis}
  AtlIniServer = 'Server'#0;          {Name des Atlas Servers}

{Deklaration für Typen der Wissensbasis}

const

{LRelFlags 32 Bit}
  tty_text      = $00000001;
  tty_integer   = $00000002;
  tty_real      = $00000003;
  tty_date      =  $00000004;
  tty_xbool     = $00000005;

  tty_UpperCase  = $00010000; {mit tty_text}
  tty_lowercase =  $00020000; {mit tty_text}
  tty_currency  = $00010000; {mit tty_integer oder tty_real}
  tty_percent    = $00020000; {mit tty_integer oder tty_real}
  tty_fixed      =  $00100000; {mit tty_integer oder tty_real}
                     {oberstes Byte bezeichnet Anzahl der Kommastellen}
  tty_longdate  = $00010000; {mit tty_date}
  tty_shortdate = $00020000; {mit tty_date}

  ptty_text: pchar    = 'Text'#0;
  ptty_integer: pchar  = 'Ganzzahl'#0;
  ptty_real: pchar    = 'Realzahl'#0;
  ptty_date: pchar    =  'Datum'#0;
  ptty_xbool: pchar    = 'Existenz (j/n/v)'#0;

{English Version
  ptty_text: pchar    = 'Text'#0;
  ptty_integer: pchar  = 'Integer'#0;
  ptty_real: pchar    = 'Real'#0;
  ptty_date: pchar    =  'Date'#0;
  ptty_xbool: pchar    = 'Existance (t/f/u)'#0;}

{const EmptyTrm: tRelation}

type
  pRelation = ^tRelation;
  tRelation = record
    SzRelBez: tLine;
    LRelFlags: LongInt;
    HRelParam,           {Attribute (Rollen)}
    HRelAddOn: tHandle;    {Filterspez Infos}
  end;

const
  leereRelation: tRelation = (
```

```
    SzRelBez: #0;
    LRelFlags: tty_xbool;
    HRelParam: 0;
    HRelAddOn: 0);

type
  pXRelation = ^tXRelation;
  tXRelation = record
    RelXRelation: tRelation;
    HXBeziehung: tHandle; {Handle eines Wertebaums}
  end;

type
  pAddOn = ^tAddOn;
  tAddOn = record
    WAddOnID,             {ID des jew Filters muß in INI-Datei angegeben werden}
    WAddOnParam: word;    {16-Bit Parameter dient dem Filter als Info}
    LAddOnFlags,          {Identifizierung des Filtertyps}
    LAddOnParam: LongInt; {32-Bit Parameter dient dem Filter als Info}
  end;

type
  pRelParam = ^tRelParam;
  tRelParam = LongInt;

{Deklaration für fallbezogene Typen}

const
  Obj_Dummy = $00000000;

const
  Obj_Virtual = $0002;
  Obj_Real    = $0001;

type
  {Beschreibung einer Entität}
  pObjekt = ^tObjekt;
  tObjekt = record
    SzObjBez: tLine;        {Eindeutiges allgemeingültiges Lable}
    HObjWert: tHandle;      {Handle eines Speicherblocks mit tObjektWert-Liste}
    LObjFlags: LongInt;     {Obj_xxx Konstanten}
    WObjDdeFlags: Word;   {**unbenutzt** DdeConversations immer an eine Ety gekoppelt}
  end;

const
  ObjLeer: tObjekt = (
    SzObjBez: '';
    HObjWert: 0;
    LObjFlags: Obj_real;
    WObjDdeFlags: 0);

const
  sa_dummy    = $00000001; {Verwendet nur Dummy}
  sa_InclDmy  = $00000002; {Verwendet immer auch! Dummy}
  sa_exact    = $00000004; {Verwendet nur Angaben}

const
  IA_Dummy      = $00000000;
  IA_OverWrite  = $00000001;

type
  {Liste von Faktoren, die für jedes Datum (Wert und Relation) gültig sind}
  pEingFlt = ^tEingFlt;
  tEingFlt = record
    ObjDatAnfang,
    ObjDatEnde,           {Gültigkeitszeitraum, IDs von Entitäten}
    LEingabeDat,          {Eingabezeitpunkt als Datumswert}
    ObjQuelle,            {Quelle des Wertes, ID eines Beteiligten}
    LEFltFlags: LongInt;  {IA_xxx Konstanten}
    HFltAddOn: tHandle;   {Hd für Filterspezifische Informationen}
  end;

  pAusgFlt = ^tAusgFlt;
  tAusgFlt = record
    LGeltDatAnfang,
    LGeltDatEnde,         {Gültigkeitszeitraum in Datumswerten!!}
    LEingabeDatAnfang,
    LEingabeDatEnde,      {Eingabezeitraum in Datumswerten!!}
    ObjEingabeQuelle,     {Quelle des Wertes, ID eines Beteiligten}
```

```
    LAusgFltFlags: LongInt;
  end;

  pPosition = ^tPosition;
  tPosition = record
    PosIndex,            {Position in der Sortierreihenfolge}
    PosSort: LongInt;    {Ind eines zweistelligen Terms}
  end;

cons
  InpFltDmy: tEingFlt = (
    ObjDatAnfang: Obj_dummy;
    ObjDatEnde: Obj_dummy;
    LEingabeDat: date_dummy;
    ObjQuelle: Obj_dummy;
    LEFltFlags: sa_InclDmy;
    HFltAddOn: 0);

  DummySrcAttr: tAusgFlt = (
    LGeltDatAnfang: date_dummy;
    LGeltDatEnde: date_dummy;
    LEingabeDatAnfang: date_dummy;
    LEingabeDatEnde: date_dummy;
    ObjEingabeQuelle: Obj_dummy;
    LAusgFltFlags: sa_InclDmy);

{   DummyAttach: tAttach = (
    AttProof: Obj_dummy;
    AttNote: 0);}

  DummyPosition: tPosition = (
    PosIndex: 0;
    PosSort: 0);

const
  wt_exist = 1;
  wt_number = 2;
  wt_string = 4;
  wt_date = 10;
  wt_id = 0;

type
  {Einzelne Angabe zum Wert einer Entität}
  pObjektWert = ^tObjektWert;
  tObjektWert = record
    FltObjWert: tEingFlt;    {Standardattribute}
    LWertFlags: LongInt;     {derzeit nicht belegt}
    case WWertTyp: word of   {Typ des Wertes}
      wt_exist:    (XbWertExist: xBool); {xbool_xxx - Konstanten}
      wt_number:   (DbWertZahl: real);
      wt_date:     (LWertDatum: LongInt);
      wt_string:   (SzWertZeichen: tLine);
  end;

const max_attrib = 64;

type
  {Connex von Entitäten mit einer Haupt-Entität}
  pBeziehung = ^tBeziehung;
  tBeziehung = record
    FltBezWert: tEingFlt;    {Strandardattribute}
    XbBezWert: xBool;        {xbool_xxx-Konstanten}
    LBezFlags: LongInt;      {Nicht Belegt}
    LBezHauptObj: LongInt;   {ID der Haupt-Entität}
    LbfBezObj: array[1..max_attrib] of LongInt;
    {maximal 64 weitere Entitäten, abhängig vom zugrundeliegenden Terminus}
  end;
```

function GetIDate: LongInt;

implementation

uses WinDOS;

function GetIDate: LongInt;

```
var
  y, m, d, dw: word;
  ai: LongInt;

begin
  GetDate(y, m, d, dw);
  DateToNum(ai, y, m, d);
  GetIDate:=ai;
end;
end.
```

AtlasSrv2.PAS

```
program AtlasDDE;

{Version 2

Letzte Änderung: 09.08.93}

{$M 16384, 8192}  {Rekursion benötigt Platz im Stack}

{AtlasSv2.res}

{$R atlassv2.res}

uses
  Strings, WinTypes, WinProcs, WinDOS, WObjects,  {Windows Standard}
  Win31, DDEML, CommDlg,                          {Windows 3.1}
  StrConv, atl2base, atl2dkl, atl2syx, atl2fnc;
                                    {Private Units}

const
  noDebug=true;

const
  ProgName: pChar = 'ATLAS-2 Server';
  ProgDate: pChar = '04.04.96';
  ProgVer: pChar = '2.02.03'; {ab V2.01 arbeitet Atlas mit atlasdb2.dll}
  AtlIniSection = 'Atlas'#0;

const
  CrLf = #13#10;
  idInst: LongInt = 0;
  aConvHd: hConv = 0;
  ConvCount:  word = 0;

{  TheFileName: tPath = #0;}
  FileConvTopic: tLine = #0;
  ProgService: pChar = 'atlassv2';
  TheClassName: pChar = 'ATLASSV2'#0;
  TheRepClassName: pChar = 'ATLASR2'#0;
  DebugFileName = 'debug.txt';

  fltAktEingFlt: tEingFlt = (
    ObjDatAnfang: Obj_dummy;
    ObjDatEnde: Obj_dummy;
    LEingabeDat: date_dummy;
    ObjQuelle: Obj_dummy;
    LEFltFlags: sa_InclDmy;
    HFltAddOn: 0);

  CurSrcAttr: tAusgFlt = (
    LGeltDatAnfang: date_dummy;
    LGeltDatEnde: date_dummy;
    LEingabeDatAnfang: date_dummy;
    LEingabeDatEnde: date_dummy;
    ObjEingabeQuelle: Obj_dummy;
    LAusgFltFlags: sa_InclDmy);

const

{Main-Window-Controls}
  id_TrmSt      = 101;
  id_McrSt      = 102;
  id_EtySt      = 109;
  id_ConSt      = 110;
  id_ValSt      = 111;
```

```
  id_MonitorLb  = 104;
  id_StatusSt    = 106;
  id_TrmTypeCb  = 113;
  cm_New        = 1101;
  cm_Open       = 1102;
  cm_Save       = 1103;
  cm_SaveAs     = 1104;
  cm_SaveBase    = 101;
  cm_Quit       = 109;
  cm_report     = 901;
  cm_update     = 902;
  cm_Info       = 909;
  id_New        = 1001;
  id_Open       = 1002;
  id_Save       = 1003;
  id_Report     = 1004;

const
  CurFileName: tLine = ''#0;
  uFindReplaceMsg: word = 0;

var
  aFindReplace: tFindReplace;
  FindWhatLine: tLine;
  hBkBrush : hbrush;

type
  TApp = object(TApplication)
    constructor Init(AName: PChar);
    destructor Done; virtual;
    procedure InitMainWindow; virtual;
    procedure InitInstance; virtual;
  end;

var
  App: TApp;

{Hauptfenster - Deklaration}

type
  PMainWnd = ^TMainWnd;
  TMainWnd = object(TDlgWindow)
    constructor Init(AParent: PWindowsObject; ATitle: PChar);
    destructor Done; virtual;
    function getClassName: pchar;  virtual;
    procedure SetupWindow; virtual;
    procedure getWindowClass(Var AWndClass: TWndClass); virtual;
    function CanClose: Boolean; virtual;
    procedure SetTitle(aPChar: pchar);
    procedure CmNew(var Msg: tMessage); virtual cm_first + cm_New;
    procedure CmOpen(var Msg: tMessage); virtual cm_first + cm_Open;
    procedure CmSave(var Msg: tMessage); virtual cm_first + cm_Save;
    procedure CmSaveAs(var Msg: tMessage); virtual cm_first + cm_SaveAs;
    procedure CmSaveBase(var Msg: tMessage); virtual cm_first + cm_SaveBase;
    procedure CmInfo(var Msg: tMessage); virtual cm_first + cm_Info;
    procedure CmReport(var Msg: tMessage); virtual cm_first + cm_Report;

    procedure IdNew(var Msg: tMessage); virtual id_first + id_New;
    procedure IdOpen(var Msg: tMessage); virtual id_first + id_Open;
    procedure IdSave(var Msg: tMessage); virtual id_first + id_Save;
    procedure IdReport(var Msg: tMessage); virtual id_first + id_Report;

    procedure CmUpdate(var Msg: tMessage); virtual cm_first + cm_Update;
    procedure WmParentNotify(var Msg: tMessage); virtual wm_first + wm_ParentNotify;
    procedure DefWndProc(var Msg: tMessage); virtual;
    procedure WmSysCommand(var Msg: tMessage); virtual wm_first + wm_SysCommand;
    procedure AddConvToLst(Conv: hConv);
    procedure DeleteConvFromLst(Conv: hConv);
  end;

const
  TheMainWnd: pMainWnd=nil;

{Info-Dialog zeigt die Information über das Programm}

type
  pInfoWnd = ^tInfoWnd;
  tInfoWnd = Object(tDialog)
    TheName, TheDate, TheVersion: pChar;
```

```
    constructor init(AParent: PWindowsObject; ATitle, aName, aVersion, aDate: PChar);
    procedure SetupWindow; virtual;
  end;


constructor tInfoWnd.init(AParent: PWindowsObject; ATitle, aName, aVersion, aDate: PChar);

begin
  tDialog.init(aParent, 'INFODLG');
  TheName:=aName;TheVersion:=aVersion;TheDate:=aDate;
end;


procedure tInfoWnd.SetupWindow;

var
  aLine: tLine;

begin
  tDialog.SetupWindow;
  SendDlgItemMessage(hWindow, 102, wm_settext, 0, LongInt(TheName));
  SendDlgItemMessage(hWindow, 104, wm_settext, 0, LongInt(TheVersion));
  SendDlgItemMessage(hWindow, 106, wm_settext, 0, LongInt(TheDate));
  DatenbankInfozeile(aLine, SizeOf(tLine));
  SendDlgItemMessage(hWindow, 109, wm_settext, 0, LongInt(@aLine));
end;

{----------------------------CallBack für Filter----------------------------}

{$I atl2call.pas}[425]

{-----------------------Standard- und Debug-Funktionen-----------------------}

var
  DebugFile: text;


function XMessageBox(hWnd: tHandle; StrId, mbTyp: word): integer;

  {Zeigt eine Messagebox mit einem String aus der Stringliste.
  Zusätzlich wird der dem Icon entsprechende Systemklang gespielt}

var
  MsgLine: array[0..1023] of char;
  STyp: word;

begin
  if LoadString(hInstance, StrID, MsgLine, 1023) = 0 then
  begin
    wvsprintf(MsgLine, 'Unbekannte Meldung oder interne Fehlermeldung: CODE %i', StrID);
    {DEBUG-Meldungen immer mit Sound und Icon "STOP"}
    If (MbTyp=mb_ok) then MbTyp:=mb_ok or mb_iconstop or mb_systemmodal;
  end;
  STyp := MbTyp and {Ausfiltern der möglichen Systemklänge}
    (mb_iconasterisk or mb_iconexclamation or mb_iconhand
    or mb_iconquestion or mb_ok or mb_iconinformation);
  MessageBeep(STyp);
  XMessageBox:=MessageBox(hWnd, MsgLine, ProgName, MbTyp);
end;


procedure ShowStatus(str1, str2: pchar; dbg: boolean);

const
  CrLf: pChar = #13#10;
```

[425] S. 293 ff

```
var
  rLine, aLine: tLine;
  aCount: LongInt;

begin
  if (noDebug and dbg) or (theMainWnd=nil) then exit;
  if (Str1=nil) or (StrLen(Str1)=0) then
    StrLCopy(aLine, 'Leerlauf', SizeOf(tLine))
  else
    StrLCopy(aLine, Str1, SizeOf(tLine));
  StrLCat(aLine, ' ', SizeOf(tLine));
  if (Str2=nil) or (StrLen(Str2)=0) then
    StrLCat(aLine, '', SizeOf(tLine))
  else
    StrLCat(aLine, Str2, SizeOf(tLine));
  if not noDebug then
  begin
    write(debugfile, str1);write(debugFile, ':'#09#00);
    write(debugfile, str2);write(debugFile, CrLf);
  end;
  SendDlgItemMessage(TheMainWnd^.hWindow, id_StatusSt, wm_settext, 0, LongInt(@aLine));
  aCount:=RelationenAnzahl;
  StrCopy(rLine, '%li ');LoadString(hInstance, 4, @rLine[4], SizeOf(tLine)-5);
  wvsprintf(aLine, rLine, aCount);
  SendDlgItemMessage(TheMainWnd^.hWindow, id_TrmSt, wm_settext, 0, LongInt(@aLine));

  aCount:=ObjekteAnzahl;
  StrCopy(rLine, '%li ');LoadString(hInstance, 6, @rLine[4], SizeOf(tLine)-5);
  wvsprintf(aLine, rLine, aCount);
  SendDlgItemMessage(TheMainWnd^.hWindow, id_EtySt, wm_settext, 0, LongInt(@aLine));

  aCount:=AlleBeziehungenAnzahl;
  StrCopy(rLine, '%li ');LoadString(hInstance, 7, @rLine[4], SizeOf(tLine)-5);
  wvsprintf(aLine, rLine, aCount);
  SendDlgItemMessage(TheMainWnd^.hWindow, id_ConSt, wm_settext, 0, LongInt(@aLine));

  aCount:=AlleWerteAnzahl;
  StrCopy(rLine, '%li ');LoadString(hInstance, 8, @rLine[4], SizeOf(tLine)-5);
  wvsprintf(aLine, rLine, aCount);
  SendDlgItemMessage(TheMainWnd^.hWindow, id_ValSt, wm_settext, 0, LongInt(@aLine));

  aCount:=GetFreeSpace(gmem_not_banked);
  StrCopy(rLine, '%li ');LoadString(hInstance, 5, @rLine[4], SizeOf(tLine)-5);
  wvsprintf(aLine, rLine, aCount);
  SendDlgItemMessage(TheMainWnd^.hWindow, 113, wm_settext, 0, LongInt(@aLine));
end;
```

function IsFlag(aBitmap, aFlag: LongInt): boolean;

```
{Funktion überprüft, ob ein Bit in einem LongInteger gesetzt ist}

begin
  IsFlag:=(aBitmap and aFlag) = aFlag;
end;

const
  cat_null     = $00000000;
  cat_begin    = $00000001;
  cat_end      = $00000100;
  cat_source   = $00010000;
  cat_entry    = $01000000;
  cat_equal    = $01010101;
```

function VergleicheMitAusgabeFlt(cAttr: tEingFlt): longint;

```
{TESTVERSION
Funktion vergleicht die übergebene Inputliste mit der aktuellen Suchmaske CurSrcAttr.
8 Bits pro Attribut zur Darstellung des Vergleichsergebnisses.
cAttr: Ergebnisprofil aus der Datenbank
An dieser Stelle muß noch mal gründlich nachgedacht werden!!!}

var
  aLI: LongInt;
  BDate, EDate : LongInt;
```

function GetDate(anEtyId: LongInt): LongInt;

```
var
  Sz, x: word;
  einObjektWert: tObjektWert;
  einObjekt: tObjekt;
  aDate: LongInt;

begin
  aDate:=0; GetDate:=0;
  if not ObjektLesen(anEtyId, einObjekt) then exit;
  Sz:=ObjektWerteAnzahl(einObjekt); if Sz<1 then exit;
  for x:=1 to Sz do
  begin
    ObjektWertLesen(x, einObjektWert, einObjekt);
    with einObjektWert do
      if WWertTyp=wt_date then aDate:=LWertDatum;
  end;
  GetDate:=aDate;
end;

begin
  aLI:=cat_null;
  with cAttr do
  begin
    if ObjDatAnfang<>Obj_dummy then BDate:=GetDate(ObjDatAnfang) else BDate:=0;
    if ObjDatEnde<>Obj_dummy then EDate:=GetDate(ObjDatEnde) else EDate:=0;
  end;
  With CurSrcAttr do
  begin
    if (BDate<=LGeltDatAnfang) then aLI:=aLI or cat_begin;
    if (EDate>=LGeltDatEnde) then aLI:=aLI or cat_end;
    if (cAttr.ObjQuelle=CurSrcAttr.ObjEingabeQuelle) then aLI:=aLI or cat_source;
    if (cAttr.LEingabeDat>=CurSrcAttr.LEingabeDatAnfang)
      and (cAttr.LEingabeDat<=CurSrcAttr.LEingabeDatEnde)
      then aLI:=aLI or cat_entry;
    if isFlag(CurSrcAttr.LAusgFltFlags, sa_InclDmy) or (CurSrcAttr.LAusgFltFlags=sa_dummy) then
    begin
      if (cAttr.ObjDatAnfang = Obj_dummy)
        or (CurSrcAttr.LGeltDatAnfang = date_dummy) then aLI:=aLI or cat_begin;
      if (cAttr.ObjDatEnde = Obj_dummy)
        or (CurSrcAttr.LGeltDatEnde = date_dummy) then aLI:=aLI or cat_end;
      if (cAttr.ObjQuelle = Obj_dummy)
        or (CurSrcAttr.ObjEingabeQuelle = Obj_dummy) then aLI:=aLI or cat_source;
      if ((CurSrcAttr.LEingabeDatAnfang = 0) and (cAttr.LEingabeDat<=CurSrcAttr.LEingabeDatEnde))
        or ((CurSrcAttr.LEingabeDatEnde = 0) and (cAttr.LEingabeDat>=CurSrcAttr.LEingabeDatAnfang))
        or ((CurSrcAttr.LEingabeDatAnfang = 0) and (CurSrcAttr.LEingabeDatEnde = 0))
        then aLI:=aLI or cat_entry;
    end;
  end;
  VergleicheMitAusgabeFlt:=aLI;
end;
```

function VergleicheMitEingabeFlt(cAttr: tEingFlt): bool;

{TESTVERSION

Funktion Vergleicht die übergebene Inputliste mit den aktuellen fltAktEingFlt.}

```
begin
  with cAttr do
  begin
    VergleicheMitEingabeFlt :=
      (ObjDatAnfang = fltAktEingFlt.ObjDatAnfang) and
      (ObjDatEnde = fltAktEingFlt.ObjDatEnde) and
      (LEingabeDat = fltAktEingFlt.LEingabeDat) and
      (ObjQuelle = fltAktEingFlt.ObjQuelle);
  end;
end;
```

function VergleicheBeziehungen(bezZiel, bezRef: tBeziehung; AtC: word): boolean;

{bezRef (aus der Abfrage) darf auch Dummys enthalten}

```
var
  xWord: word;

begin
  {Standardwert 40}

  VergleicheBeziehungen:=false;

  {Identität des Hauptobjekts}
  if (bezRef.LBezHauptObj <> bezZiel.LBezHauptObj)
    and (bezZiel.LBezHauptObj <> Obj_dummy)
    and (bezRef.LBezHauptObj <> Obj_dummy)
    then exit;

  {Identität der weiteren Objekte}
  if AtC>0 then for xWord:=1 to AtC do
  begin
    if (bezRef.LbfBezObj[xWord] <> bezZiel.LbfBezObj[xWord])
      and (bezZiel.LbfBezObj[xWord] <> Obj_dummy)
      and (bezRef.LbfBezObj[xWord] <> Obj_dummy)
      then exit;
  end;

  {Identität der Beziehungswerte}
  if (bezRef.XbBezWert <> bezZiel.XbBezWert)
    and (bezZiel.XbBezWert <> xBoDmy)
    and (bezRef.XbBezWert <> xBoDmy)
    then exit;

  VergleicheBeziehungen:=VergleicheMitAusgabeFlt(bezZiel.FltBezWert)=cat_equal;
end;
```

procedure StrCopyTopic(aTopic, aFile: pchar);

```
var
  aDir, anExt, aName: tPath;

begin
  FileSplit(aFile, aDir, aName, anExt);
  StrCopy(aTopic, aName); StrCat(aTopic, anExt);
end;
```

function ValidateString(**var** aLine: tLine; flags: LongInt): boolean;

```
var
  aReal: real;
  year, month, day: word;
  aXBool: Xbool;

begin
  ValidateString:=false;
  aLine[SizeOf(tLine)-1]:=#0;
  while
    (aLine[StrLen(aLine)-1] in [#10, #13, #9]) and
    (StrLen(aLine)>1) do
    aLine[StrLen(aLine)-1] :=#0;
  case LoWord(Flags) of
    tty_integer, tty_real:
      begin
        if not StrToNum(aLine, aReal) then  exit;
        if  LoWord(Flags)=tty_integer then
          NumToStr(aLine, aReal, 0, SizeOf(tLine))
        else
          NumToStr(aLine, aReal, 3, SizeOf(tLine));
      end;
    tty_date:
      begin
        if not StrToDate(aLine, year, month, day) then exit;
        DateToStr(aLine, year, month, day, SizeOf(tLine));
      end;
    tty_xbool:
      begin
        if not StrToXBool(aLine, aXBool) then exit;
        XBoolToStr(aLine, aXBool, SizeOf(tLine));
```

```
    end;
 end;
 ValidateString:=true;
end;

{Allgemeine Funktionen}
```

function MatchTopic(aPchar: pChar): boolean;

```
{Funktion überprüft, ob ein String als Topic für eine DDE-Verbindung

akzeptiert wird}

begin
 if (lStrCmpI(aPChar, DefConvTopic)=0) {Standardtopic ist "System"}
   or (lStrCmpI(aPChar, SystemTopic)=0) then
   MatchTopic:=true
 else
   MatchTopic:=false;
end;

{Allgemeine Befehlsbearbeitungsfunktionen}
```

function ObjektIndexLesen(aPChar: pChar): LongInt;

```
{Diese Funktion prüft, ob pChar eine Entität beschreibt.

Ist dies der Fall, so wird der Index der Entität zurückgegeben, }

var
 aLongInt: LongInt;
 aReal: real;

begin
 ObjektIndexLesen:=0;
 if aPChar[0]='#' then
 begin {Die Entität wird durch den Index beschrieben.}
   {Der String nach "#" muß eine Zahl sein}
   if not StrToNum(@aPChar[1], aReal) then exit;

   {Die Zahl darf nicht größer als die Anzahl der Entitäten sein}
   if aReal<=ObjekteAnzahl then ObjektIndexLesen:=trunc(aReal);
 end else

 begin {Die Entität ist durch den Bezeichner beschrieben}
   {Gibt es keine Entitäten, ist die Suche obsolet}
   if ObjekteAnzahl<1 then exit;

   {ObjektSuchen ist eine Funktion des DB-Moduls}
   ObjektIndexLesen:=ObjektSuchen(aPChar, 1);
 end;
end;
```

function RelationIndexLesen(aPChar: pChar): LongInt;

```
{Diese Funktion prüft, ob pChar einen Term beschreibt.

Ist dies der Fall, so wird der Index des Terms zurückgegeben, }

var
 aLongInt: LongInt;
 aReal: real;

begin
 RelationIndexLesen:=0;
 if aPChar[0]='#' then
 begin {Der Term wird durch seinen Index beschrieben}
   {Der String nach "#" muß eine Zahl sein}
   if not StrToNum(@aPChar[1], aReal) then exit;
   {Die Zahl darf nicht größer als die Anzahl der Terme sein}
   if aReal<=RelationenAnzahl then RelationIndexLesen:=trunc(aReal);
 end else
```

```
  begin
    {Gibt es keine Terme, ist die Suche obsolet}
    if RelationenAnzahl<1 then exit;
    {RelationIndexLesen ist eine Funktion des DB-Moduls}
    RelationIndexLesen:=RelationSuchen(aPChar, 1);
  end;
end;

type
  ptLine = ^tLine;
  ptWord = word;

{$I ATL2FPE.PAS}[426]

{Der Funktionsblock FPE dient der rekursiven Auswertung der Ausdrücke.

Er sucht nach einer Entität in einer Parameterleiste. Diese Entität

kann durch eine Beziehung oder einen Bezeichner beschrieben sein.}


function RelationAusParameterLesen(szParameter: pChar; aPos: word): LongInt;

{Funktion überprüft, ob an der Position "aPos" der

Parameterleiste "szParameter" ein Term steht.

Sie gibt den Index des Terms zurück}

var
  aLine: tLine;
  aPChar: pChar;

begin
  RelationAusParameterLesen:=0;
  aPChar:=aLine;   {Puffer aufbauen}
  if not GetParam(szParameter, apChar, aPos, SizeOf(tLine)) then exit;  {Einlesen des Parameters}
  clipQuotes(aPChar); {Entfernen möglicher Anführungszeichen}
  RelationAusParameterLesen:=RelationIndexLesen(aPChar); {Suchen des Terms}
end;

{---------------------------------DDE-Mcr-Funktionen---------------------------------}


function McrObjektAnfuegen(ParamLine: pChar): boolean;

{Funktion fügt eine Entität an die Entitätenliste an.

Der Rückgabewert bezeichnet den Erfolg}

var
  aLine: tLine;
  aPChar: pChar;
  einObjekt: tObjekt;

begin
  McrObjektAnfuegen:=false; {Default ist pessimistisch}
  aPChar:=aLine; {Freischlagen des Puffers}

  {Auslesen des ersten Parameters. Er enthält den Term}
  if not GetParam(ParamLine, aPChar, 1, SizeOf(tLine)) then exit;

  {Der erste Buchstabe muß "$" sein}
  if aPChar[0]<>prefix_Objekt then exit; aPChar:=aPChar+1;

  {Entfernen zulässiger Anführungszeichen}
  ClipQuotes(aPChar);

  {Defaulteinstellungen der Entität}
  einObjekt:=ObjLeer;
  with einObjekt do StrCopy(SzObjBez, apChar);
```

[426] S. 283 ff

```
  {Anfügen mittels DB-Funktion ObjektZufuegen}
  if not ObjektZufuegen(einObjekt)=0 then exit;
  McrObjektAnfuegen:=true;
  ShowStatus(nil, nil, false);
end;
```

function McrAkteSpeichern(ParamLine: pChar): boolean;

```
{Funktion speichert die fallbezogenen Daten unter einem gegebenen Namen.

Der Rückgabewert bezeichnet den Erfolg}

var
  aLine: tLine;
  aPChar: pChar;

begin
  McrAkteSpeichern:=false;
  aPChar:=aLine;

  {Auslesen des Dateinamens nach aPChar}
  if not GetParam(ParamLine, aPChar, 1, SizeOf(tLine)) then exit;
  ClipQuotes(aPChar);
  ShowStatus('Speichern...', apChar, false);
  if AkteSichern(aPChar) then {speichern}
  begin
    FileExpand(CurFileName, aPchar); {Pfad für Anzeige vervollständigen}
    McrAkteSpeichern:=true;
    TheMainWnd^.SetTitle(aPChar);
  end;
  ShowStatus(nil, nil, false);
end;
```

function McrAkteOeffnen(ParamLine: pChar): boolean;

```
{Funktion öffnet eine Falldatendatei}

var
  aLine: tLine;
  aPChar: pChar;

begin
  McrAkteOeffnen:=false;
  aPChar:=aLine;
  if not GetParam(ParamLine, aPChar, 1, SizeOf(tLine)) then exit;
  ClipQuotes(aPChar);
  ShowStatus('Öffnen...', apChar, false);
  if AkteLaden(apChar) then
  begin
    FileExpand(CurFileName, aPChar);
    McrAkteOeffnen:=true;
    TheMainWnd^.SetTitle(apChar);
  end;
  ShowStatus(nil, nil, false);
end;
```

function McrAkteNeu: boolean;

```
{Funktion legt eine neue Falldatendatei an.}

begin
  {Mit AkteNeu legt das DB-Modul einen neuen Fall an.}
  McrAkteNeu:=AkteNeu;

  {Der aktuelle Dateiname wird gestrichen.}
  StrCopy(CurFileName, #0);

  {Der Titel wird auf [ohne Namen] gesetzt.}
  TheMainWnd^.SetTitle(nil);
  ShowStatus(nil, nil, false);
end;
```

```
function McrBeziehungSpeichern(ParamLine: pChar): boolean;

{Die Funktion legt eine neue Relation an.

Der Rückgabewert bezeichnet den Erfolg.

Syntax: @RelationAnlegen(Term($HauptEty;$Ety1, ...;Standardattribute))}

var
  TrmLine, aLine: tLine;
  TrmPChar, aPChar: pChar;
  aTrmID, TrmAttrC: LongInt;
  anEtyRel: tBeziehung;
  aXTrm: tXRelation;
  aWord, wObjTreffer: word;
  aPosition: tPosition;
  hRelation, hBeziehung: longint;

{  anAttach: tAttach;}

begin
  {Initialisierung}
  McrBeziehungSpeichern:=false;
  ShowStatus('Relation anlegen:', nil, false);
  aPChar:=aLine; TrmPChar:=TrmLine;

  aPosition:=dummyPosition;

{  anAttach:=DummyAttach;}
  ReadPosition(ParamLine, aPosition);

  {Der erste Parameter enthält die Relation.
  Diese wird in TrmPChar kopiert}
  GetParam(ParamLine, TrmPChar, 1, SizeOf(tLine));

  {Nun wird die Parameterleiste der Relation in aPChar kopiert.}
  SplitCmdLine(TrmPChar, aPChar);

  {Suche den Term in TrmPChar}
  aTrmID:=RelationAusParameterLesen(TrmPChar, 1); if aTrmID=0 then exit; {Term suchen}

  {Lese den Term in aXTrm ein}
  if not RelationLesen(aTrmID, aXTrm) then exit; {Exit bei Fehler}

  {Rufe die Anzahl der Attribute ab.}
  TrmAttrC:=RelationParamAnzahl(aXTrm.relXRelation);
  with anEtyRel do
  begin
    {Der neuen Relation wird der aktuelle Eingabefilter mitgegeben.}
    FltBezWert:=fltAktEingFlt;
    XbBezWert:=xBoDmy;
    LBezFlags:=0;

    {Lese die Hauptentität.
    Die Entitäten der Relation müssen explizit vorhanden sein.}
    LBezHauptObj:=SucheParamObjekt(aPChar, 1, false, hRelation, hBeziehung, wObjTreffer);
    if LBezHauptObj<=0 then exit; {Referenz Ety}

    {Lese nun die weiteren Entitäten der Relation}
    if TrmAttrC>0 then for aWord:=1 to TrmAttrC do
    begin
      LbfBezObj[aWord]:=SucheParamObjekt(aPChar, aWord+1, false, hRelation, hBeziehung, wObjTreffer); {Attr-Ety}
      if LbfBezObj[aWord]<=0 then exit;
    end;
  end;

  {Anfügen der Relation mit der DB-Funktion BeziehungZufuegen}
  If BeziehungZufuegen(anEtyRel, aXTrm)=0 then exit; {Relation speichern}

  {Nun muß die Veränderung auch im zugrundeliegenden Term gespeichert werden.}
  if not RelationSchreiben(aTrmID, aXTrm) then exit; {Term speichern}
  ShowStatus(nil, nil, false);
  McrBeziehungSpeichern:=true;
end;
```

function ExecuteCommand(McrCmdLine: pChar): boolean;

{Die Funktion verteilt die Makrobefehle auf die einzelnen Funktionen.}

```
var
  szParameter, CmdName: PChar;
  aBf: tLine;
  aWord: Word;
  aPos: word;
  aSort: LongInt;

begin
  ExecuteCommand:=false;
  if (McrCmdLine[0]<>'[') or (McrCmdLine[StrLen(McrCmdLine)-1]<>']') then exit;
  McrCmdLine:=McrCmdLine+1; McrCmdLine[StrLen(McrCmdLine)-1]:=#0;
  CmdName:=aBf; GetParam(McrCmdLine, CmdName, 1, SizeOf(tLine));
  if not SplitCmdLine(CmdName, szParameter) then exit;
  if CmdName[0]<>prefix_Funktion then exit;  CmdName:=aBf+1;
  if lStrCmpI(CmdName, makro_ObjektZufuegen)=0 then ExecuteCommand:=McrObjektAnfuegen(szParameter);
  if lStrCmpI(CmdName, makro_AkteSpeichernUnter)=0 then ExecuteCommand:=McrAkteSpeichern(szParameter);
  if lStrCmpI(CmdName, makro_AkteLaden)=0 then ExecuteCommand:=McrAkteOeffnen(szParameter);
  if lStrCmpI(CmdName, frage_BezSchreiben)=0 then
    ExecuteCommand:=McrBeziehungSpeichern(szParameter);
  if lStrCmpI(CmdName, makro_AkteNeu)=0 then ExecuteCommand:=McrAkteNeu;
  if lStrCmpI(CmdName, makro_SaveBase)=0 then ExecuteCommand:=DatenbankSichern;
end;

{---------------------------------DDE-Request-Funktionen---------------------------------}
```

function AntwZuSystem(szFrage, aBuffer: pChar; BfSz: word): boolean;

{Die Funktion beantwortet die DDE-Systemanfragen mit lapidaren Standardantworten.

Das Ergebnis wird in aBuffer geschrieben}

```
begin
  AntwZuSystem:=true;
  if lStrCmpI(szFrage, sysp_AktuelleAkte)=0 then
  begin
    if CurFileName[0]=#0 then AntwZuSystem:=false
    else StrlCopy(aBuffer, CurFileName, BfSz);
  end;
  if lStrCmpI(szFrage, szddesys_Item_SysItems)=0 then StrlCopy(aBuffer, sysp_SysItemsList, BfSz);
  if lStrCmpI(szFrage, szddesys_Item_Topics)=0 then StrlCopy(aBuffer, sysp_TopicsList, BfSz);
  if lStrCmpI(szFrage, szddesys_Item_Status)=0 then StrlCopy(aBuffer, 'Ready', BfSz);
  if lStrCmpI(szFrage, szddesys_Item_Formats)=0 then StrlCopy(aBuffer, sysp_FormatsList, BfSz);
end;
```

function AntwZuObjekt(szFrage, szParameter, aBuffer: pChar; BfSz: word): boolean;

{Die Funktion beantwortet unterschiedliche Fragen, bei denen Informationen zu einem Objekt abgerufen werden. Der Rückgabewert bezeichnet den Erfolg. Das Ergebnis wird in den Puffer aBuffer kopiert.}

```
var
  anId: LongInt;
  einObjekt: tObjekt;
  wObjTreffer: word;
  aBf: array [0..1023] of char;
  hRelation, hBeziehung: longint;

begin
  AntwZuObjekt:=false;

  {SucheParamObjekt sucht nach der Entität, die durch einen definierten Parameter
  beschrieben wird. Die Funktion läßt rekursive Verschachtelungen zu.}
  anId:=SucheParamObjekt(szParameter, 1, false, hRelation, hBeziehung, wObjTreffer);
  if anId<=0 then exit; {Verschoben am 15.01.96}

  if lStrCmpI(szFrage, frage_AnzahlObjekte)=0 then
  begin
    {Anzahl der Entitäten. (Wird von SucheParamObjekt mitgeliefert)}
    NumToStr(aBuffer, wObjTreffer, 0, BfSz-1);
    AntwZuObjekt:=true;
  end;
```

```
 if lStrCmpI(szFrage, frage_ObjektIndex)=0 then
 {Index der Entität}
 begin
  NumToStr(aBuffer, anId, 0, BfSz-1);
 end;

 if lStrCmpI(szFrage, frage_ObjektBezeichner)=0 then
 {Bezeichner der Entität}
 begin
  {ObjektLesen ermittelt den Bezeichner der Entität}
  if not ObjektLesen(anId, einObjekt) then exit;
  StrlCopy(aBuffer, einObjekt.SzObjBez, BfSz);
 end;
 AntwZuObjekt:=true;
end;
```

function AntwZuRelation(szFrage, szParameter, aBuffer: pChar; BfSz: word): boolean;

```
{Funktion beantwortet einzelne Anfragen bezüglich eines Terms.

Der Rückgabewert bezeichnet den Erfolg, das Ergebnis winr nach aBuffer kopiert.}

var
 aLine: tLine;
 aLongInt: LongInt;
 aReal: Real;
 anId: LongInt;
 aXTrm: tXRelation;

begin
 AntwZuRelation:=false;
 {Sucht den ersten Term in einer Kommandozeile.}
 anId:=RelationAusParameterLesen(szParameter, 1); if anId=0 then exit;
 {Der Term wird aus der DB ermittelt.}
 if not RelationLesen(anId, aXTrm) then exit;

 {Nun werden die unterschiedlichen Fragen abgearbeitet}

 {Index des beschriebenen Terms.}
 if lStrCmpI(szFrage, frage_RelIndex)=0 then
  NumToStr(aBuffer, anId, 0, BfSz-1);

 {Bezeichner des beschriebenen Terms.}
 if lStrCmpI(szFrage, frage_RelBezeichner)=0 then
  StrlCopy(aBuffer, aXTrm.RelXRelation.SzRelBez, BfSz);

 {Anzahl der Attribute eines beschriebenen Terms}
 if lStrCmpI(szFrage, frage_RelAnzahlAttribute)=0 then
  NumToStr(aBuffer, RelationParamAnzahl(aXTrm.relXRelation), 0, BfSz-1);

 {Auslesen der Attribute eines Terms}
 if (lStrCmpI(szFrage, frage_RelAttributIndex)=0)
  or (lStrCmpI(szFrage, frage_RelAttributBezeichner)=0) then
 begin
  {Lies den zweiten Parameter der Funktion.
  Hier steht der Index des Terms des entsprechenden Parameters}
  if not GetParam(szParameter, aLine, 2, SizeOf(tLine)) then exit;
  {Nun wird die ID des in dem Parameter gefundenen Terms ermittelt.}
  StrToNum(aLine, aReal); anId:=trunc(aReal);
  {Jetzt kann das Attribut gelesen werden.}
  if not RelationParamLesen(anId, aLongInt, aXTrm.relXRelation) then exit;

  if lStrCmpI(szFrage, frage_RelAttributIndex)=0 then
   {Hier wird der Index ohne # in den Puffer kopiert}
   NumToStr(aBuffer, aLongInt, 0, BfSz-1)
  else begin
   {Hier wird der Bezeichner in den Puffer kopiert}
   if not RelationLesen(aLongInt, aXTrm) then exit;
   StrlCopy(aBuffer, aXTrm.RelXRelation.SzRelBez, BfSz);
  end;
 end;
 AntwZuRelation:=true;
end;

{Vorbereitende Funktionen für AntwZuBeziehung}
```

function BeziehungAuslesen(szParameter: pChar; **var** aTrmID: LongInt; **var** aXTrm: tXRelation; **var** anEtyRel: tBeziehung): boolean;

```
{Die Funktion liest in die Variable "anEtyRel" die Informationen ein, die sich

aus dem Kommando pChar ergeben.}

var
  aACount: LongInt;
  aPChar: pChar;
  wObjTreffer, aWord: Word;
  TrmLine: tLine;
  hRelation, hBeziehung: longint;

begin
  BeziehungAuslesen := false;
  {Lege zunächst eine Kopie von szParameter in TrmLine ab}
  StrlCopy(TrmLine, szParameter, SizeOf(tLine));
  {Trenne die Parameterzeile ab und verschiebe sie nach aPChar.}
  SplitCmdLine(TrmLine, aPChar);

{  MessageBox(0, TrmLine, aPChar, 0);}

  {Lese den der Relation zugrundeliegenden Term.}
  aTrmId:=RelationAusParameterLesen(TrmLine, 1); if aTrmId=0 then exit;
  {Lese die Daten zu diesem Term}
  if not RelationLesen(aTrmId, aXTrm) then exit;

  {Lese die Anzahl der sekundären Attribute des Terms}
  aACount:=RelationParamAnzahl(aXTrm.relXRelation);

  with anEtyRel do
  begin
    {Lese zunächst die Haupt-Entität der konkreten Relation.}
    LBezHauptObj:=SucheParamObjekt(aPChar, 1, false, hRelation, hBeziehung, wObjTreffer);
    if LBezHauptObj<0 then exit;

    {Lese nun die Entitäten zu den weiteren Attributen}
    if aACount>0 then for aWord:=1 to aACount do
      begin
        LbfBezObj[aWord]:=SucheParamObjekt(aPChar, aWord+1, false, hRelation, hBeziehung, wObjTreffer);
        if LbfBezObj[aWord]<0 then
        begin
          LbfBezObj[aWord]:=Obj_dummy;
        end
      end;

    FltBezWert:=fltAktEingFlt; {eigentlich unnötig, da fltAktEingFlt immer Referenz}
    XbBezWert:=xBoDmy;
  end;
  BeziehungAuslesen:=true;
end;
```

function AnzahlPassendeBeziehungen(aXTrm: tXRelation; RefEtyRel: tBeziehung; aPosition: tPosition): LongInt;

```
var
  HitCo, aWord: word;
  anEtyRel: tBeziehung;
  aACount, aCCount: Word;
  anInt: Integer;
  MsgLine: tLine;

begin
  AnzahlPassendeBeziehungen := 0;
  HitCo:=0;
  aCCount:=BeziehungenAnzahl(aXTrm);   {Anzahl der Verknüpfungen eines Terms}
  aACount:=RelationParamAnzahl(aXTrm.relXRelation);   {Anzahl der Attribute eines Terms}
  if aCCount>0 then for aWord:=1 to aCCount do
  begin
    BeziehungLesen(aWord, anEtyRel, aXTrm);
    if VergleicheBeziehungen(anEtyRel, RefEtyRel, aACount) then  hitCo:=hitCo+1;
  end;
  AnzahlPassendeBeziehungen:=HitCo;
end;
```

function PassendeBeziehung(aXTrm: tXRelation; RefEtyRel: tBeziehung; aPosition: tPosition): LongInt;

```
var
  HitCo, aHit, aWord: word;
  anEtyRel: tBeziehung;
  aACount, aCCount: Word;
  anInt: Integer;
  MsgLine: tLine;

begin
  PassendeBeziehung := -1;
  HitCo:=0; aHit:=0;
  aCCount:=BeziehungenAnzahl(aXTrm);   {Anzahl der Verknüpfungen eines Terms}
  aACount:=RelationParamAnzahl(aXTrm.relXRelation);   {Anzahl der Attribute eines Terms}
  if aCCount>0 then for aWord:=1 to aCCount do
  begin
    BeziehungLesen(aWord, anEtyRel, aXTrm);
    if VergleicheBeziehungen(anEtyRel, RefEtyRel, aACount) then
    begin
      hitCo:=hitCo+1;
      if (HitCo=aPosition.PosIndex) or (aPosition.PosIndex=0) then aHit:=aWord;
    end
  end;

  anInt:=0;
  if (HitCo>0) and (aPosition.PosIndex<0) then
    for aWord:=aCCount downto 1 do
  begin
     anInt:=anInt-1;
    BeziehungLesen(aWord, anEtyRel, aXTrm);
    if VergleicheBeziehungen(anEtyRel, RefEtyRel, aACount) then
    begin
      if (anInt=aPosition.PosIndex) then aHit:=aWord;
    end;
  end;
  PassendeBeziehung:=aHit;
end;
```

function AntwZuBeziehung(szFrage, szParameter, aBuffer: pChar; aPosition: tPosition; BfSz: word): boolean;

```
var
  aTrmId, aConId: LongInt;
  aXTrm: tXRelation;
  RefEtyRel: tBeziehung;
  {aPChar: pChar;}
  TheHit: word;

begin
  AntwZuBeziehung:=false;
  if Not BeziehungAuslesen(szParameter, aTrmID, aXTrm, RefEtyRel) then exit;
  if lStrCmpI(szFrage, frage_BeziehungWert)=0 then
  begin
    TheHit:=PassendeBeziehung(aXTrm, RefEtyRel, aPosition);
    if TheHit<0 then exit;
    if not BeziehungLesen(TheHit, RefEtyRel, aXTrm) then exit;
    XBoolToStr(aBuffer, RefEtyRel.XbBezWert, BfSz-1);
  end;
  if lStrCmpI(szFrage, frage_AnzahlBeziehungen)=0 then
  begin
    TheHit:=AnzahlPassendeBeziehungen(aXTrm, RefEtyRel, aPosition);
    NumToStr(aBuffer, TheHit, 0, BfSz-1);
  end;
  AntwZuBeziehung:=true;
end;
```

function AntwZuWert(szFrage, szParameter, aBuffer: pChar; aPosition: tPosition; BfSz: word): boolean;

```
var
```

```
{  anEtyLine, szParameter: PChar;}
  wObjTreffer, VCo, aWord, TheHit, HitCo, y, m, d, vt: word;
  Hit: Integer;
  anId, aLongInt: LongInt;
  einObjekt: tObjekt;
  einObjektWert: tObjektWert;
  szRelation: tLine;
  aPChar: pChar;
  RelXRelation: tXRelation;
  hRelation, hBeziehung: longint;

begin
  {Initialisierung}
  AntwZuWert:=false;
  vt:=wt_id;

  {Auslesen eines eventuellen Positionsparameters}
  ReadPosition(szParameter, aPosition);

  if szFrage=nil then
  begin
    {wenn szFrage NULL ist, dann wird der Wertetyp
    anhand des Standardtypos der Relation bestimmt.
    Andernfalls gibt szFrage den gesuchten Wertetyp vor.}

    {Legen wir eine Kopie der Command Line an.
    Damit kann der Relationsbezeichner isoliert werden.}
    StrLCopy(szRelation, szParameter, SizeOf(tLine)-1);

    {Splitten der Kopie in Relationsbezeichner und Attribute}
    SplitCmdLine(szRelation, aPChar);

    {Lesen des Indexes der Relation}
    anId:=RelationIndexLesen(szRelation);

    {Fehlerbehandlung}
    if anId<1 then exit;

    {Lesen der Relationsinformationen}
    if not RelationLesen(anId, RelXRelation) then exit;

    {Setzen des gesuchten Wertetyps anhand des Standardtyps}
    with RelXRelation.RelXRelation do
    begin
      if LRelFlags = tty_xbool then vt:=wt_exist;
      if (LRelFlags = tty_integer) or (LRelFlags = tty_real) then vt:=wt_number;
      if LRelFlags = tty_date then vt:=wt_date;
      if LRelFlags = tty_text then vt:=wt_string;
    end;
  end else {szFrage = nil}

  begin
    {In diesem Fall ergebt sich der gesuchte Wertetyp aus der
    angegebenen Funktion in szFrage.}
    if !StrCmpI(szFrage, frage_ObjektWertExist)=0 then vt:=wt_exist;
    if !StrCmpI(szFrage, frage_ObjektWertString)=0 then vt:=wt_string;
    if !StrCmpI(szFrage, frage_ObjektWertNumber)=0 then vt:=wt_number;
    if !StrCmpI(szFrage, frage_ObjektWertDate)=0 then vt:=wt_date;
    if !StrCmpI(szFrage, frage_Objekt)=0 then vt:=wt_id;
  end;

  {In der Parameterzeile wird nun das Hauptobjekt gesucht}
  anId:=SucheParamObjekt(szParameter, 1, false, hRelation, hBeziehung, wObjTreffer);
  {Wenn kein Objekt gefunden wurde, wird die Funktion verlassen}
  if anId<=0 then exit;

  {Nun wird die Objektinformation gelesen}
  if not ObjektLesen(anId, einObjekt) then exit;

  {Das ist durchaus noch verbesserungsfähig.
  Die Frage nach dem Objektbezeichner (@Objekt) stellt eine Ausnahme dar.
  Hier müssen nicht umständlich die Werte ausgelesen werden.
  Die Frage wird also vorab behandelt.}
  if vt=wt_id then
  begin
    {Kopiere den Objektbezeichner in den Ausgabepuffer}
    StrLCopy(aBuffer, einObjekt.szObjBez, BfSz-1);
    {Bestätige den Erfolg der Frage}
    AntwZuWert:=true;
```

```
  {Verlasse die Funktion}
  exit;
end;

{Ab hier werden nun die eigentlichen Objektwerte geprüft.}

{Initialisieren der Trefferzähler}
TheHit:=0;
HitCo:=0;
{Ermitteln der Anzahl aller verfügbaren Werte des Objekts}
VCo:=ObjektWerteAnzahl(einObjekt);

{Wenn's keine Werte gibt, gibt's keine Antwort}
if VCo <= 0 then exit;

{Überprüfen der einzelnen Werte}
if (VCo>0) and (aPosition.PosIndex>=0) then
begin
  for aWord:=1 to VCo do
  begin
    {Einlesen des aWord-sten Objektwertes}
    ObjektWertLesen(aWord, einObjektWert, einObjekt);
    with einObjektWert do
    begin
      {Überprüfen des Werttyps und Abgleich der Ein-/Ausgabefilter}
      if (WWertTyp=vt) and (VergleicheMitAusgabeFlt(FltObjWert) = cat_equal) then
      begin
        {Hochzählen der Treffer}
        HitCo:=HitCo+1;
        {Überprüfen des Positionsparameters}
        if (HitCo=aPosition.PosIndex) or (aPosition.PosIndex=0)
        then TheHit:=aWord;

        {AGENDA: Sortierung fehlt}
      end;
    end;
  end;
end;

{Die folgende Schleife ist nötig, falls der Positionsparameter negativ ist.
Dann wird die Position von hinten gezählt.}
Hit:=0;
if (VCo>0) and (aPosition.PosIndex<0) then
begin
  for aWord:=VCo DownTo 1 do
  begin
    ObjektWertLesen(aWord, einObjektWert, einObjekt);
    with einObjektWert do
    begin
      if (WWertTyp=vt) and (VergleicheMitAusgabeFlt(FltObjWert) = cat_equal) then
      begin
        Hit:=Hit-1;
        if (Hit = aPosition.PosIndex) then TheHit:=aWord;

{Sortierung fehlt}
      end;
    end;
  end;
end;

if TheHit<1 then
begin
  {In der folgenden Schleife werden die Filter
  zur Ermittlung des Wertes herangezogen.
  Die Schleife wird nur durchlaufen, falls der Wert nicht schon feststeht.}

{AGENDA: Filter. Hier nur Test der AtlUsrDg.DLL}
  for aWord:=1 to 16 do with TheFltLst[aWord] do
  begin
    if @FltCallBack<>nil then
    begin
      FltCallBack(fct_GetVal, 0, anId, LongInt(@einObjektWert), 0);

{AGENDA: ?}
    end;
  end;
end else {TheHit<1}
begin
```

```
    {Falls der Objektwert feststeht, wird er hier nochmals gelesen}
    ObjektWertLesen(TheHit, einObjektWert, einObjekt);
  end;

  {Hier wird der Objektwert in den Puffer kopiert.
  Dabei werden die Wertetypen konvertiert}
  with einObjektWert do
  begin
    if vt=wt_string then StrLCopy(aBuffer, SzWertZeichen, BfSz-1);
    if vt=wt_exist then XBoolToStr(aBuffer, XbWertExist, BfSz-1);
    if vt=wt_number then NumToStr(aBuffer, DbWertZahl, 2, BfSz-1);
    if vt=wt_date then
    begin
      NumToDate(LWertDatum, y, m, d);
      DateToStr(aBuffer, y, m, d, BfSz-1);
    end;
  end;

  {Bestätige den Erfolg der Frage}
  if TheHit>=1 then AntwZuWert:=true;
end;
```

function AntwZuEingFlt(szFrage, aBuffer: pChar; BfSz: word): boolean;

```
var
  einObjekt: tObjekt;
  aLine: tLine;
  y, m, d: word;

begin
  AntwZuEingFlt:=false;
  if (aBuffer=nil) or (BfSz<1) then exit;
  aBuffer[0]:=#0;
  with fltAktEingFlt do
  begin
    if ObjDatAnfang>0 then
    begin
      if !StrCmpI(szFrage, frage_Eingabefilter)=0 then
      begin
        if not ObjektLesen(ObjDatAnfang, einObjekt) then exit;
        StrLCat(aBuffer, einObjekt.SzObjBez, BfSz);
      end else
      begin
        NumToStr(aLine, ObjDatAnfang, 0, SizeOf(tLine)-1);
        StrLCat(aBuffer, aLine, BfSz);
      end;
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;
    if ObjDatEnde>0 then
    begin
      if !StrCmpI(szFrage, frage_Eingabefilter)=0 then
      begin
        if not ObjektLesen(ObjDatEnde, einObjekt) then exit;
        StrLCat(aBuffer, einObjekt.SzObjBez, BfSz);
      end else
      begin
        NumToStr(aLine, ObjDatEnde, 0, SizeOf(tLine)-1);
        StrLCat(aBuffer, aLine, BfSz);
      end;
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;
    if ObjQuelle>0 then
    begin
      if !StrCmpI(szFrage, frage_Eingabefilter)=0 then
      begin
        if not ObjektLesen(ObjQuelle, einObjekt) then exit;
        StrLCat(aBuffer, einObjekt.SzObjBez, BfSz);
      end else
      begin
        NumToStr(aLine, ObjQuelle, 0, SizeOf(tLine)-1);
        StrLCat(aBuffer, aLine, BfSz);
      end;
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;
    if LEingabeDat>0 then
```

```
    begin
      NumToDate(LEingabeDat, y, m, d); DateToStr(aLine, y, m, d, SizeOf(tLine)-1);
      StrLCat(aBuffer, aLine, BfSz);
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;
    if LEFltFlags>0 then
    begin
      if isFlag(LEFltFlags, sa_exact) then
        begin StrLCat(aBuffer, frage_SAFExact, BfSz);StrLCat(aBuffer, ' ', BfSz); end;
      if isFlag(LEFltFlags, sa_InclDmy) then
        begin StrLCat(aBuffer, frage_SAFInclDmy, BfSz);StrLCat(aBuffer, ' ', BfSz); end;
      if isFlag(LEFltFlags, sa_Dummy) then
        begin StrLCat(aBuffer, frage_SAFDummy, BfSz);StrLCat(aBuffer, ' ', BfSz); end;
    end;
  end;
  AntwZuEingFlt:=true;
end;
```

function AntwZuAusgFlt(szFrage, aBuffer: pChar; BfSz: word): boolean;

```
var
  einObjekt: tObjekt;
  aLine: tLine;
  y, m, d: word;

begin
  AntwZuAusgFlt:=false;
  if (aBuffer=nil) or (BfSz<1) then exit;
  aBuffer[0]:=#0;
  with CurSrcAttr do
  begin
    if LGeltDatAnfang>0 then
    begin
      NumToDate(LGeltDatAnfang, y, m, d); DateToStr(aLine, y, m, d, SizeOf(tLine)-1);
      StrLCat(aBuffer, aLine, BfSz);
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;

    if LGeltDatEnde>0 then
    begin
      NumToDate(LGeltDatEnde, y, m, d); DateToStr(aLine, y, m, d, SizeOf(tLine)-1);
      StrLCat(aBuffer, aLine, BfSz);
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;

    if ObjEingabeQuelle>0 then
    begin
      if not ObjektLesen(ObjEingabeQuelle, einObjekt) then exit;
      StrLCat(aBuffer, einObjekt.SzObjBez, BfSz);
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;

    if LEingabeDatAnfang>0 then
    begin
      NumToDate(LEingabeDatAnfang, y, m, d); DateToStr(aLine, y, m, d, SizeOf(tLine)-1);
      StrLCat(aBuffer, aLine, BfSz);
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;

    if LEingabeDatEnde>0 then
    begin
      NumToDate(LEingabeDatEnde, y, m, d); DateToStr(aLine, y, m, d, SizeOf(tLine)-1);
      StrLCat(aBuffer, aLine, BfSz);
    end;
    StrLCat(aBuffer, ';', BfSz); if StrLen(aBuffer)>=BfSz then exit;

    if LAusgFltFlags>0 then
    begin
      if isFlag(LAusgFltFlags, sa_exact) then
        begin StrLCat(aBuffer, frage_SAFExact, BfSz);StrLCat(aBuffer, ' ', BfSz); end;
      if isFlag(LAusgFltFlags, sa_InclDmy) then
        begin StrLCat(aBuffer, frage_SAFInclDmy, BfSz);StrLCat(aBuffer, ' ', BfSz); end;
      if isFlag(LAusgFltFlags, sa_Dummy) then
        begin StrLCat(aBuffer, frage_SAFDummy, BfSz);StrLCat(aBuffer, ' ', BfSz); end;
    end;
```

```
  end;
  AntwZuAusgFlt:=true;
end;


function DdeAnswerRequest(szFrage, aBuffer: pChar; BfSz: word): boolean;

var
  szParameter, CmdName: PChar;
  aPosition: tPosition;
  aBf, aLine: tLine;
  aFncId: LongInt;
  aWord: word;

begin
  DdeAnswerRequest:=false; CmdName:=aBf;
  aPosition:=dummyPosition;
  ReadPosition(szFrage, aPosition);
  StrCopy(aBuffer, '');

{?}  GetParam(szFrage, CmdName, 1, SizeOf(tLine));

  if CmdName[0]<>prefix_Funktion then
  begin
    DdeAnswerRequest:=AntwZuWert(nil, CmdName, aBuffer, aPosition, BfSz);
    exit;
  end;

  if not SplitCmdLine(CmdName, szParameter) then exit;
  CmdName:=CmdName+1;

  if (lStrCmpI(CmdName, frage_ObjektIndex)=0)
    or (lStrCmpI(CmdName, frage_ObjektBezeichner)=0)
    or (lStrCmpI(CmdName, frage_AnzahlObjekte)=0) then
    DdeAnswerRequest:=AntwZuObjekt(CmdName, szParameter, aBuffer, BfSz);

  if (lStrCmpI(CmdName, frage_RelIndex)=0)
    or (lStrCmpI(CmdName, frage_RelBezeichner)=0)
    or (lStrCmpI(CmdName, frage_RelAttributIndex)=0)
    or (lStrCmpI(CmdName, frage_RelAttributBezeichner)=0)
    or (lStrCmpI(CmdName, frage_RelAnzahlAttribute)=0) then
    DdeAnswerRequest:=AntwZuRelation(CmdName, szParameter, aBuffer, BfSz);

  if (lStrCmpI(CmdName, frage_BeziehungWert)=0)
    or (lStrCmpI(CmdName, frage_AnzahlBeziehungen)=0) then
    DdeAnswerRequest:=AntwZuBeziehung(CmdName, szParameter, aBuffer, aPosition, BfSz);

  if (lStrCmpI(CmdName, frage_ObjektWertExist)=0)
    or (lStrCmpI(CmdName, frage_ObjektWertString)=0)
    or (lStrCmpI(CmdName, frage_ObjektWertNumber)=0)
    or (lStrCmpI(CmdName, frage_Objekt)=0)
    or (lStrCmpI(CmdName, frage_ObjektWertDate)=0) then
    DdeAnswerRequest:=AntwZuWert(CmdName, szParameter, aBuffer, aPosition, BfSz);

  if (lStrCmpI(CmdName, frage_Eingabefilter)=0)
    or (lStrCmpI(CmdName, frage_EingabefilterIx)=0) then
    DdeAnswerRequest:=AntwZuEingFlt(CmdName, aBuffer, BfSz);

  if (lStrCmpI(CmdName, frage_Ausgabefilter)=0) then
    DdeAnswerRequest:=AntwZuAusgFlt(CmdName, aBuffer, BfSz);

  if (lStrCmpI(CmdName, sysp_AktuelleAkte)=0)
    or (lStrCmpI(CmdName, szddesys_Item_SysItems)=0)
    or (lStrCmpI(CmdName, szddesys_Item_Topics)=0)
    or (lStrCmpI(CmdName, szddesys_Item_Status)=0)
    or (lStrCmpI(CmdName, szddesys_Item_Formats)=0) then
    DdeAnswerRequest:=AntwZuSystem(CmdName, aBuffer, BfSz);

{AGENDA Filter: Hier nur Test der AtlUsrDg.DLL}
  for aWord:=1 to 16 do with TheFltLst[aWord] do
  begin
    if @FltCallBack<>nil then
    begin
      aFncId:= FltCallBack(fct_GetFncName, 0, 0, LongInt(CmdName), 0);
      if aFncID>0 then
      begin
        DdeAnswerRequest:=bool(FltCallBack(fct_ExecFnc, BfSz, 1, LongInt(szFrage), LongInt(aBuffer)));
      end;
```

```
{...}
   end;
  end;
end;

{DDE-Poke-Funktionen}
```

function DdePokeVal(CmdName, szParameter, aDataBf: pChar;

```
{  anAttach: tAttach;}
  aPosition: tPosition): boolean;

var
  aPChar: PChar;
  RelXRelation: tXRelation;
  szRelation: tLine;
  add: boolean;
  cEty: xBool;
  einObjektWert: tObjektWert;
  wObjTreffer, y, m, d: word;
  anId: LongInt;
  einObjekt: tObjekt;
  hRelation, hBeziehung: longint;

begin
  DdePokeVal:= false;
  {cEty:=Xtrue;} cEty:=XBoDmy;
  add:=false;
  with einObjektWert do
  begin
    FltObjWert:=fltAktEingFlt;
    if CmdName=nil then
    begin
      StrLCopy(szRelation, szParameter, SizeOf(tLine)-1); SplitCmdLine(szRelation, aPChar);
      anId:=RelationIndexLesen(szRelation); if anId<1 then exit; if not RelationLesen(anId, RelXRelation) then exit;
      with RelXRelation.RelXRelation do
      begin
        if LRelFlags = tty_xbool then
        begin
          WWertTyp:=wt_exist; StrToXBool(aDataBf, XbWertExist); add:=true; cEty:=XbWertExist;
        end;
        if (LRelFlags = tty_integer) or (LRelFlags = tty_real) then
          begin WWertTyp:=wt_number; StrToNum(aDataBf, DbWertZahl);  add:=true; end;
        if LRelFlags = tty_date then
        begin
          WWertTyp:=wt_date; StrToDate(aDataBf, y, m, d);
          DateToNum(LWertDatum, y, m, d); add:=true;
        end;
        if LRelFlags = tty_text then
          begin WWertTyp:=wt_string; StrLCopy(SzWertZeichen, aDataBf, SizeOf(tLine)-1); add:=true; end;
      end;
    end else
    begin
      if !StrCmpI(CmdName, frage_ObjektWertExist)=0 then
      begin
        WWertTyp:=wt_exist; StrToXBool(aDataBf, XbWertExist); add:=true; cEty:=XbWertExist;
      end;
      if !StrCmpI(CmdName, frage_ObjektWertString)=0 then
        begin WWertTyp:=wt_string; StrLCopy(SzWertZeichen, aDataBf, SizeOf(tLine)-1); add:=true; end;
      if !StrCmpI(CmdName, frage_ObjektWertNumber)=0 then
        begin WWertTyp:=wt_number; StrToNum(aDataBf, DbWertZahl);  add:=true; end;
      if !StrCmpI(CmdName, frage_ObjektWertDate)=0 then
      begin
        WWertTyp:=wt_date; StrToDate(aDataBf, y, m, d);
        DateToNum(LWertDatum, y, m, d); add:=true;
      end;

{...}
   end;
  end;
  if add then
  begin

{messagebox(0, szParameter,'Hey',0);}
    anId := SucheParamObjekt(szParameter, 1, true, hRelation, hBeziehung, wObjTreffer);
```

```
    if anId <= 0 then exit;
    if not ObjektLesen(anId, einObjekt) then exit;
    if not ObjektWertZufuegen(einObjektWert, einObjekt)>0 then exit;
    if not ObjektSchreiben(anId, einObjekt) then exit;
    DdePokeVal:=true;
  end;
  DdePokeVal:= true;
end;
```

function DdePokeRel(CmdName, szParameter, aDataBf: pChar;

```
{  anAttach: tAttach;}
  aPosition: tPosition): boolean;

var
  lAnzPar,
  lAnzBez,
  hRel,
  hAttr
    : LongInt;

  hRelBf,
  hBezBf,
  hObjekt
    : longint;

  wObjTreffer,
  wx,
  wObjAnz,
  wAnzObjUnbek
    : word;

  wTreffer: word;

  einObjekt,
  objBf,
  objAttr
    : tObjekt;

  relx,
  relxAttr
    : tXRelation;

  bezNeu,
  bezRef,
  bezKop
    : tBeziehung;

  szPar: pChar;

  szRelation,
  aName,
  aLine
    : tLine;

begin
  {Bei frühzeitigem Abbruch standardmäßig Fehler ausgeben}
  DdePokeRel:=false;

  {Kopie der Relation anlegen}
  StrLCopy(szRelation, szParameter, SizeOf(tLine)-1);

  {Hiermit erzwingen wir die Anlage der Beziehung}
  {Parameter und Relationsbezeichner trennen}
  SplitCmdLine(szRelation, szPar);

  {Suchen nach der entsprechenden Relation}
  hRel := RelationAusParameterLesen(szRelation, 1);
  if hRel = 0 then exit;

  {Auslesen der Relation}
  if not RelationLesen(hRel, relx) then exit;

  {Ermittle die Anzahl der Parameter der Relation}
  lAnzPar := RelationParamAnzahl(relx.RelXRelation);
```

```
  {Ermittle die Anzahl der bereits vorhandenen Beziehungen}
  lAnzBez := BeziehungenAnzahl(relx);

  {Anlegen einer neuen Beziehung}
  with bezNeu do
  begin
   {Initialisieren der einzelnen Felder}
   FltBezWert := InpFltDmy;
   XbBezWert := xBoDmy;
   LBezHauptObj := -1;
   for wx := 1 to max_attrib do LbfBezObj[wx] := -1;

   {Übermitteln des Wertes der Beziehung}
   StrToXBool(aDataBf, XbBezWert);

   {Flags werden nicht genutzt}
   LBezFlags:=0;

   {Der Eingabeflilter entspricht dem global gesetzten Filter}
   FltBezWert:=fltAktEingFlt;

   {Nun werden die einzelnen Objekte ermittelt}
   {wAnzObjUnbek zählt die unbekannten Objekte}
   wAnzObjUnbek := 0;

   {Zunächst das Hauptobjekt}
   LBezHauptObj := SucheParamObjekt(szPar, 1, true, hRelBf, hBezBf, wObjTreffer);
   if LBezHauptObj <= 0 then
   begin
     wAnzObjUnbek := wAnzObjUnbek +1;

{MessageBox(0, szPar, 'Hauptobjekt Fehlt', 0);}
   end;

   {Gehen wir die einzelnen Parameter durch}
   if lAnzPar > 0 then for wx := 1 to lAnzPar do
   begin
     {Suchen wir hier den Parmater}
     LbfBezObj[wx] :=  SucheParamObjekt(szPar, wx + 1, true, hRelBf, hBezBf, wObjTreffer);
     if LbfBezObj[wx] <= 0 then
     begin
       wAnzObjUnbek := wAnzObjUnbek +1;

{MessageBox(0, szPar, 'Neben Fehlt', 0);}
     end;
   end;
  end;

  if (wAnzObjUnbek > 0) and (lAnzBez > 0) then
  begin

{MessageBox(0, szRelation, 'Beziehungen prüfen', 0);}
   {Zunächst einmal suchen wir die
   vorhandenen Beziehungen durch.
   Vielleicht paßt eine Beziehung}

   {Dazu wird eine Kopie der Beziehung angelegt}
   bezKop := bezNeu;

   {Der Wert und die Filter werden allerdings auf Dummy gesetzt}
   bezKop.xbBezWert := xBoDmy;
   bezKop.FltBezWert := InpFltDmy;

   wTreffer := 0;

   for wx := 1 to lAnzBez do
   begin
    {Einlesen der konkreten Beziehung}
    BeziehungLesen(wX, bezRef, relx);

    {Vergleichen der Beziehungen}
    if VergleicheBeziehungen(bezKop, bezRef, lAnzPar) then
    begin
      wTreffer := wx;
    end;
   end;

   if wTreffer > 0 then
   begin
    {Wenn wir eine passende Beziehung gefunden haben,
```

```
    kennen wir ja alle Objekte}
    wAnzObjUnbek := 0;

    {Lesen wir die Beziehung nochmal, um sie weiter zu verwerten}
    BeziehungLesen(wX, bezKop, relx);

    {Die Werte setzen wir auf die eigentlich gewünschten}
    bezKop.xbBezWert := bezNeu.xbBezWert;
    bezKop.FltBezWert := bezNeu.FltBezWert;

    {nun kopieren wir die kopierte Beziehung zurück in die neue Beziehung}
    bezNeu := bezKop;
   end;
  end;

  if wAnzObjUnbek > 0 then
  begin

{MessageBox(0, szRelation, 'Neue Relation anlegen', 0);}
    {Die Suche nach einer ähnlichen Beziehung hat uns nicht weiter gebracht.
    Sonst wären wir nämlich nicht hier.
    Jetzt müssen wir die Beziehung von Hand zu Fuß bauen!}
    with bezNeu do
    begin
     {Zunächst einmal schauen wir uns die Parameter an.}
     if lAnzPar > 0 then for wx := 1 to lAnzPar do
     begin
      {Suchen wir hier den Parmater}
      if LbfBezObj[wx] <= 0 then
      begin
       {Fehlt ein Attribut, dann bauen wir jetzt einfach eins}

       {Wir ermitteln hier den Bezeichner des Parameters}
       RelationParamLesen(wx, hAttr, relx.relxRelation);
       RelationLesen(hAttr, relxAttr);

       {Nun erstellen wir ein leeres Objekt}
       einObjekt := ObjLeer;

       {Der Objektbezeichner wird der Parameterbezeichner...}
       strcopy(einObjekt.SzObjBez, relxAttr.relxRelation.szRelBez);
       {...erweitert um die Anzahl der Objekte +1}
       strcat(einObjekt.SzObjBez, '/%i');
       wObjAnz := ObjekteAnzahl + 1;
       wvsprintf(einObjekt.SzObjBez, einObjekt.SzObjBez, wObjAnz);

       {Das Objekt ist virtuell}
       einObjekt.LObjFlags := Obj_virtual;

       {Nun erstellen wir das Objekt}
       LbfBezObj[wX] := ObjektZufuegen(einObjekt);
      end; {LbfBezObj[wx] <= 0}
     end; {for wx := 1 to lAnzPar do}

    {Nun überprüfen wir das Hauptobjekt}
     if LBezHauptObj <= 0 then
     begin
      {Unser Hauptobjekt fehlt}

      {Also erstellen wir ein leeres Objekt}
      einObjekt := ObjLeer;

      {Das Objekt ist virtuell}
      einObjekt.LObjFlags := Obj_virtual;

      if lAnzPar > 0 then
      begin
       {Wenn eine echte Beziehung und keine Eigenschaft vorliegt,
       {dann erstellen wir die erste Hälfte des Namens aus dem Bezeichner
       der zugrundeliegenden Relation}
       StrLCopy(einObjekt.SzObjBez, relx.RelXRelation.SzRelBez, (SizeOf(tLine) div 2)-1);

       {Den Rest des Namens entwickeln wir aus den Bezeichnern der Parameter}
       for wx := 1 to lAnzPar do
       begin
        {Einlesen des Objektes}
        ObjektLesen(LbfBezObj[wx], objBf);

        if wx > 1 then
         {Bindestrich zwischen den Objekten}
```

```
          StrLCat(einObjekt.SzObjBez, '-', SizeOf(tLine))
        else
          {Leerstelle zwischen Relationsbezeichner und Objekt}
          StrLCat(einObjekt.SzObjBez, ' ', SizeOf(tLine));

        {Anfügen des Objektnamens}
        {Die Bezeichner teilen sich die zweite Namenshälfte
        abzüglich des Platzes für die Striche}
        StrLCat(einObjekt.SzObjBez, objBf.SzObjBez,
          ((SizeOf(tLine) div 2) - lAnzPar) div lAnzPar);

      end; {for wx := 1 to wAnzPar do}
    end {lAnzPar > 0}
    else
    begin
      {Hier handelt es sich eigentlich um eine Eigenschaft}
      {Hierfür nehmen wir den Relationsbezeichner
      und hängen eine eindeutige Zahl an.}
      StrLCopy(einObjekt.SzObjBez, relx.RelXRelation.SzRelBez, SizeOf(tLine) - 6);
      strcat(einObjekt.SzObjBez, '/%i');

      {ID ist die Anzahl der bisherigen Beziehungen der Relation +1}
      {die Variable wObjAnz haben wir uns hier mal kurz ausgeliehen}
      wObjAnz := BeziehungenAnzahl(relx) + 1;
      wvsprintf(einObjekt.SzObjBez, einObjekt.SzObjBez, wObjAnz);
    end; {lAnzPar > 0}

    {nun haben wir also einen Namen}
    {jetzt erzeugen wir das Objekt}
    LBezHauptObj  := ObjektZufuegen(einObjekt);
  end; {LBezHauptObj <= 0}
 end; {with bezNeu do}
end; {wAnzObjUnbek > 0}

{Hier folgt ein Airbag}
if bezNeu.LBezHauptObj<=0 then exit;

{Jetzt müssen wir nur noch die Beziehung anlegen}
if BeziehungZufuegen(bezNeu, relx) <=0 then exit;

{Die Relation muß ebenfalls aktualisiert werden}
if not RelationSchreiben(hRel, relx) then exit;

{ToiToiToi ist alles gut gegangen}
DdePokeRel:=true;
end;
```

function DdePokeInpAttr(szFrage, aDataBf: pChar): boolean;

```
var
 y, m, d: word;
 aLine: tLine;
 wObjTreffer: word;
 anEtyId: LongInt;
 hRelation, hBeziehung: longint;

begin
 DdePokeInpAttr:=false;
 fltAktEingFlt:=InpFltDmy;
 fltAktEingFlt.LEingabeDat:=GetIDate;
 fltAktEingFlt.LEFltFlags := 0;
 with fltAktEingFlt do
 begin
  anEtyId:=SucheParamObjekt(aDataBf, 1, true, hRelation, hBeziehung, wObjTreffer);
  if anEtyId>0 then ObjDatAnfang:=anEtyId;  if anEtyId<0 then exit;
  anEtyId:=SucheParamObjekt(aDataBf, 2, true, hRelation, hBeziehung, wObjTreffer);
  if anEtyId>0 then ObjDatEnde:=anEtyId;  if anEtyId<0 then exit;
  anEtyId:=SucheParamObjekt(aDataBf, 3, true, hRelation, hBeziehung, wObjTreffer);
  if anEtyId>0 then ObjQuelle:=anEtyId; if anEtyId<0 then exit;
  if GetParam(aDataBf, aLine, 4, SizeOf(tLine)) then
  begin
    if StrToDate(aLine, y, m, d) then DateToNum(LEingabeDat, y, m, d);
  end;
  if GetParam(aDataBf, aLine, 5, SizeOf(tLine)) then
  begin
    LEFltFlags:=0;
```

```
    if StrPos(aLine, frage_SAFExact)<>nil then LEFltFlags:=LEFltFlags or sa_exact;
    if StrPos(aLine, frage_SAFInclDmy)<>nil then LEFltFlags:=LEFltFlags or sa_Incldmy;
    if StrPos(aLine, frage_SAFDummy)<>nil then LEFltFlags:=LEFltFlags or sa_dummy;
   end;
  end;
  if fltAktEingFlt.LEFltFlags = 0 then fltAktEingFlt.LEFltFlags := InpFltDmy.LEFltFlags;
  DdePokeInpAttr:=true;
end;
```

function DdePokeSrcAttr(szFrage, aDataBf: pChar): boolean;

```
var
  y, m, d: word;
  aLine: tLine;
  wObjTreffer: word;
  anEtyId: LongInt;
  hRelation, hBeziehung: longint;

begin
  DdePokeSrcAttr:=false;
  CurSrcAttr:=DummySrcAttr;
  with CurSrcAttr do
  begin
    if GetParam(aDataBf, aLine, 1, SizeOf(tLine)) then
    begin
      if StrToDate(aLine, y, m, d) then DateToNum(LGeltDatAnfang, y, m, d);
    end;
    if GetParam(aDataBf, aLine, 2, SizeOf(tLine)) then
    begin
      if StrToDate(aLine, y, m, d) then DateToNum(LGeltDatEnde, y, m, d);
    end;
    anEtyId:=SucheParamObjekt(aDataBf, 3, true, hRelation, hBeziehung, wObjTreffer);
    if anEtyId>0 then ObjEingabeQuelle:=anEtyId; if anEtyId<0 then exit;
    if GetParam(aDataBf, aLine, 4, SizeOf(tLine)) then
    begin
      if StrToDate(aLine, y, m, d) then DateToNum(LEingabeDatAnfang, y, m, d);
    end;
    if GetParam(aDataBf, aLine, 5, SizeOf(tLine)) then
    begin
      if StrToDate(aLine, y, m, d) then DateToNum(LEingabeDatEnde, y, m, d);
    end;
    if GetParam(aDataBf, aLine, 6, SizeOf(tLine)) then
    begin
      LAusgFltFlags:=0;
      if StrPos(aLine, frage_SAFExact)<>nil then LAusgFltFlags:=LAusgFltFlags or sa_exact;
      if StrPos(aLine, frage_SAFInclDmy)<>nil then LAusgFltFlags:=LAusgFltFlags or sa_Incldmy;
      if StrPos(aLine, frage_SAFDummy)<>nil then LAusgFltFlags:=LAusgFltFlags or sa_dummy;
    end;
  end;
  if CurSrcAttr.LAusgFltFlags = 0 then CurSrcAttr.LAusgFltFlags := DummySrcAttr.LAusgFltFlags;
  DdePokeSrcAttr:=true;
end;
```

function DdePokeData(szFrage, aDataBf: pChar): boolean;

```
var
  szParameter, CmdName: PChar;
  aBf: tLine;
  aPosition: tPosition;

{  anAttach: tAttach;}

begin
  DdePokeData:=false; CmdName:=aBf;
  {Auslesen der Standardattribute, Syntax: @Name(Ety);@Infos(...)}
  aPosition:=dummyPosition;
  ReadPosition(szFrage, {anAttach, }aPosition);
  GetParam(szFrage, CmdName, 1, SizeOf(tLine));
  if CmdName[0]<>prefix_Funktion then
  begin
```

```
    DdePokeData:=DdePokeVal(nil, CmdName, aDataBf, {anAttach, }aPosition);
    exit;
  end;
  if not SplitCmdLine(CmdName, szParameter) then exit;
  CmdName:=CmdName+1;
  if (lStrCmpI(CmdName, frage_ObjektWertExist)=0) or
    (lStrCmpI(CmdName, frage_ObjektWertString)=0) or
    (lStrCmpI(CmdName, frage_ObjektWertNumber)=0) or
    (lStrCmpI(CmdName, frage_ObjektWertDate)=0) then
    DdePokeData:=DdePokeVal(CmdName, szParameter, aDataBf, {anAttach, }aPosition);
  if (lStrCmpI(CmdName, frage_BeziehungWert)=0) then
    DdePokeData:=DdePokeRel(CmdName, szParameter, aDataBf, {anAttach, }aPosition);
  if (lStrCmpI(CmdName, frage_Eingabefilter)=0) then
    DdePokeData:=DdePokeInpAttr(CmdName, aDataBf);
  if (lStrCmpI(CmdName, frage_Ausgabefilter)=0) then
    DdePokeData:=DdePokeSrcAttr(CmdName, aDataBf);
  ShowStatus(nil, nil, false);
end;

var
  HszSysTopic,
  HszServName,
  HszDefConvTopic,
  HszFileConvTopic: HSZ;
  CallBack: tCallback;
```

function DdeCallback(CallType, Fmt: Word; Conv: HConv; hsz1, hsz2: HSZ; Data: HDDEData; Data1, Data2: Longint): HDDEData; export;

```
var
  aLine, Hsz1pChar, Hsz2pChar: tLine;
  aDataBf: pChar;
  aDataBfSz: longInt;
  aHandle: tHandle;

begin
  DDECallBack:=0;
  DdeQueryString(idInst, Hsz1, Hsz1pChar, SizeOf(tLine)-1, cp_winansi);
  DdeQueryString(idInst, Hsz2, Hsz2pChar, SizeOf(tLine)-1, cp_winansi);
  case CallType of
    xtyp_connect:
      begin
        if MatchTopic(hsz1pchar) then
          DDECallBack:=1
        else
          DDECallBack:=0;
      end;
    xtyp_connect_confirm:
      begin
        TheMainWnd^.AddConvToLst(Conv);
        DDECallBack:=0;
      end;
    xtyp_disconnect:
      begin
        TheMainWnd^.DeleteConvFromLst(Conv);
        DDECallBack:=0;
      end;
    xtyp_request:
      begin
        DdeCallBack:=0;
        if not DdeAnswerRequest(Hsz2pChar, aLine, SizeOf(tLine)-1) then exit;
        DdeCallBack:=DdeCreateDataHandle(idInst, @aLine, StrLen(aLine)+1, 0, Hsz2, cf_text, 0);
      end;
    xtyp_poke:
      begin
        DdeCallBack:=0;
        aDataBfSz:=DdeGetData(Data, nil, 0, 0)+1;
        aHandle:=LocalAlloc(lmem_moveable, aDataBfSz); if aHandle=0 then exit;
        aDataBf:=LocalLock(aHandle); if aDataBf=nil then begin LocalFree(aHandle); exit; end;
        DdeGetData(Data, aDataBf, SizeOf(tLine)-1, 0);
        DdeFreeDataHandle(Data);
        if DdePokeData(Hsz2pChar, aDataBf)
          then  DDECallBack:=DDE_FACK else DDECallBack:=DDE_FNotProcessed;
        LocalFree(aHandle);
```

```
    end;
  xtyp_execute:
    begin
      DdeCallBack:=0;
      aDataBfSz:=DdeGetData(Data, nil, 0, 0)+1;
      aHandle:=LocalAlloc(lmem_moveable, aDataBfSz); if aHandle=0 then exit;
      aDataBf:=LocalLock(aHandle); if aDataBf=nil then begin LocalFree(aHandle); exit; end;
      DdeGetData(Data, aDataBf, SizeOf(tLine)-1, 0);
      DdeFreeDataHandle(Data);
      if ExecuteCommand(aDataBf) then  DDECallBack:=DDE_FACK;
      LocalFree(aHandle);
    end;
  xtyp_advreq:
    begin
      ddeCallBack:=0;
    end;
  xtyp_advdata:
    begin
      if true then
        DDECallBack:=DDE_FACK
      else
        DDECallBack:=DDE_FNotProcessed;
    end;
  xtyp_advstart:
    begin
      if true then
        DDECallBack:=1
      else
        DDECallBack:=0;
    end;
  xtyp_advStop:
    begin
      ddeCallBack:=0;
    end;
 end;
 ShowStatus(nil, nil, false);
end;

{----------------------------Dialoge-----------------------------}

{$I reldlg.pas}[427]

{$I repdlg.pas}[428]

{----------------------------TMainWnd----------------------------}

{$I atlmainw.pas}[429]


constructor TApp.Init(AName: PChar);

begin
 TApplication.Init(AName);
end;


destructor TApp.Done;

begin
 RewriteAtlFncIni; {nur für Entwicklung sinnvoll}
 TApplication.Done;
end;
```

[427] S. 297 ff

[428] S. 299 ff

[429] S. 307 ff

```
procedure TApp.InitMainWindow;

begin
  MainWindow := New(PMainWnd, init(nil, nil));
  TheMainWnd:=pMainWnd(MainWindow);
end;


procedure TApp.InitInstance;

begin
  if hPrevInst>0 then
  begin
    XMessageBox(0, 3, mb_ok or mb_systemmodal);
    PostQuitMessage(0);
    exit;
  end;
  TApplication.Initinstance;
end;

begin
  App.Init('Program');
  App.Run;
  App.Done;
end.
```

ATL2FPE.PAS

```
{OBJEKT zur Auswertung beliebiger Instanzen innerhalb von Ausdrücken}

{Letze Änderung : 20.4.93}

const
  InstCount : word = 0;

{Die eigentliche Funktion folgt etwas weiter unten.}

{Die Funktion wird in LesPos eingesetzt und wurde deshalb forward deklariert}


function ScParObj(lpParLine : pChar; wParIx : word; bCreateNew : boolean; var hRelation : LongInt;
var hBeziehung : longint; var wTreffer : word) : LongInt;

forward;


function LesPos(lpParLine : pChar; var Position : tPosition) : boolean;

{Funktion liest aus einer Parameterzeile an beliebiger Position

Angaben über die Position aus. Die Eingabewerte werden NICHT durch Dummys

überschrieben.

Die Funktion ist falsch, falls ein Syntaxfehler auftrat.}

var
  wX : word;
  aPChar : pChar;

  szFrage       {Enthält das aktuelle Kommando}
    : tLine;
  lpParams      {Zeigt auf die aktuell bearbeitete Parameterliste}
    : PChar;
  lObjId        {Enthält die ID einer Entität nächster Instanz}
    : LongInt;
  lParCount     {Enthält die Anzahl der Parameter einer Zeile}
    : word;
  bParOK        {Parameter korrekt ausgewertet}
    : boolean;
```

function ReadPosition : boolean;

```
{Index, Sort}

var

{  aParam : pChar;}
  aLine : tLine;
  y, m, d : word;
  aReal : real;

begin

{!!!Die Möglichkeit, zur Positionsangabe noch eine Sortierung anzugeben,

fehlt derzeit. Hiermit könnte man Probleme lösen wie:

"Der erste nach Reihenfolge des Geburtsdatums" etc.

Derzeit wird lediglich nach Eingabereihenfolge sortiert!!!}
  ReadPosition := false;
  if not GetParam(lpParams, aLine, 1, SizeOf(tLine)) then exit;
  if aLine[0]<>#0 then
  begin
    if not StrToNum(aLine, aReal) then exit;
    Position.PosIndex := trunc(aReal);
  end;
  ReadPosition := true;
end;

begin  {ReadAllStdAttr}
  {with RPStack^ do
  begin}
    LesPos := false;
    lParCount := GetParamCount(lpParLine);
    for wX := 1 to lParCount do
    begin
      GetParam(lpParLine, szFrage, wX, SizeOf(tLine));
      bParOK := false;
      if szFrage[0] = prefix_Funktion then
      begin
        SplitCmdLine(szFrage, lpParams);
        aPChar := szFrage+1;
        if lStrCmpI(aPChar, frage_Position) = 0 then
          begin bParOK := true; if not ReadPosition then exit; end;
      end;
    end;
    LesPos := true;
  {end;}
end;
```

function ScParObj(

```
  lpParLine : pChar;            {Enthält die Parameterzeile}
  wParIx : word;                {Gibt den gesuchten Parameter an}
  bCreateNew : boolean;         {Gibt an, ob das Objekt erzeugt werden soll}
  var hRelation : LongInt;      {nimmt ein Handle der zugrundeliegenden
                                 Relation auf.}
  var hBeziehung : longint;     {nimmt ein Handle der Beziehung auf,
                                 aus der der Parameter stammt}
  var wTreffer : word) : LongInt;  {nimmt die Anzahl der möglichen Parameter auf}

{Die Funktion liest den wParIx-ten Parametern aus lpParLine.

Sie gibt ein Handle eines Objektes zurück.}

{Wenn bCreateNew gesetzt ist, wird eine fehlende Entität

automatisch erzeugt und auf Exist gesetzt!}

{Line : tLine; allStdAttr : record SA : tStdAttr; PO : word; SO : longint; end;}

type
  tBezURes = record
```

```
{   hBez: longint;}
    Beziehung: tBeziehung;
    iRes: integer;
  end;
  tBezBf = array[1..1] of tBezURes;

var
  wX, wY : Word;          {Laufvariable}
  wObjTreffer : Word;     {Trefferzahl}
  wObjAnz : Word;

  aPChar : pChar;
  lpParams : pChar;

  aLongInt : LongInt;
  hRelBf,
  hBezBf,
  hTreffer                {ID des Treffers}
    : LongInt;

  wAnzTreffer,            {Zähler der wTreffer}
  wParamIX,
  aLen,                   {Längeneinheit für Parameter einer Zeile}
  wAnzBez,                {Anzahl der Verbindungen des beschreibenden Terms}
  wAnzParams,             {Anzahl der Attribute des beschreibenden Terms}
  icErr, icPrErr          {Anzahl der nicht ermittelten Objekte}
    : word;

  Hit : Integer;          {Trefferzähler bei negativer Positionsangabe = Zurückzählen}

  bezTreffer,             {Beschreibung des Treffers}
  bezRef2,                {Vergleichsbeziehung}
  bezRef1                 {Beschreibung der angeforderten Beziehung}
    : tBeziehung;

  bezListe : ^tBezBf;     {Puffer zur Aufnahme der Beziehungen eines Terms}

  ObjektNeu,              {Anzulegende neue Entität}
  einObjekt               {Puffer für zu lesende Entitäten}
    : tObjekt;

  bCreate : boolean;

  szMsg : tLine;
  szParam : tLine;        {Puffer für einzelne Parameter}

  relxBf,                 {Term zur Beschreibung der Entität}
  relxAttr: tXRelation;   {Attribute einer Relation}
  hAttr: tRelParam;
  Position : tPosition;

const
  szPr1 : pChar = 'ScParObj %i';
  szPr2 : pChar = 'BEZ %i';

begin
  {with FpeStack^ do
  begin}

  {Initialisieren aller Variablen}
  wX := 0;
  wY := 0;
  wObjTreffer := 0;
  wObjAnz := 0;

  aLongInt := 0;
  wAnzTreffer := 0;
  hTreffer := 0;
  aLen := 0;
  Hit := 0;
  icErr := 0;
  icPrErr := 0;
  bCreate := false;
  wParamIX := 0;
  wAnzBez := 0;
  wAnzParams := 0;
  hRelBf := 0;
  hBezBf := 0;
```

```
  szMsg[0] := #0;
  szParam[0] := #0;

{  bezTreffer : tBeziehung;
  bezRef2 : tBeziehung;
  bezRef1 : tBeziehung;
  bezListe : ^tBezBf;

  ObjektNeu : tObjekt;
  einObjekt : tObjekt;
  relxBf : tXRelation;
  relxAttr: tXRelation;
  hAttr: tRelParam;}
  Position := DummyPosition;

{messagebox(0, lpParLine,'Hey',0);}

    {Defaultwert für die Rückgabe}
    ScParObj := -1;

    {Defaultwert für das Beziehungshandle}
    hRelation := -1;
    hBeziehung := -1;

    {Defaultwert für die Position}
    Position := DummyPosition;

    {Objekt-Kopien der Parameter Anlegen}
    lpParams := lpParLine;

    {Prüfen, ob die Parameterzeile leer ist.}
    if StrLen(lpParams) < 1 then
    begin
      ScParObj := Obj_dummy;
      exit;
    end;

    {Index des gesuchten Paramters}
    wParamIX := wParIx;

    {Prüfen, ob der gesuchte Parameter(index) existiert.}
    if not GetParam(lpParams, szParam, wParamIX, SizeOf(tLine)) then exit;

    {Prüfen, ob der gesuchte Parameter leer ist.}
    if szParam[0] = #0 then
    begin
      ScParObj := Obj_dummy;
      exit;
    end;

    {Objekt herstellen, falls es fehlt.}
    bCreate := bCreateNew;

{MessageBox(0, szParam, 'Frage', 0);}

    {Prüfen, ob der Parameter mit einem Objektpräfix beginnt.}
    if szParam[0] = prefix_Objekt then
    begin
      {Bei benannten Parametern gibt es genau einen wTreffer}
      wTreffer := 1;

      {Objekt wird durch den Bezeichner benannt}
      aPChar := @szParam[1];
      ClipQuotes(aPChar);

      {Suchen des Objektindexes mit der Funktion ObjektIndexLesen}
      aLongInt := ObjektIndexLesen(aPChar);

      {Prüfen, ob der Bezeichner bereits bekannt war.}
      if aLongInt > 0 then
      begin
        {Bezeichner war bekannt}
        ScParObj := aLongInt
      end
      else begin {aLongInt>0}
        if bCreate then
        begin
          {Bezeichner war nicht bekannt, soll aber erstellt werden}
          {Leeres Objekt erstellen}
          ObjektNeu := ObjLeer;
```

```
        with ObjektNeu do
        begin
          {Bezeichner vergeben}
          StrLCopy(SzObjBez, aPChar, SizeOf(tLine)-1);

          {Objekt ist dezidiert = nicht fiktiv}
          LObjFlags := Obj_Real;
        end; {with ObjektNeu do}

        ScParObj := ObjektZufuegen(ObjektNeu);
      end {bCreate} else
      begin
        {Falls der Bezeichner unbekannt war, wird -1 zurückgegeben}
        ScParObj := -1;
      end;

    end; {aLongInt>0}

    {Die folgende Zeile nur zur Erinnerung wiederholt}
    hRelation := -1;
    hBeziehung := -1;

    {Jedenfalls verlassen wir die Funktion hier vorzeitig}
    exit;
  end;  {Ende Entität direkt}

  {Die Systemeigenschaft @Objekt gilt für alle Objekte}
  {Ein Objekt kann durch @Objekt in Verbindung mit @Pos bestimmt werden}
  {@Anzahl(@Objekt) ergibt die Anzahl aller Objekte}

  if (szParam[0] = prefix_Funktion)
    and (StrLIComp(@szParam[1], frage_Objekt, StrLen(frage_Objekt)) = 0) then
  begin

{MessageBox(0, szParam, '@Objekt', 0);}
    SplitCmdLine(szParam, aPChar);

    {Speichern der Position}
    {ocPosition := Position;}
    {Ermitteln eines dedizierten Positionsparameters @Pos, falls vorhanden}
    LesPos(aPChar, Position);

    {Ermitteln der Anzahl der Objekte}
    {Sie entspricht der Anzahl der Beziehungen mit @Objekt als Relation}
    wAnzBez := ObjekteAnzahl;

    {Nun beginnt die Suche in Abhängigkeit vom Positionsparameter.}
    if Abs(Position.PosIndex) > 0 then
    begin
      {Mit dem Positionsparameter wurde die Position ausdrücklich angegeben}

      {Prüfen, ob der Positionsparameter von zu vielen Objekten ausgeht}
      if Abs(Position.PosIndex) > wAnzBez then
      begin
        {Es gibt weniger Objekte als der Positionsparameter erwartet}
        {Die Funktion tut nun nichts}

        {Die Anzahl der passenden wTreffer ist bei fehlerhafter Positionsangabe 0}
        wTreffer := 0;
      end else {Abs(ocPosition.PosIndex) > ^wAnzTreffer}
      begin
        {Die gewünschte Position liegt also unter der Trefferzahl}
        {Nun ist zu differenzieren, ob abwärts oder aufwärts zu zählen ist.}
        {Mit Hit zählen wir die Anzahl der wTreffer}
        Hit := 0;

        {Wir Zählen aufwärts, wenn der PosIndex > 0 ist}
        if Position.PosIndex>0 then
        begin
          hTreffer := Position.PosIndex;
        end else
        begin
          {Man bedenke PosIndex ist negativ, deshalb Addition!}
          hTreffer := wAnzBez +Position.PosIndex +1;
        end;

        {Die Anzahl der passenden wTreffer ist bei Positionsangabe 1}
        wTreffer := 1;
      end;
    end else {Abs(ocPosition.PosIndex) > 0}
```

```
    begin
     {Der Positionsbezeichner ist 0}
     {Wenn überhaupt ein wTreffer existiert,
     wird der letzte gültige Wert gewählt}

     wTreffer := wAnzBez;
     hTreffer := wAnzBez
    end; {Abs(ocPosition.PosIndex) > 0}

    if hTreffer > 0 then
    begin
     {Es existiert ein wTreffer}
      {Das Ergebnis ist die Hauptentität des Treffers}
     ScParObj := hTreffer;
    end;

    {Die folgende Zeile nur zur Erinnerung wiederholt}
    hRelation := -1;
    hBeziehung := -1;

    exit;
   end; {@Objekt}

  {Der Zeiger kommt nur hier an, wenn
  weder ein Objektbezeichner ($) vorliegt
  noch das Objekt mit @Objekt gesucht wurde}
  begin
   {Nur damit wir's nicht vergessen!}
   ScParObj := -1;

{MessageBox(0, szParam, 'Indirekt', 0);}
    {Das Objekt muß nun aus einem Ausdruck ermittelt werden}
    {Der Parameter wird in Relation und neue Parameterzeile zerlegt}
    SplitCmdLine(szParam, aPChar);

    {Prüfen, ob eine Relation vorhanden ist.}
    if StrLen(szParam)<1 then  exit;

    {Ermitteln des Index der Relation}
    hRelation := RelationIndexLesen(szParam);

    {Prüfen, ob der Index gefunden wurde.}
    if hRelation < 1 then exit;

    {Lesen der Daten der Relation}
    if not RelationLesen(hRelation, relxBf) then exit;

    {Speichern der Position}
    {ocPosition := Position;}
    {Ermitteln eines dedizierten Positionsparameters @Pos, falls vorhanden}
    LesPos(aPChar, Position);

    {Ermitteln der Anzahl der Parameter der Relation}
    wAnzParams := RelationParamAnzahl(relxBf.relXRelation);

    {Um zu ermitteln, welche Beziehungen auf den angegebenen Ausdruck passen,
    wird der Ausdruck in eine Referenzbeziehung verwandelt.
    Hierdurch entsteht der rekursive Aufruf von ScParObj!!!}

    {***-------------------------------***}
    {*** Beginn des rekursiven Bereichs ***}
    {***-------------------------------***}

    {Hier können einige Objekte Dummy-Objekte sein.
    Die folgende Passage stellt diese Referenzbeziehung zusammen.}
    with bezRef1 do
    begin
     {Eingabefilter abgeleichen}
     FltBezWert := fltAktEingFlt;

     {Alle Objekte erst auf falsch stellen}
     {icPrErr zählt hier die Anzahl der Objekte}
     {ansich enthält er die letzte Anzahl der negativen Rückmeldungen von SPosObj}
     icPrErr := 1;

     {Hauptobjekt}
     LBezHauptObj := -1;

     {weitere Objekte}
     if wAnzParams>0 then for wX := 1 to wAnzParams do
```

```
  begin
    LbfBezObj[wX] := -1;
    icPrErr := icPrErr +1;
  end; {if wAnzParams>0 then for wX := 1 to wAnzParams do}

  icErr := 0;

  {Wir probieren soviele Durchläufe,
  bis sich an der Fehlerzahl nichts mehr ändert.}
  {Beim Einstieg entspricht die vorangegangene Fehlerzahl
  der Anzahl der Objekte selbst}
  while icErr<>icPrErr do
  begin
    icPrErr := icErr;
    icErr := 0;

    {Ermitteln des Hauptobjektes}
    if LBezHauptObj<0 then
    begin
      LBezHauptObj := ScParObj(aPChar, 1, bCreate, hRelBf, hBezBf, wObjTreffer);
      {Prüfen, ob ein Fehler aufgetreten ist}
      if LBezHauptObj<0 then icErr := icErr +1;
    end;

    {Ermitteln der weiteren Parameter}
    if wAnzParams>0 then for wX := 1 to wAnzParams do
    begin
      {Einlesen des Parameters}
      {Der Positionsparameter muß 0 sein}
      if LbfBezObj[wX]<0 then
      begin
        LbfBezObj[wX] := ScParObj(aPChar, wX+1, bCreate, hRelBf, hBezBf, wObjTreffer);
        {Prüfen, ob ein Fehler aufgetreten ist}
        if LbfBezObj[wX]<0 then icErr := icErr +1;
      end;

    end; {if wAnzParams>0 then for wX := 1 to wAnzParams do}
  end; {while icErr<>icPrErr do}

  {Wenn sich Fehler nicht beheben ließen, steigen wir hier aus}
  if icErr > 0 then
  begin
    ScParObj := -1;
    exit;
  end;
end; {with bezRef1 do}

{***-------------------------------***}
{***  Ende des rekursiven Bereichs  ***}
{***-------------------------------***}

{Initialisieren der wesentlichen Variablen
Rücksetzen des Default-Wertes
ScParObj := -1;
Initialisieren des Trefferzählers
^wAnzTreffer := 0;
Initialisieren des Trefferbezeichners
hTreffer := 0;}

{Steht die Referenzbeziehung, so werden alle aktuellen Beziehungen,
die auf der Relation beruhen, danach überprüft, ob sie mit der Referenz
übereinstimmen. Dummies stimmen dabei immer überein!}

{Ermitteln der Anzahl der abgelegten Beziehungen der Relation}
wAnzBez := BeziehungenAnzahl(relxBf);

{Vergleichen der einzelnen Beziehungen der Relation
mit der Referenzbeziehung}
if wAnzBez>0 then
begin
  {Reservieren des Speicherplatzes für alle Relationen}
  GetMem(bezListe, SizeOf(tBezURes) * wAnzBez);

  {Prüfen, ob Speicherreservierung erfolgreich}
  if bezListe = nil then exit;

  {Einlesen der Beziehungen}
  for wX := 1 to wAnzBez do
  begin
```

```
        {Auslesen der Beziehung}
        BeziehungLesen(wX, bezListe^[wX].Beziehung, relxBf);

        {Zunächst wird überprüft, ob die Beziehung ungeachtet des Wertes
        mit der Referenzbeziehung übereinstimmt.}
        bezRef1.XbBezWert := xBoDmy;
        if VergleicheBeziehungen(bezListe^[wX].Beziehung, bezRef1, wAnzParams) then
        begin
          {Lockerer Vergleich fällt positiv aus}

          {Nun wird überprüft, ob auch die Beziehungswerte übereinstimmen.}
          bezRef1.XbBezWert := xTrue;
          if VergleicheBeziehungen(bezListe^[wX].Beziehung, bezRef1, wAnzParams) then
          begin
            {Strenger Vergleich fällt positiv aus}

            {Stimmen Beziehung und Wert überein, so handelt es sich um eine
            passende Beziehung, die in die Liste eingetragen wird.}
            bezListe^[wX].iRes := 1;
          end {VergleicheBeziehungen(bezListe^[wX], bezRef1, wAnzParams)}

          else begin
            {Strenger Vergleich fällt negativ aus}

            {Stimmen zwar die Objekte, nicht aber der Wert der Beziehung überein,
            so werden alle vorangegangenen Werte entsprechender
            Beziehung negativ gesetzt}
            {Dabei wird auch der Vergleichswert selbst negativ gesetzt}
            {Generieren einer weiteren lockeren Referenzbeziehung}
            bezRef2 := bezListe^[wX].Beziehung;
            bezRef2.XbBezWert := xBoDmy;

            for wY := 1 to wX do
            begin
              if VergleicheBeziehungen(bezListe^[wY].Beziehung, bezRef2, wAnzParams) then
              begin
                bezListe^[wX].iRes := -1;
              end; {VergleicheBeziehungen(bezListe^[wY], bezRef2, wAnzParams)}
            end; {for wY := 1 to wX do}

          end; {VergleicheBeziehungen(bezListe^[wX], bezRef1, wAnzParams)}

        end {VergleicheBeziehungen(bezListe^[wX], bezRef1, wAnzParams)}
        else begin
          {Lockerer Vergleich fällt negativ aus}

{MessageBox(0, 'Lockerer Vergleich fällt negativ aus', szMsg, 0);}

          {Der Vergleichswert wird indifferent gesetzt}
          bezListe^[wX].iRes := 0;
        end; {VergleicheBeziehungen(bezListe^[wX], bezRef1, wAnzParams)}

      end; {for wX := 1 to wAnzBez do}

      {Nachdem wir nun sicher alle relevanten Beziehungen kennen,
      können wir uns auf die Suche nach der gefragten Beziehung machen}

      {Zunächst einmal zählen wir die relevanten wTreffer durch.}
      {Initialisieren des Trefferzählers}
      wAnzTreffer := 0;
      {Initialisieren des Trefferbezeichners}
      hTreffer := 0;

      {Hochzählen der wTreffer}
      for wX := 1 to wAnzBez do
      begin
        if bezListe^[wX].iRes = 1 then wAnzTreffer := wAnzTreffer +1;
      end;

      {Nun beginnt die Suche in Abhängigkeit vom Positionsparameter.}
      if Abs(Position.PosIndex) > 0 then
      begin
        {Mit dem Positionsparameter wurde die Position ausdrücklich angegeben}

        {Prüfen, ob der Positionsparameter von zu vielen Treffern ausgeht}
        if Abs(Position.PosIndex) > wAnzTreffer then
        begin
          {Es weniger wTreffer als der Positionsparameter erwartet}
          {Die Funktion tut nun nichts}
          {Sie hilft nicht einmal aus, wenn bCreateNew wahr ist}
```

```
    {Die Anzahl der Passenden wTreffer ist bei fehlerhafter Positionsangabe 0}
    wTreffer := 0;
   end else {Abs(ocPosition.PosIndex) > ^wAnzTreffer}
   begin
    {Die gewünschte Position liegt also unter der Trefferzahl}
    {Nun ist zu differenzieren, ob abwärts oder aufwärts zu zählen ist.}
    {Mit Hit zählen wir die Anzahl der wTreffer}
    Hit := 0;

    {Wir zählen aufwärts, wenn der PosIndex > 0 ist}
    if Position.PosIndex>0 then
    begin
     {Aufwärts zählen}
     for wX := 1 to wAnzBez do
     begin
      if bezListe^[wX].iRes = 1 then
      begin
       Hit := Hit +1;
       if Hit = Position.PosIndex then hTreffer := wX;
      end; {aResBf^[wX] = 1}
     end; {for wX := 1 to wAnzBez}

    end else begin {ocPosition.PosIndex>0}
     {Abwärts zählen, Positionsbezeichner ist negativ}
     for wX :=  wAnzBez DownTo 1 do
     begin
      if bezListe^[wX].iRes = 1 then
      begin
       Hit := Hit -1;
       if Hit = Position.PosIndex then hTreffer := wX;
      end; {aResBf^[wX] = 1}
     end; {for wX :=  wAnzBez DownTo 1}
    end; {ocPosition.PosIndex>0}

    {Die Anzahl der passenden wTreffer ist bei Positionsangabe 1}
    wTreffer := 1;
   end; {Abs(ocPosition.PosIndex) > ^wAnzTreffer}
  end
  else {Abs(ocPosition.PosIndex) > 0}
  begin
   {Der Positionsbezeichner ist 0}
   {Wenn überhaupt ein wTreffer existiert,
   wird der letzte gültige Wert gewählt}

   Hit := 0;
   if wAnzTreffer > 0 then
   begin
    {Abwärts zählen}
    for wX :=  wAnzBez DownTo 1 do
    begin
     if bezListe^[wX].iRes = 1 then
     begin
      Hit := Hit -1;
      if Hit = -1 then hTreffer := wX;
     end; {aResBf^[wX] = 1}
    end; {for wX :=  wAnzBez DownTo 1}
   end; {^wAnzTreffer > 0}

   {Die Anzahl der passenden wTreffer ist ^wAnzTreffer}
   wTreffer := wAnzTreffer;
  end; {Abs(ocPosition.PosIndex) > 0}

  {Bevor der Speicher leergeräumt wird,
  wird der wTreffer ausgelesen}
  if hTreffer>0 then
  begin
   {Es existiert ein wTreffer}
   {Das Ergebnis ist die Hauptentität des Treffers}
   ScParObj := bezListe^[hTreffer].Beziehung.LBezHauptObj;
   bezTreffer := bezListe^[hTreffer].Beziehung;
  end;

  {Nun wird der Speicher leergeräumt}
  FreeMem(bezListe, SizeOf(tBezURes)*wAnzBez);

 end; {wAnzBez > 0}
```

```
{Wenn wir einen wTreffer haben, sind wir hier fertig!}
hBeziehung := hTreffer;
if hTreffer > 0 then exit;

{Unabhängig von der Zahl der bisher existierenden Beziehungen
wird nun überprüft, ob ein wTreffer gefunden wurde}

{Wurde kein wTreffer gefunden, ist zudem bCreate wahr
und gibt es keine Positionsangabe,
dann wird die geforderte Beziehung und
falls nötig ein passendes Objekt creiert}

if (bCreate) and (Position.PosIndex = 0) then
begin
 {Wenn kein Hauptobjekt angegeben wurde, so muß eines instantiiert werden}
 if bezRef1.LBezHauptObj <= 0 then
 begin
  {Es existiert in der Beziehung kein Hauptobjekt}

  {Es wird ein leeres Objekt erzeugt}
  ObjektNeu := ObjLeer;

  with ObjektNeu do
  begin
   {Das Objekt ist virtuell, es war kein Bezeichner angegeben}
   LObjFlags := Obj_virtual;

   {Der Objektbezeichner wird synthetisiert
   aus dem Relationsbezeichner und den Objektbezeichner der Attribute}

   {Zunächst wird die max Länge der Einzelteile bestimmt}
   aLen := (SizeOf(tLine) - wAnzParams) div ((wAnzParams + 1) * 2);

   {Hier wird der Relationsbezeichner für den Objektbezeichner eingesetzt}
   StrLCopy(SzObjBez, relxBf.RelXRelation.SzRelBez, aLen * (wAnzParams + 1));

   if wAnzParams > 0 then
   begin
    {Nun werden die Objektbezeichner der Parameter an den Bezeichner des
    Hauptobjektes angehängt}
    StrCat(SzObjBez, ' '#0);

    for wX := 1 to wAnzParams do
    begin
     einObjekt := ObjLeer;

     {Einlesen des Objektes}
     if (bezRef1.LbfBezObj[wX] > 0) then
     begin
      ObjektLesen(bezRef1.LbfBezObj[wX], einObjekt)
     end
     else
     begin

      {Der anschließend beschriebene Versuch,
      auch Attributsobjekte zu erzeugen,
      wird erst einmal aufgegeben!}

      exit;

      {Existiert das Attribut-Objekt nicht,
      wird ein passendes Objekt erstellt}

      {Es erhält einen synthetischen Namen aus Rollenbezeichner
      und Anzahl der bisher instantiierten Objekte}

      {Wir ermitteln hier den Bezeichner des Parameters}
      RelationParamLesen(wx, hAttr, relxBf.relxRelation);
      RelationLesen(hAttr, relxAttr);

      {Nun erstellen wir ein leeres Objekt}
      einObjekt := ObjLeer;

      {Der Objektbezeichner wird der Parameterbezeichner...}
      strcopy(einObjekt.SzObjBez, relxAttr.relxRelation.szRelBez);
      {...erweitert um die Anzahl der Objekte +1}
      strcat(einObjekt.SzObjBez, '/%i');
      wObjAnz := ObjekteAnzahl + 1;
      wvsprintf(einObjekt.SzObjBez, einObjekt.SzObjBez, wObjAnz);
```

```
            {Das Objekt ist virtuell}
            einObjekt.LObjFlags := Obj_virtual;

            {Nun erstellen wir das Objekt}
            bezRef1.LbfBezObj[wX] := ObjektZufuegen(einObjekt);
          end;

          {Anfügen des Objektbezeichners}
          StrLCat(SzObjBez, einObjekt.SzObjBez, StrLen(SzObjBez)+aLen);

          {zwischen den Objektbezeichnern wird ein Bindestrich gesetzt}
          if wX < wAnzParams then StrCat(SzObjBez, '-'#0);
        end; {for wX := 1 to wAnzParams do}
      end; {wAnzParams>0}
    end; {with ObjektNeu do}

    {Anlegen des Objektes}
    bezRef1.LBezHauptObj := ObjektZufuegen(ObjektNeu);
  end; {bezRef1.LBezHauptObj<1}

  {Nur wenn das Hauptobjekt existiert, wird die Relation erzeugt}
  if bezRef1.LBezHauptObj > 0 then
  begin
    bezRef1.XbBezWert := xTrue;

    {in hBeziehung wird das Handle der neuen Beziehung eingetragen}
    hBeziehung := BeziehungZufuegen(bezRef1, relxBf);

    {Die Relation muß noch neu gesetzt werden}
    RelationSchreiben(hRelation, relxBf);

    {Das gesuchte Objekt ist das Hauptobjekt der Relation}
    ScParObj := bezRef1.LBezHauptObj
  end; {bezRef1.LBezHauptObj > 0}
  wTreffer := 1;
 end else
 begin
  {hBeziehung wird auf Fehler gesetzt}
  {hRelation wird belassen!
  Eine Funktion kann diesen Wert später verwenden}
  hBeziehung := -1;
 end; {hTreffer = 0 and bCreate and ocPosition.PosIndex = 0}

  end; {szParam[0] = prefix_Objekt : Entität war durch Relation beschrieben}
 {end;} {with FpeStack^ do}
end; {function MainFnc : LongInt}
```

function SucheParamObjekt(lpParLine : pChar; wParIx : word;

```
{  StdAttr : tStdAttr;}
  bCreateNew : boolean;
  var hRelation : longint;
  var hBeziehung : longint;
  var wTreffer : word) : LongInt;

{Funktion wertet die Suchanfrage nach einer Entität aus und gibt eine ID zurück}

{Wenn EtyCreate gesetzt ist, dann wird eine fehlende Entität automatisch erzeugt!}

begin
  SucheParamObjekt := ScParObj(lpParLine, wParIx, bCreateNew, hRelation, hBeziehung , wTreffer);
end;
```

function ReadPosition(lpParLine : pChar; **var** Position : tPosition) : boolean;

```
begin
  ReadPosition := LesPos(lpParLine, Position);
end;
```

ATLCALL.PAS

```
{----------------------------CallBack für Filter-----------------------------}
```

```
const
  {Funktionsaufrufe, die von der CallBack-Funktion der DLL bedient werden
    L1 = lParam1
    L2 = lParam2
    W = wParam
    R = Rückgabewert}

{Initialisierung}
  fct_init = $0001;
    {Initialisiert die DLL und gibt den Namen zurück
      L1: tAtlCallBack (r);
      L2: Handle des Atlas-Fensters
      R: Flags
        LoWord: bezeichnet unterstützte fct_xxx Befehle, 0=Fehler
        HiWord: noch zu bestimmen}
  fct_GetDllName = $0002;
    {Aufforderung, den Namen der DLL für einen Listeneintrag zurückzugeben
      W: BfSize;
      L1: pChar (w);
      R: 0/1}
  fct_GetFncName = $0004;
    {Aufforderung, den Namen einer Funktion zu übermitteln
    oder zu Prüfen, ob eine Funktion unterstützt wird
      W: BfSize, wenn ID=0 unbeachtet;
      ID: Index der Funktion, Wenn 0 dann wird L1 Überprüft
      L1: pChar (w);
      R:
        ID=0: Index der Funktion in L1 oder 0
        ID>0: Flags <>0 (noch zu bestimmen) oder 0}

{Laufzeit Meldungen}
  fct_GetMainEty = $0008;
    {Aufforderung, die MainEty des Records zu finden und dort einzutragen
      L1: pBeziehung (rw);
      R: 0/1}
  fct_CountMainEty = $0010;
    {Aufforderung, die die Anzahl der MainEty zu zählen, die auf die Beschreibung passen
      L1: pBeziehung (r);
      R: Anzahl}
  fct_GetVal = $0020;
    {Aufforderung, den Wert einer definierten Ety durch Auffüllen des EtyVal-Records zurückzugeben.
      ID: ID der Entität;
      L1: pObjektWert;
      R: 0/1}
  fct_GetValFromCon = $0040;
    {Aufforderung, den Wert einer gesuchten Ety durch Auffüllen des EtyVal-Records zurückzugeben.
      L1: pObjektWert;
      L2: pBeziehung;
      R: 0/1}
  fct_ExecFnc = $0080;
    {Aufforderung, eine angemeldete Funktion der DLL auszuführen
      W: Größe des Ausgabepuffers
      ID: Index des Eintrags in der Kommandozeile
      L1: pChar (Zeiger auf die Kommandozeile);
      L2: pChar Zeiger auf einen Puffer;
      R: 0/1}

type

{CallBack-Funktion für Filter

Die CallBack-Funktionen der Filter sind vom selben Typ}
  tAtlCallBack = function(
    CallType,
    wParam: word;
    ID,  {Da die Funktnion auf atl2mem Funktionen zugreift, liegt hier immer die ID}
    lParam1,
    lParam2: LongInt): LongInt;

type
  tFlt = record
    FltDll: tHandle;
    LAddOnFlags: longint;
    FltCallBack: tAtlCallBack;
  end;

var
  TheAtlCallBack: tAtlCallBack;
```

```
  TheFltLst: array[1..16] of tFlt; {Maximal 16 Filter!}
  FltLstSz: word;

const

{Diese Konstanten entsprechen den Funktionen der atl2mem.tpu}

{Trm-Funktionen}
  act_RelationLesen       = $0001;
  act_RelationenAnzahl  = $0002;
  act_RelationSchreiben       = $0003;
  act_RelationZufuegen       = $0004;
  act_RelationSuchen     = $0005;

{TrmAttr-Funktionen}
  act_RelationParamLesen      = $0011;
  act_RelationParamAnzahl  = $0012;
  act_RelationParamSchreiben       = $0013;
  act_RelationParamZufuegen       = $0014;

{TrmFlt-Funktionen}
  act_RelationAddOnLesen       = $0021;
  act_RelationAddOnsAnzahl  = $0022;
  act_RelationAddOnSchreiben       = $0023;
  act_RelationAddOnZufuegen       = $0024;

{Ety-Funktionen}
  act_ObjektLesen      = $0031;
  act_ObjekteAnzahl  = $0032;
  act_ObjektSchreiben       = $0033;
  act_ObjektZufuegen       = $0034;
  act_ObjektSuchen     = $0035;

{EtyVal-Funktionen}
  act_ObjektWertLesen       = $0041;
  act_ObjektWerteAnzahl  = $0042;
  act_ObjektWertSchreiben       = $0043;
  act_ObjektWertZufuegen       = $0044;

{EtyRel-Funktionen}
  act_BeziehungLesen       = $0051;
  act_BeziehungenAnzahl  = $0052;
  act_BeziehungSchreiben       = $0053;
  act_BeziehungZufuegen       = $0054;
```

function AtlCallBack(CallType, wParam: word; ID, lParam1, lParam2: LongInt): LongInt; export;

```
begin
  AtlCallBack:=0;
  case CallType of

{TRM-Funktionen:
  Param1 immer TRM  (Anders als in Origaínal-Funktionen!!)
  Param2 weitere Info}
    act_RelationLesen:
      AtlCallBack:=LongInt(RelationLesen(ID, tXRelation(Pointer(lParam1)^)));
    act_RelationenAnzahl:
      AtlCallBack:=RelationenAnzahl;
    act_RelationSchreiben:
      AtlCallBack:=LongInt(RelationSchreiben(ID, tXRelation(Pointer(lParam1)^)));
    act_RelationZufuegen:
      AtlCallBack:=RelationZufuegen(tXRelation(Pointer(lParam1)^));
    act_RelationSuchen:
      AtlCallBack:=RelationSuchen(PChar(lParam1), lParam2);
    act_RelationParamLesen:
      AtlCallBack:=LongInt(RelationParamLesen(ID, tRelParam(Pointer(lParam2)^), tRelation(Pointer(lParam1)^)));
    act_RelationParamAnzahl:
      AtlCallBack:=RelationParamAnzahl(tRelation(Pointer(lParam1)^));
    act_RelationParamSchreiben:
      AtlCallBack:=LongInt(RelationParamSchreiben(ID, tRelParam(Pointer(lParam2)^), tRelation(Pointer(lParam1)^)));
    act_RelationParamZufuegen:
      AtlCallBack:=RelationParamZufuegen(tRelParam(Pointer(lParam2)^), tRelation(Pointer(lParam1)^));
    act_RelationAddOnLesen:
      AtlCallBack:=LongInt(RelationAddOnLesen(ID, tAddOn(Pointer(lParam2)^), tRelation(Pointer(lParam1)^)));
    act_RelationAddOnsAnzahl:
      AtlCallBack:=RelationAddOnsAnzahl(tRelation(Pointer(lParam1)^));
```

```
    act_RelationAddOnSchreiben:
      AtlCallBack:=LongInt(RelationAddOnSchreiben(ID, tAddOn(Pointer(lParam2)^), tRelation(Pointer(lParam1)^)));
    act_RelationAddOnZufuegen:
      AtlCallBack:=RelationAddOnZufuegen(tAddOn(Pointer(lParam2)^), tRelation(Pointer(lParam1)^));
    act_BeziehungLesen:
      AtlCallBack:=LongInt(BeziehungLesen(ID, tBeziehung(Pointer(lParam2)^), tXRelation(Pointer(lParam1)^)));
    act_BeziehungenAnzahl:
      AtlCallBack:=BeziehungenAnzahl(tXRelation(Pointer(lParam1)^));
    act_BeziehungSchreiben:
      AtlCallBack:=LongInt(BeziehungSchreiben(ID, tBeziehung(Pointer(lParam2)^), tXRelation(Pointer(lParam1)^)));
    act_BeziehungZufuegen:
      AtlCallBack:=BeziehungZufuegen(tBeziehung(Pointer(lParam2)^), tXRelation(Pointer(lParam1)^));

{ETY-Funktionen:
  Param1 immer ETY  (Anders als in Origaínal-Funktionen!!)
  Param2 weitere Info}
    act_ObjektLesen:
      AtlCallBack:=LongInt(ObjektLesen(ID, tObjekt(Pointer(lParam1)^)));
    act_ObjekteAnzahl:
      AtlCallBack:=ObjekteAnzahl;
    act_ObjektSchreiben:
      AtlCallBack:=LongInt(ObjektSchreiben(ID, tObjekt(Pointer(lParam1)^)));
    act_ObjektZufuegen:
      AtlCallBack:=ObjektZufuegen(tObjekt(Pointer(lParam1)^));
    act_ObjektSuchen:
      AtlCallBack:=ObjektSuchen(PChar(lParam1), lParam2);
    act_ObjektWertLesen:
      AtlCallBack:=LongInt(ObjektWertLesen(ID, tObjektWert(Pointer(lParam2)^), tObjekt(Pointer(lParam1)^)));
    act_ObjektWerteAnzahl:
      AtlCallBack:=ObjektWerteAnzahl(tObjekt(Pointer(lParam1)^));
    act_ObjektWertSchreiben:
      AtlCallBack:=LongInt(ObjektWertSchreiben(ID, tObjektWert(Pointer(lParam2)^), tObjekt(Pointer(lParam1)^)));
    act_ObjektWertZufuegen:
      AtlCallBack:=ObjektWertZufuegen(tObjektWert(Pointer(lParam2)^), tObjekt(Pointer(lParam1)^));

{    act_SolveString: hier sollte ein Reststring nach Standardregeln gelöst werden!}
  end;
end;


function InitFilters(wnd: hWnd): boolean;

var
  aLine, iLine: tLine;
  aWord: word;
  aPChar: pChar;
  hDll: tHandle;
  aFnc: tFarProc;

begin
  @TheAtlCallBack:=MakeProcInstance(@AtlCallBack, hInstance);
  for aWord:=1 to 16 do
  begin
    wvsprintf(aLine,'Filter%i', aWord);
    GetPrivateProfileString(AtlIniSection, aLine,'', iLine, SizeOf(tLine), AtlIniFile);
    aPChar:=StrScan(iLine,';'); {Anmerkungen eliminieren}
    if aPChar<>nil then aPChar[0]:=#0;
    if (StrLen(iLine)>0) then
    begin
      hDll:=LoadLibrary(iLine);
      with TheFltLst[aWord] do
      begin
        if hDll>=32 then
        begin
          FltDll:=hDll;
          FltLstSz:=FltLstSz+1;
          aFnc:=GetProcAddress(hDll,'FltCallBack');
          @FltCallBack:=aFnc;
          LAddOnFlags:=FltCallBack(fct_init,0,0, LongInt(@TheAtlCallBack), wnd);
        end else
        begin
          FltDll:=0;
          @FltCallBack:=nil;
          LAddOnFlags:=0;
        end;
      end;
```

```
  end;
 end;
end;
```

function FreeFilters: boolean;

```
var
 aWord: word;

begin
 for aWord:=1 to 16 do
  with TheFltLst[aWord] do if FltDll>=32 then FreeLibrary(FltDll);
end;
```

RELDLG.PAS

```
{-----------------------------tRelBearbDlg-----------------------------}

type
 pRelBearbDlg = ^tRelBearbDlg;
 tRelBearbDlg = Object(tDialog)
  CurTrm: LongInt;
  constructor init(aParent: pWindowsObject; aTrmID: LongInt);
  procedure SetupWindow; virtual;
  procedure IdAddBn(var Msg: tMessage); virtual id_first + 108;
  procedure IdRemoveBn(var Msg: tMessage); virtual id_first + 109;
  procedure IdNewBn(var Msg: tMessage);  virtual id_first + 105;
  procedure DefWndProc(var Msg: tMessage); virtual;
  function CanClose: boolean; virtual;
 end;
```

constructor tRelBearbDlg.init(aParent: pWindowsObject; aTrmID: LongInt);

```
begin
 tDialog.init(aParent, 'EDITTERM');
 CurTrm:=aTrmId;
end;
```

procedure tRelBearbDlg.SetupWindow;

```
var
 anAttrTrm, aXTrm: tXRelation;
 anAttrID: LongInt;
 aLine: tLine;
 IX, aY, aTyp: word;

begin
 tDialog.SetupWindow;
 StrCopy(aLine, '[ohne]'); SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_addString, 0, LongInt(@aLine));
 SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_addString, 0, LongInt(ptty_text));
 SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_addString, 0, LongInt(ptty_integer));
 SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_addString, 0, LongInt(ptty_real));
 SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_addString, 0, LongInt(ptty_date));
 SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_addString, 0, LongInt(ptty_xbool));
 if CurTrm=0 then
 begin
  SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_SetCurSel, 5, 0);
  exit;
 end;
 RelationLesen(CurTrm, aXTrm);
 with aXTrm do
 begin
  IX:=SendDlgItemMessage(hWindow, 101, wm_SetText, 0, LongInt(@relXRelation.SzRelBez));
  if RelationParamAnzahl(relXRelation)>0 then for aY:=1 to RelationParamAnzahl(relXRelation) do
  begin
   RelationParamLesen(aY, anAttrId, relXRelation);
   RelationLesen(anAttrId, anAttrTrm);
   IX:=SendDlgItemMessage(hWindow, 107, cb_addString, 0, LongInt(@anAttrTrm.relXRelation.SzRelBez));
   SendDlgItemMessage(hWindow, 107, cb_SetItemData, IX, anAttrId);
  end;
```

```
  end;
  aTyp:=LoWord(axTrm.relXRelation.LRelFlags);
  SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_SetCurSel, aTyp, 0);
end;
```

procedure tRelBearbDlg.IdAddBn(**var** Msg: tMessage);

```
var
  IX: integer;
  aLine: tLine;

begin
  if SendDlgItemMessage(hWindow, 107, cb_GetCurSel, 0, 0) <> cb_Err then
     begin MessageBeep(0); exit; end;
  if SendDlgItemMessage(hWindow, 107, wm_GetText, SizeOf(tLine)-1, LongInt(@aLine))<=1 then
    begin MessageBeep(0); exit; end;
  IX:=SendDlgItemMessage(hWindow, 107, cb_AddString, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 107, cb_SetCurSel, IX, 0);
end;
```

procedure tRelBearbDlg.IdRemoveBn(**var** Msg: tMessage);

```
var
  IX: integer;
  aLine: tLine;

begin
  IX:=SendDlgItemMessage(hWindow, 107, cb_GetCurSel, 0, 0);
  if IX = cb_Err then  begin MessageBeep(0); exit; end;
  if SendDlgItemMessage(hWindow, 107, wm_GetText, SizeOf(tLine)-1, LongInt(@aLine))<=1 then
    begin MessageBeep(0); exit; end;
  SendDlgItemMessage(hWindow, 107, cb_DeleteString, IX, 0);
  SendDlgItemMessage(hWindow, 107, wm_settext, 0, LongInt(@aLine));
end;
```

procedure tRelBearbDlg.IdNewBn(**var** Msg: tMessage);

```
var
  aLine: tLine;

begin
  CurTrm:=0; aLine[0]:=#0;
  SendDlgItemMessage(hWindow, 107, cb_ResetContent, 0, 0);
  SendDlgItemMessage(hWindow, 101, wm_SetText, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_SetCurSel, 5, 0);
end;
```

function tRelBearbDlg.CanClose: boolean;

```
var
  aXTrm, anAttrTrm: tXRelation;
  anAttrId: LongInt;
  aLine: tLine;
  LnCo, aWord: Integer;
  aTyp: integer;
  add: bool;

begin
  CanClose:=false;{ aXTrm.relXRelation:=emptyTrm;}
  SendDlgItemMessage(hWindow, 101, wm_GetText, SizeOf(tLine), LongInt(@aLine));
  if StrLen(aLine)<1 then begin MessageBeep(0); exit; end;
  if CurTrm=0 then
  begin
    CurTrm:=RelationSuchen(aLine, 1);
    if CurTrm>0 then
      if xMessageBox(hWindow, 9, mb_iconquestion or mb_yesno)<>id_yes then begin exit; end;
  end else if xMessageBox(hWindow, 9, mb_iconquestion or mb_yesno)<>id_yes then exit;
  LnCo:=SendDlgItemMessage(hWindow, 107, cb_GetCount, 0, 0);
```

```
 Add:=false;
 if LnCo>0 then for aWord:=0 to LnCo-1 do
 begin
  SendDlgItemMessage(hWindow, 107, cb_GetLBText, aWord, LongInt(@aLine));
  anAttrID:=RelationSuchen(aLine, 1);
  if (anAttrId<1) and (not add) then
  begin
    if xMessageBox(hWindow, 10, mb_iconinformation or mb_okcancel)<>id_ok then begin exit; end;
    Add:=true;
  end;
 end;
 if CurTrm=0
  then axtrm.relXRelation:=leereRelation
  else if not RelationLesen(CurTrm, axTrm) then begin MessageBeep(0); exit; end;
 with aXTrm do
 begin
  SendDlgItemMessage(hWindow, 101, wm_GetText, SizeOf(tLine), LongInt(@aLine));
  StrCopy(relXRelation.szRelBez, aLine);
  aTyp:=SendDlgItemMessage(hWindow, id_TrmTypeCb, cb_GetCurSel, 0, 0);
  if aTyp=cb_err then aTyp:=0;
  relXRelation.LRelFlags:=MakeLong(aTyp, 0);
  RelationParamsLoeschen(relXRelation);
 end;
 if CurTrm>0 then RelationSchreiben(CurTrm, axTrm) else CurTrm:=RelationZufuegen(aXTrm);
 if CurTrm<=0 then
 begin
  xMessageBox(hWindow, 101, mb_ok or mb_IconHand);
  CanClose:=true; exit;
 end;
 if LnCo>0 then for aWord:=0 to LnCo-1 do
 begin
  SendDlgItemMessage(hWindow, 107, cb_GetLBText, aWord, LongInt(@aLine));
  anAttrId:=RelationSuchen(aLine, 1);
  if anAttrId<1  then
  begin
    with anAttrTrm do
    begin
      StrCopy(relXRelation.szRelBez, aLine);
      relXRelation.LRelFlags:=tty_xbool;
      relXRelation.HRelParam:=0;
    end;
    anAttrId:=RelationZufuegen(anAttrTrm);
    RelationParamZufuegen(CurTrm, anAttrTrm.relXRelation); RelationSchreiben(anAttrId, anAttrTrm);
  end;
  RelationParamZufuegen(anAttrId, axTrm.relXRelation);
 end;
 RelationSchreiben(CurTrm, axTrm);
 CanClose:=tDialog.CanClose;
end;
```

procedure tRelBearbDlg.DefWndProc(**var** Msg: tMessage);

```
begin
 tDialog.DefWndProc(Msg);
 if (Msg.Message = wm_ctlcolor) then
 begin
  case HiWord(Msg.lParam) of
    ctlcolor_listbox:;
    ctlcolor_edit:;
    else
      begin
        SetBkColor(Msg.wParam, hBkBrush);
        SetBkMode(Msg.wParam, Transparent);
        Msg.Result := hBkBrush;
      end;
  end;
 end;
end;
```

REPDLG.PAS

{----------TRepDlg----------}

```
type
  pRepDlg = ^tRepDlg;
  tRepDlg = object(tDlgWindow)
    EntryRect: tRect;
    destructor done; virtual;
    procedure SetupWindow; virtual;
    procedure ShowTrm; virtual;
    procedure ShowEty; virtual;
    procedure ShowCon; virtual;
    procedure ShowFilter; virtual;
    procedure ShowMasks; virtual;
    function getClassName: pchar;  virtual;
    procedure getWindowClass(Var AWndClass: TWndClass); virtual;
    procedure WmSize(var Msg: tMessage); virtual wm_first + wm_Size;
    procedure WmGetMinMaxInfo(var Msg: tMessage); virtual wm_first + wm_getminmaxinfo;
    procedure IdRepCB(var Msg: tMessage); virtual id_first + 205;
    procedure IdEditBN(var Msg: tMessage); virtual id_first + 207;
    procedure IdFindBN(var Msg: tMessage); virtual id_first + 208;
    procedure UmFind(var Msg: tMessage); virtual;
    procedure DefWndProc(var Msg: tMessage); virtual;
  end;

const
  TheReportDlg: pRepDlg = nil;
```

procedure tRepDlg.SetupWindow;

```
var
  aLine: tLine;

begin
  tDlgWindow.SetupWindow;
  LoadString(hInstance, 4, aLine, SizeOf(tLine)-5);
  SendDlgItemMessage(hWindow, 205, cb_AddString, 0, LongInt(@aLine));
  LoadString(hInstance, 6, aLine, SizeOf(tLine)-5);
  SendDlgItemMessage(hWindow, 205, cb_AddString, 0, LongInt(@aLine));
  LoadString(hInstance, 7, aLine, SizeOf(tLine)-5);
  SendDlgItemMessage(hWindow, 205, cb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, 'Masken');SendDlgItemMessage(hWindow, 205, cb_AddString, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 205, cb_SetCurSel, 0, 0);
  StrCopy(aLine, 'Filter');SendDlgItemMessage(hWindow, 205, cb_AddString, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 205, cb_SetCurSel, 0, 0);
  GetWindowRect(hWindow, EntryRect);
  ShowTrm;
end;
```

function TRepDlg.getClassName: PChar;

```
begin
  GetClassName:=TheRepClassName;
end;
```

procedure TRepDlg.getWindowClass(**var** AWndClass: TWndClass);

```
begin
  TDlgWindow.GetWindowClass(AWndClass);
  AWndClass.hIcon:=LoadIcon(HInstance, 'REPORTICO');
  AWndClass.hbrBackGround := CreateSolidBrush (GetSysColor(color_BtnFace));
end;
```

procedure tRepDlg.ShowFilter;

```
var
  aFnc, aLine: tLine;
  aX, aY: word;

begin
  EnableWindow(GetDlgItem(hWindow, 207), false);
```

```
  StrCopy(aLine, 'Filter');SendDlgItemMessage(hWindow, 201, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9'-> Funktionen');SendDlgItemMessage(hWindow, 202, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'(nicht belegt)');SendDlgItemMessage(hWindow, 203, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, '');SendDlgItemMessage(hWindow, 204, wm_settext, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 101, lb_resetContent, 0, 0);
  for aX:=1 to 16 do
  begin
    with TheFltLst[aX] do
    begin
      if @FltCallBack<>nil then
      begin
        FltCallBack(fct_GetDllName, SizeOf(tLine)-1, 0, LongInt(@aLine), 0);
        SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@aLine));
        aY:=1;
        while FltCallBack(fct_GetFncName, SizeOf(tLine)-1, aY, LongInt(@aFnc), 0)>0 do
        begin
          strcopy(aLine, #9'-> @'); StrCat(aLine, aFnc);
          SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@aLine));
          aY:=aY+1;
        end;
      end;
    end;
  end;
```

procedure tRepDlg.ShowTrm;

```
var
  aX, aY, IX: LongInt;
  aXTrm, anAttrTrm: tXRelation;
  anAttrId: tRelParam;
  aLine: tLine;

begin
  if ConvCount=0
    then EnableWindow(GetDlgItem(hWindow, 207), true)
    else EnableWindow(GetDlgItem(hWindow, 207), false);
  StrCopy(aLine, 'Prädikate');SendDlgItemMessage(hWindow, 201, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9'-> Rollen');SendDlgItemMessage(hWindow, 202, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'(nicht belegt)');SendDlgItemMessage(hWindow, 203, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, '');SendDlgItemMessage(hWindow, 204, wm_settext, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 101, lb_resetContent, 0, 0);
  for aX:=1 to RelationenAnzahl do
  begin
    RelationLesen(aX, aXTrm);
    with aXTrm do
    begin
      IX:=SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@relXRelation.SzRelBez));
      SendDlgItemMessage(hWindow, 101, lb_SetItemData, IX, aX);
      if RelationParamAnzahl(relXRelation)>0 then for aY:=1 to RelationParamAnzahl(relXRelation) do
      begin
        RelationParamLesen(aY, anAttrId, relXRelation);
        RelationLesen(anAttrId, anAttrTrm);
        strcopy(aLine, #9'-> '); StrCat(aLine, anAttrTrm.relXRelation.SzRelBez);
        IX:=SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@aLine));
        SendDlgItemMessage(hWindow, 101, lb_SetItemData, IX, anAttrId);
      end;
    end;
  end;
```

procedure tRepDlg.ShowEty;

```
var
  aX, aY: LongInt;
  anEty: tObjekt;
  anObjektWert: tObjektWert;
  aLine: tLine;
  y, m, d: word;

begin
  EnableWindow(GetDlgItem(hWindow, 207), false);
```

```
  StrCopy(aLine, 'Entitäten');SendDlgItemMessage(hWindow, 201, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9'-> Werte');SendDlgItemMessage(hWindow, 202, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'(nicht belegt)');SendDlgItemMessage(hWindow, 203, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, '* virtuell');SendDlgItemMessage(hWindow, 204, wm_settext, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 101, lb_resetContent, 0, 0);
  for aX:=1 to ObjekteAnzahl do
  begin
    ObjektLesen(aX, anEty);
    with anEty do
    begin
      StrCopy(aLine, @SzObjBez); if lObjFlags=Obj_virtual then StrCat(aLine, '*');
      SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@aLine));
      if ObjektWerteAnzahl(anEty)>0 then for aY:=1 to ObjektWerteAnzahl(anEty) do
      begin
        ObjektWertLesen(aY, anObjektWert, anEty);
        strcopy(aLine, #9'-> ');
        with anObjektWert do
        begin
          case WWertTyp of
            wt_exist: xBoolToStr(@aLine[4], XbWertExist, SizeOf(tLine)-7);
            wt_number: NumToStr(@aLine[4], DbWertZahl, 2, SizeOf(tLine)-7);
            wt_date: begin
              NumToDate(LWertDatum, y, m, d); DateToStr(@aLine[4], y, m, d, SizeOf(tLine)-7);
            end;
            wt_string: StrLCat(aLine, SzWertZeichen, SizeOf(tLine));
          end;
        end;
        SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@aLine));
      end;
    end;
  end;
end;
```

procedure tRepDlg.ShowMasks;

```
var
  aLine, dLine: tLine;
  anEty: tObjekt;
  y, m, d: word;

begin
  EnableWindow(GetDlgItem(hWindow, 207), false);
  StrCopy(aLine, 'Maske');SendDlgItemMessage(hWindow, 201, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9'Werte');SendDlgItemMessage(hWindow, 202, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'-');SendDlgItemMessage(hWindow, 203, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, '');SendDlgItemMessage(hWindow, 204, wm_settext, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 101, lb_resetContent, 0, 0);
  StrCopy(aLine, 'Eingabemaske');
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  with fltAktEingFlt do
  begin
    StrCopy(aLine, #9'Gültigkeitszeitraum');
    SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
    StrCopy(aLine, #9#9'von:'#9);
    if ObjDatAnfang>0 then
    begin
      if not ObjektLesen(ObjDatAnfang, anEty) then exit;
      StrCat(aLine, anEty.SzObjBez);
    end;
    SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
    StrCopy(aLine, #9#9'bis:'#9);
    if ObjDatEnde>0 then
    begin
      if not ObjektLesen(ObjDatAnfang, anEty) then exit;
      StrCat(aLine, anEty.SzObjBez);
    end;
    SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
    StrCopy(aLine, #9'Datenquelle:'#9);
    if ObjQuelle>0 then
    begin
      if not ObjektLesen(ObjQuelle, anEty) then exit;
      StrCat(aLine, anEty.SzObjBez);
    end;
    SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
```

```
  StrCopy(aLine, #9'Eintragungsdatum:'#9);
  if LEingabeDat>0 then
  begin
    NumToDate(LEingabeDat, y, m, d); DateToStr(dLine, y, m, d, SizeOf(tLine)-1);
    StrCat(aLine, dLine);
  end;
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9'Flags:'#9);
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  if LEFltFlags>0 then
  begin
    if isFlag(LEFltFlags, sa_exact) then
      begin StrCat(aLine, frage_SAFexact);StrCat(aLine, ' '); end;
    if isFlag(LEFltFlags, sa_InclDmy) then
      begin StrCat(aLine, frage_SAFInclDmy);StrCat(aLine, ' '); end;
    if isFlag(LEFltFlags, sa_Dummy) then
      begin StrCat(aLine, frage_SAFDummy);StrCat(aLine, ' '); end;
  end;
 end;
 StrCopy(aLine, 'Suchmaske');
 SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
 with CurSrcAttr do
 begin
  StrCopy(aLine, #9'Gültigkeitszeitraum');
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'von:'#9);
  if LGeltDatAnfang>0 then
  begin
    NumToDate(LGeltDatAnfang, y, m, d); DateToStr(dLine, y, m, d, SizeOf(tLine)-1);
    StrCat(aLine, dLine);
  end;
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'bis:'#9);
  if LGeltDatEnde>0 then
  begin
    NumToDate(LGeltDatEnde, y, m, d); DateToStr(dLine, y, m, d, SizeOf(tLine)-1);
    StrCat(aLine, dLine);
  end;
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9'Datenquelle:'#9);
  if ObjEingabeQuelle>0 then
  begin
    if not ObjektLesen(ObjEingabeQuelle, anEty) then exit;
    StrCat(aLine, anEty.SzObjBez);
  end;
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9'Eintragungszeitraum');
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'von:'#9);
  if LEingabeDatAnfang>0 then
  begin
    NumToDate(LEingabeDatAnfang, y, m, d); DateToStr(dLine, y, m, d, SizeOf(tLine)-1);
    StrCat(aLine, dLine);
  end;
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'bis:'#9);
  if LEingabeDatEnde>0 then
  begin
    NumToDate(LEingabeDatEnde, y, m, d); DateToStr(dLine, y, m, d, SizeOf(tLine)-1);
    StrCat(aLine, dLine);
  end;
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  StrCopy(aLine, #9'Flags:'#9);
  SendDlgItemMessage(hWindow, 101, lb_AddString, 0, LongInt(@aLine));
  if LAusgFltFlags>0 then
  begin
    if isFlag(LAusgFltFlags, sa_Exact) then
      begin StrCat(aLine, frage_SAFExact);StrCat(aLine, ' '); end;
    if isFlag(LAusgFltFlags, sa_InclDmy) then
      begin StrCat(aLine, frage_SAFInclDmy);StrCat(aLine, ' '); end;
    if isFlag(LAusgFltFlags, sa_Dummy) then
      begin StrCat(aLine, frage_SAFDummy);StrCat(aLine, ' '); end;
  end;
 end;
end;
```

procedure tRepDlg.ShowCon;

```
var
  aX, aY, aZ: LongInt;
  aXTrm: tXRelation;
  anEtyRel: tBeziehung;
  anEty: tObjekt;
  aLine, bLine: tLine;

begin
  EnableWindow(GetDlgItem(hWindow, 207), false);
  StrCopy(aLine, 'Prädikate');SendDlgItemMessage(hWindow, 201, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9'Hauptentitäten <Status>');SendDlgItemMessage(hWindow, 202, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, #9#9'-> Entitäten');SendDlgItemMessage(hWindow, 203, wm_settext, 0, LongInt(@aLine));
  StrCopy(aLine, '* virtuell');SendDlgItemMessage(hWindow, 204, wm_settext, 0, LongInt(@aLine));
  SendDlgItemMessage(hWindow, 101, lb_resetContent, 0, 0);
  for aX:=1 to RelationenAnzahl do
  begin
    RelationLesen(aX, aXTrm);
    if BeziehungenAnzahl(aXTrm)>0 then
    begin
      SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@axTrm.relXRelation.SzRelBez));
      for aY:=1 to BeziehungenAnzahl(axTrm) do
      begin
        BeziehungLesen(aY, anEtyRel, aXTrm);
        with anEtyRel do
        begin
          bLine[0]:=#0;
          if not xBoolToStr(bLine, XbBezWert, SizeOf(tLine)-2) then bLine[0]:=#0;
          ObjektLesen(LBezHauptObj, anEty);
          strcopy(aLine, #9#0);  StrLCat(aLine, anEty.SzObjBez, SizeOf(tLine));
          if anEty.lObjFlags=Obj_virtual then StrCat(aLine, '*');
          StrLCat(aLine, '  <', SizeOf(tLine));StrLCat(aLine, bLine, SizeOf(tLine));StrLCat(aLine, '>', SizeOf(tLine));
          SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@aLine));
          if RelationParamAnzahl(aXTrm.relXRelation)>0 then for aZ:=1 to RelationParamAnzahl(aXTrm.relXRelation) do
          begin
            ObjektLesen(LbfBezObj[aZ], anEty);
            strcopy(aLine, #9#9'-> '); StrLCat(aLine, anEty.SzObjBez, SizeOf(tLine));
            if anEty.lObjFlags=Obj_virtual then StrCat(aLine, '*');
            SendDlgItemMessage(hWindow, 101, lb_addString, 0, LongInt(@aLine));
          end;
        end;
      end;
    end;
  end;
end;
```

procedure tRepDlg.UmFind(**var** Msg: tMessage);

```
var
  FR: pFindReplace;
  startPos, aSz: integer;
  aLine, cLine: tLine;
```

function FindDown: integer;

```
var
  aWord: word;

begin
  FindDown:=-1;
  if StartPos>=aSz-1 then StartPos:=-1;
  for aWord:=StartPos+1 to aSz-1 do
  begin
    SendDlgItemMessage(hWindow, 101, lb_GetText, aWord, LongInt(@aLine));
    StrUpper(aLine);
    if StrPos(aLine, cLine)<>nil then begin FindDown:=aWord; exit; end;
  end;
  if xMessageBox(hWindow, 12, mb_iconquestion or mb_yesno)<>id_yes then exit;
  for aWord:=0 to StartPos do
  begin
    SendDlgItemMessage(hWindow, 101, lb_GetText, aWord, LongInt(@aLine));
```

```
    StrUpper(aLine);
    if StrPos(aLine, cLine)<>nil then begin FindDown:=aWord; exit; end;
  end;
  xMessageBox(hWindow, 14, mb_iconhand or mb_ok);
end;
```

function FindUp: integer;

```
var
  aWord: word;

begin
  FindUp:=-1;
  if StartPos<=0 then StartPos:=aSz;
  for aWord:=StartPos-1 downto 0 do
  begin
    SendDlgItemMessage(hWindow, 101, lb_GetText, aWord, LongInt(@aLine));
    StrUpper(aLine);
    if StrPos(aLine, cLine)<>nil then begin FindUp:=aWord; exit; end;
  end;
  if xMessageBox(hWindow, 13, mb_iconquestion or mb_yesno)<>id_yes then exit;
  for aWord:=aSz-1 downto StartPos do
  begin
    SendDlgItemMessage(hWindow, 101, lb_GetText, aWord, LongInt(@aLine));
    StrUpper(aLine);
    if StrPos(aLine, cLine)<>nil then begin FindUp:=aWord; exit; end;
  end;
  xMessageBox(hWindow, 14, mb_iconhand or mb_ok);
end;

begin {UmFind}
  FR:=Pointer(msg.lparam);
  StrCopy(cLine, FindWhatLine); StrUpper(cLine);
  aSz:=SendDlgItemMessage(hWindow, 101, lb_GetCount, 0, 0);
  if bool(FR^.Flags and FR_FindNext) then
  begin
    StartPos:=SendDlgItemMessage(hWindow, 101, lb_GetCurSel, 0, 0);
    if StartPos=lb_err then StartPos:=0;
    if bool(FR^.Flags and FR_Down) then  StartPos:=FindDown else StartPos:=FindUp ;
    if StartPos  >=0 then SendDlgItemMessage(hWindow, 101, lb_SetCurSel, StartPos, 0);
  end;
end;
```

procedure tRepDlg.DefWndProc(**var** Msg: tMessage);

```
begin
  if Msg.Message = uFindReplaceMsg then begin UmFind(Msg); end;
  tDlgWindow.DefWndProc(Msg);
  if (Msg.Message = wm_ctlcolor) then
  begin
    case HiWord(Msg.lParam) of
      ctlcolor_listbox:;
      ctlcolor_edit:;
      else
        begin
          SetBkColor(Msg.wParam, hBkBrush);
          SetBkMode(Msg.wParam, Transparent);
          Msg.Result := hBkBrush;
        end;
    end;
  end;
end;
```

procedure tRepDlg.IdFindBN(**var** Msg: tMessage);

```
begin
  uFindReplaceMsg:=RegisterWindowMessage(FINDMSGSTRING);
  aFindReplace.lStructSize:=SizeOf(tFindReplace);
  aFindReplace.Flags:=FR_nowholeword or fr_nomatchcase or fr_down;
  aFindReplace.hWndOwner:= hWindow;
```

```
  aFindReplace.lpstrFindWhat:=FindWhatLine;
  aFindReplace.wFindWhatLen:=SizeOf(tLine)-1;
  FindText(aFindReplace);
end;
```

procedure tRepDlg.ldEditBN(**var** Msg: tMessage);

```
var
  anEditTrmDlg: pRelBearbDlg;
  aTrmID: LongInt;
  IX: integer;

begin
  IX:=SendDlgItemMessage(hWindow, 101, lb_GetCurSel, 0, 0);
  if IX=lb_err then  aTrmID:=0 else
    aTrmID:=SendDlgItemMessage(hWindow, 101, lb_GetItemData, IX, 0);
  anEditTrmDlg:=New(pRelBearbDlg, init(@self, aTrmID));
  Application^.ExecDialog(anEditTrmDlg);
  ShowTrm;
  SendDlgItemMessage(hWindow, 101, lb_SetCurSel, IX, 0);
end;
```

procedure tRepDlg.ldRepCB(**var** Msg: tMessage);

```
var
  anInt: integer;

begin
  anInt:=SendDlgItemMessage(hWindow, 205, cb_GetCurSel, 0, 0);
  if HiWord(Msg.lParam)<>cbn_SelChange then exit;
  case anInt of
    0: ShowTrm;
    1: ShowEty;
    2: ShowCon;
    3: ShowMasks;
    4: ShowFilter;
  end;
end;
```

procedure tRepDlg.WmSize(**var** Msg: tMessage);

```
var
  wRect, lRect: tRect;
  nh: word;

begin

{  tDlgWindow.WmSize(Msg);}
  getWindowRect(hWindow, wRect);
  getWindowRect(getDlgItem(hWindow, 101), lRect);
  nh:=wRect.bottom-lRect.top-2-GetSystemMetrics(sm_cyFrame);
  SetWindowPos(
    getDlgItem(hWindow, 101), 0, 0, 0,
    lRect.Right-lRect.Left, nh,
    swp_nomove or swp_nozorder);
end;
```

procedure tRepDlg.WmGetMinMaxInfo(**var** Msg: tMessage);

```
type
  pPtStc = ^tPtStc;
  tPtStc = array[0..4] of tpoint;

var
  aPtStc: pPtStc;

begin
  aPtStc:=pointer(Msg.lParam);
  with EntryRect do
```

```
  begin
    aPtStc^[3].x:=right-Left;
    aPtStc^[3].y:=(bottom-top) div 2;
    aPtStc^[4].x:=right-Left;
  end;
end;
```

destructor tRepDlg.done;

```
begin
  tDlgWindow.done;
  TheReportDlg:=nil;
end;
```

ATLMAINW.PAS

{Hauptfenster}

procedure TMainWnd.AddConvToLst(Conv: hConv);

```
var
  aConvInfo: tConvInfo;
  aConvInfoP: pConvInfo;
  aServer, aClient, aTopic, aLine: tLine;
  anIx: LongInt;
  aSz, aWord: word;
  hwndServer: hwnd;

begin
  aSz:=SendDlgItemMessage(hWindow, id_MonitorLb, lb_GetCount, 0, 0);
  if aSz>0 then for aWord:=0 to aSz-1 do
    if SendDlgItemMessage(hWindow, id_MonitorLb, lb_GetItemData, aWord, 0) = Conv then exit;
  aConvInfoP:=@aConvInfo;
  DdeQueryConvInfo(Conv, 0, aConvInfoP);
  DdeQueryString(idinst, aConvInfo.hszSvcPartner, aServer, SizeOf(tLine)-1, cp_winansi);
  if aServer[0] = #0 then
  begin
    {Wenn kein Servername existiert, suchen wir den Namen des aktuellen Fensters}
    hwndServer := GetActiveWindow;
    {Mehr als 32 Zeichen wäre etwas unübersichtlich}
    GetWindowText(hwndServer, aServer, 32);
  end;
  DdeQueryString(idinst, aConvInfo.hszServiceReq, aClient, SizeOf(tLine)-1, cp_winansi);
  DdeQueryString(idinst, aConvInfo.hszTopic, aTopic, SizeOf(tLine)-1, cp_winansi);
  StrCopy(aLine, aServer); StrLCat(aLine, ' -> ', SizeOf(tLine)-1);
  StrLCat(aLine, aClient, SizeOf(tLine)-1);StrLCat(aLine, ' / ', SizeOf(tLine)-1);
  StrLCat(aLine, aTopic, SizeOf(tLine)-1);
  if StrLen(aLine)>0 then
  begin
    anIx:=SendDlgItemMessage(hWindow, id_MonitorLb, lb_addString, 0, LongInt(@aLine));
    SendDlgItemMessage(hWindow, id_MonitorLb, lb_SetItemData, anIx, Conv);
  end;
  ConvCount:=ConvCount+1;
end;
```

procedure TMainWnd.DeleteConvFromLst(Conv: hConv);

```
var
  aWord, aSz: word;

begin
  aSz:=SendDlgItemMessage(hWindow, id_MonitorLb, lb_GetCount, 0, 0);
  if aSz<1 then exit;
  for aWord:=0 to aSz-1 do
  begin
    if SendDlgItemMessage(hWindow, id_MonitorLb, lb_GetItemData, aWord, 0) = Conv then
    begin
      ConvCount:=ConvCount-1;
      SendDlgItemMessage(hWindow, id_MonitorLb, lb_DeleteString, aWord, 0); exit;
```

```
    end;
  end;
end;
```

procedure TMainWnd.SetTitle(aPChar: pChar);

```
var
  aLine, aDir, aName, anExt: tLine;

begin
  StrCopy(aLine, ProgName); StrCat(aLine, ' - [');
  if (aPChar=nil) or (StrLen(aPChar)=0) then
    StrCat(aLIne, 'ohne Namen')
  else begin
    StrLower(aPChar);
    FileSplit(aPChar, aDir, aName, anExt);
    StrLCat(aLine, aName, SizeOf(tLine)-4);
    StrLCat(aLine, anExt, SizeOf(tLine)-4);
  end;
  StrCat(aLine, ']');
  SetWindowText(hWindow, aLine);
end;
```

procedure TMainWnd.CMNew(**var** Msg: TMessage);

```
begin
  AkteNeu;
  StrCopy(CurFileName, #0);
  TheMainWnd^.SetTitle(nil);
  ShowStatus(nil, nil, false);
end;
```

procedure TMainWnd.CMSave(**var** Msg: TMessage);

```
begin
  if CurFileName[0]=#0 then CmSaveAs(Msg);
  ShowStatus('Speichern...', CurFileName, false);
  if AkteSichern(CurFileName) then SetTitle(CurFileName);
  ShowStatus(nil, nil, false);
end;
```

procedure TMainWnd.CMOpen(**var** Msg: TMessage);

```
var
  aOFN: tOpenFileName;
  aLine: tLine;

begin
  aOFN:=DefAtl2OpenFilename;
  with aOFN do
  begin
    flags := flags or ofn_FileMustExist;
    lPStrTitle:='Atlas-2 - Akte Öffnen'#0;
    hwndOwner:=hWindow;
  end;
  if GetOpenFileName(aOFN) then
  begin
    ShowStatus('Laden...', aOFN.lpstrFile, false);
    if AkteLaden(aOFN.lpstrFile) then
    begin
      FileExpand(CurFileName, aOFN.lpstrFile);
      SetTitle(aOFN.lpstrFile);
    end;
    ShowStatus(nil, nil, false);
  end;
end;
```

```
procedure TMainWnd.CMSaveAs(var Msg: TMessage);

var
  aOFN: tOpenFileName;
  aLine: tLine;

begin
  aOFN:=DefAtl2OpenFilename;
  with aOFN do
  begin
    flags := flags or ofn_PathMustExist or ofn_OverWritePrompt;
    lPStrTitle:='Atlas-2 - Akte Speichern'#0;
    hwndOwner:=hWindow;
  end;
  if GetSaveFileName(aOFN) then
  begin
    ShowStatus('Speichern...', aOFN.lpstrFile, false);
    if AkteSichern(aOFN.lpstrFile) then
    begin
      FileExpand(CurFileName, aOFN.lpstrFile);
      SetTitle(aOFN.lpstrFile);
    end;
    ShowStatus(nil, nil, false);
  end;
end;


procedure TMainWnd.IdNew(var Msg: tMessage);

begin
  TMainWnd.CmNew(Msg);
end;


procedure TMainWnd.IdSave(var Msg: tMessage);

begin
  TMainWnd.CmSave(Msg);
end;


procedure TMainWnd.IdOpen(var Msg: tMessage);

begin
  TMainWnd.CmOpen(Msg);
end;


procedure TMainWnd.IdReport(var Msg: tMessage);

begin
  TMainWnd.CmReport(Msg);
end;


procedure TMainWnd.CmSaveBase(var Msg: tMessage);

begin
  if not DatenbankSichern then
    xMessageBox(hWindow, 0, mb_ok or mb_iconhand);
end;


procedure TMainWnd.CmReport(var Msg: tMessage);

begin
  if TheReportDlg<>nil then exit;
  TheReportDlg:=New(pRepDlg, init(nil, 'REPORTDLG'));
  application^.MakeWindow(TheReportDlg);
```

```
  ShowWindow(TheReportDlg^.hWindow, sw_normal);
end;


procedure TMainWnd.WmParentNotify(var Msg: tMessage);

begin
  {Do Nothing}
end;


procedure TMainWnd.CmUpdate(var Msg: tMessage);

var
  aDlg: pInfoWnd;

begin
  ShowStatus(nil, nil, false);
end;


procedure TMainWnd.CmInfo(var Msg: tMessage);

var
  aDlg: pInfoWnd;

begin
  aDlg:=New(pInfoWnd, init(@self, 'INFODLG', ProgName, ProgVer, ProgDate));
  application^.ExecDialog(aDlg);
end;


constructor TMainWnd.Init(AParent: PWindowsObject; ATitle: PChar);

begin
  TDlgWindow.Init(AParent, 'MAINWND');
  @CallBack := MakeProcInstance(@DDECallBack, HInstance);
  DDEInitialize(idInst, CallBack, 0, 0);
  HszServName:=DDECreateStringHandle(idInst, ProgService, cp_winansi);
  HszDefConvTopic:=DDECreateStringHandle(idInst, DefConvTopic, cp_winansi);
  DDENameService(idInst, HszServName, HszDefConvTopic, dns_register);
  if not noDebug then
  begin
    assign(debugfile, debugfilename);
    rewrite(debugFile);
  end;
  InitAtlFnc;
  DatenbankInstallieren;
  fltAktEingFlt:=InpFltDmy;
  fltAktEingFlt.LEingabeDat:=GetIDate;
  CurSrcAttr:=DummySrcAttr;
  hBkBrush := CreateSolidBrush (GetSysColor(color_BtnFace));
end;


procedure TMainWnd.SetupWindow;

var
  aLine:tLine;

begin
  tDlgWindow.SetupWindow;
  AppendMenu(GetSystemMenu(hWindow, false), mf_Separator or mf_SysMenu, 0, nil);
  AppendMenu(GetSystemMenu(hWindow, false), mf_SysMenu, cm_report, '&Report...');
  HszFileConvTopic:=0;

{  DatenbankLaden; in install lists ***}
  SetTitle(nil);
  if GetArgCount>0 then
  begin
    GetArgStr(aLine, 1, SizeOf(tLine)-1);
```

```
   if AkteLaden(aLine) then
   begin
     TheMainWnd^.SetTitle(aLine);
     FileExpand(CurFileName, aLine);
   end;
 end;
 InitFilters(hWindow);
 ShowStatus(nil, nil, false);
end;
```

procedure TMainWnd.DefWndProc(**var** Msg: tMessage);

```
begin
 tDlgWindow.DefWndProc(Msg);
 if (Msg.Message = wm_ctlcolor) then
 begin
   case HiWord(Msg.lParam) of
     ctlcolor_listbox:;
     ctlcolor_edit:;
     else
       begin
         SetBkColor(Msg.wParam, hBkBrush);
         SetBkMode(Msg.wParam, Transparent);
         Msg.Result := hBkBrush;
       end;
   end;
 end;
end;
```

procedure TMainWnd.WmSysCommand(**var** Msg: tMessage);

```
begin
 if msg.wParam = cm_Report then
 begin
   if TheReportDlg<>nil then exit;
   TheReportDlg:=New(pRepDlg, init(nil, 'REPORTDLG'));
   application^.MakeWindow(TheReportDlg);
   ShowWindow(TheReportDlg^.hWindow, sw_normal);
 end else
   tDlgWindow.DefWndProc(Msg);
end;
```

function TMainWnd.getClassName: PChar;

```
begin
 GetClassName:=TheClassName;
end;
```

procedure TMainWnd.getWindowClass(**var** AWndClass: TWndClass);

```
begin
 TDlgWindow.GetWindowClass(AWndClass);
 AWndClass.hIcon:=LoadIcon(HInstance, 'MAINICO');
 AWndClass.hbrBackGround := CreateSolidBrush (GetSysColor(color_BtnFace));
end;
```

destructor TMainWnd.Done;

```
var
 aTrm: pRelation;
 aWord, aCount: word;

begin
 if not noDebug then Close(DebugFile);
 StrCopy(CurFileName, #0);
 DDENameService(idInst, HszServName, HszDefConvTopic, dns_unregister);
```

```
  if HszFileConvTopic>0 then
  begin
    DDENameService(idInst, HszServName, HszFileConvTopic, dns_unregister);
    DdeFreeStringHandle(idInst, HszFileConvTopic);
  end;
  DdeFreeStringHandle(idInst, HszServName);
  DDEUninitialize(idInst);
  FreeFilters;
  FreeProcInstance(@CallBack);
  DatenbankDeInstallieren;
  DeleteObject(hBkBrush);
  TDlgWindow.Done;
end;
```

function TMainWnd.CanClose: Boolean;

```
var
  mbResult: integer;

begin
  CanClose:=false;
  if not TDlgWindow.CanClose then exit;
  if ConvCount>0 then
    if xMessageBox(hWindow, 1, mb_yesno or mb_iconhand or mb_DefButton2)<>id_yes then exit;
  if StrLen(CurFileName)>0 then
  begin
    mbResult:=xMessageBox(hWindow, 2, mb_yesnocancel or mb_iconquestion);
    case mbResult of
      id_yes: AkteSichern(CurFileName);
      id_cancel: exit;
    end;
  end;
  {In älteren Versionen wurde immer gefragt, ob die globale Datenbasis
  gespeichert werden soll. Gestrichen seit 96-01-27.
  mbResult:=xMessageBox(hWindow, 11, mb_yesnocancel or mb_iconquestion);
  Case MbResult of
    id_yes:  if not DatenbankSichern then  xMessageBox(hWindow, 0, mb_ok or mb_iconhand);
    id_cancel: exit;
  end;}
  if not DatenbankSichern then  xMessageBox(hWindow, 0, mb_ok or mb_iconhand);
  CanClose:=true;
end;
```

Teil VI. Verzeichnisse

A. Abbildungen

B. Literatur

AIDA als gemeinsame Datenstrukktur, CoR 4/93, 3.

Alexy, R.: Expert Systems and Legal Theory, in: Expert Systems in Law: Impacts in Legal Theory (Fiedler, H.; Haft, F.; Traunmüller, R. Hrsg.), Tübingen: Attempo, 1988.

Bähring, W.; Roschmann, C.; Schäffner, L.: Das Mandantengespräch, Theorie - Besonderheiten - Regeln, Essen 1989.

Bartsch, M.: Anwaltssoftware - Qualitätsstandard, Probleme und Wünsche, in: Computerintegrierter Arbeitsplatz im Büro - Informatik Fachberichte 156, Heidelberg 1987.

Bauer, A.; Lichtner, R.: Computertechnologie im Anwaltsbüro, München 1988.

Becker, H.: Der juristische Arbeitsplatz der Zukunft aus anwaltlicher Sicht, in: Informationstechnik am Arbeitsplatz von Juristen..., Köln 1989.

Birkigt, K.; Waltl, P.: Umbruch - Anwaltssoftware SNI Jupiter im Test, CoR 5/91, 19.

Blaser, A.; Jarke, M.; Lehmann, H.; Müller, G.: Datenbanksprachen und Datenbaknbenutzung, in: Datenbankhandbuch (Lockeman, P.C.; Schmidt, J.W. Hrsg.), Heidelberg 1987.

Bönninger, Ingrid & Karl: Grundzüge des juristischen Expertensystems JUREX; jur-pc 90, 683 ff.

Börger, E.: Berechenbarkeit, Komplexität, Logik: Eine Einführung in die Algorithmen, Sprachen und Kalküle unter besonderer Berücksichtigung ihrer KomplexitätBraunschweig 1986.

Britz, J.: Urkundenbeweisrecht und Elektroniktechnologie - Eine Studie zur Tauglichkeit gesetzlicher Beweisregeln für elektronische Dokumente und ihre Reproduktionen im Zivilprozeß, München 1996 (vermutlich - Zum Zeitpunkt der Fertigstellung dieser Arbeit noch nicht veröffentlicht).

Brudermüller, G.; Klattenhof, R.: Tabellen zum Familienrecht - 7. Aufl., Bonn 1992

Brühl, R. (Hrsg.): Die juristische Fallbearbeitung in Klausur, Hausarbeit und Vortrag, 2. Aufl., Köln 1989.

Büchting, H.-U.; Heussen, B. (Hrsg.): Beck'sches Rechtsanwaltshandbuch 1993/94, München 1993 (Abk.: *Rechtsanwaltshandbuch 93/94*. Aus dem Handbuch 1995/96 ist der EDV-Teil entfallen, so daß teilweise dieser Band zitiert wird.)

Büchting, H.-U.; Heussen, B.: Beck'sches Rechtsanwaltshandbuch 1995/96, München 1995

Bund, E.: Einführung in die Rechtsinformatik, Heidelberg 1991 (Abk. *Bund*).

Bund, E.: Juristische Logik und Argumentation, Freiburg 1983.

Bydlinski, F.: Juristische Methodenlehre und Rechtsbegriff, 2. Aufl., Wien 1991.

Chiotellis, A.; Fikentscher, W. Hrsg.: Rechtstatsachenforschung: Methodische Probleme und Beispiele aus dem Schuld- und Wirtschaftsrecht (Chiotellis, A.; Fikentscher, W. Hrsg.), Köln 1985.

Commichau, Gerhard: Die Anwaltliche Praxis in Zivilsachen: c. Einf. in d. Anwaltstätigkeit, 3. Aufl., Stuttgart 1988 (Abk. *Commichau*).

Cremens, A.; *Griefahn, U.*; *Hinze, R.*: Deduktive Datenbanken, Braunschweig/Wiesbaden 1994.

Ebeling, S.: ArBIS - Experten im Hyperraum; jur-pc 91, 1237.

Ehrlich, E.: Die juristische Logik - 2. Aufl (1925 - Neudruck), Tübingen 1966

Endrös, A.: Anwaltsforschung und EDV; jur-pc 90, 466 ff.

Endrös, A.: Computergestützte Strategie in der Anwaltspraxis - Das neue Denken; CoR 6/89, 22 ff.

Engisch, K.: Logische Studien zur Gesetzesanwendung, (Sitzungsberichte der Heidelberger Akademie der Wissenschaften, phil.-hist. Kl., Jahrgang 1960) 3. Aufl., Heidelberg 1963.

Erdmann, U.; Fiedler, H.; Haft, F.; Traunmüller, R.: Computergestützte juristische Expertensysteme, Tübingen 1986.

Expertensysteme - quo vadis? - Neues aus der Praxis von Fachtagungen; CoR 1/90 S. 23, 25

Fiedler, H.: Die Rechtsfindung aus dem Gesetz im Lichte der neueren Logik und Methodenlehre, in: Festschrift für Ulrich Klug zum 70. Geburtstag (Kohlmann, G Hrsg.), Köln 1983.

Fiedler, H.; Barthel, T.; Voogt, G.: Untersuchungen zur Formalisierung im Recht als Beitrag zur Grundlagenforschung juristischer Datenverarbeitung, Opladen 1984.

Fiedler, H.; Haft, F.; Traunmüller, R. (Hrsg.): Expert Systems in Law: Impacts in Legal Theory (Fiedler, H.; Haft, F.; Traunmüller, R. Hrsg.), Tübingen 1988.

Fiedler, H.; Traunmüller, R: Methodisches Vorgehen in Recht und Informatik im Vergleich - GI Jahrestagung 1989; Fachgespräch Computergestützter Arbeitsplatz und Arbeitsmethoden, Informatik Fachberichte Bd. 223, Berlin 1989.

Fiedler, H: Logische Struktur und informationstechnische Unterstützung richterlicher Rechtsprechung, in: Rechtsprechungslehre (int. Symposium in Münster, 1984) (Achterberg, N. Hrsg.) - Köln 1986, S. 311-327.

Gerblinger, M.: Durchblicke - Windows als Systembasis juristischer Anwendungen, CoR 2/91 9 ff.

Germ, M.: Lohnt sich der Computer in der Kanzlei - Vergleich konventionelle vs. computergestützte Arbeitsabläufe, CoR 2/90, 28 ff.

Gordon, T. F.: Künstliche Intelligenz und Recht; jur-pc 90, S. 605 ff., 638 f.

Gordon, T. F.: OBLOG-2: Ein hybrides Wissensrepräsentationssystem zur Modellierung rechtswissenschaftlicher Probleme, in: Informatikanwendungen Trendes und Perspektiven (Hommel, G.; Schnindler, S. Hrsg.), Informatik Fachberichte 127, Berlin 1986.

Günther, A.: Juristische Expertensysteme, - Gedanken zwischen Theorie und Praxis, 3-teiliger Beitrag; jur-pc 89, 309 ff.; 365 ff.; 90, 428 ff.

Gutdeutsch, W.: Erfahrungen beim Einsatz von Informationstechnik in Familiensachen - In: Informationstechnik am Arbeitsplatz von Juristen..., Köln 1989

Haft, F.: Einführung in das juristische Lernen, 4. Aufl. Bielefeld 1988.

Haft, F.: Rethorik und Computer - Die Normalität am Problemfall lernen; CoR 3/89, 15.

Hatz, H.: Rechtssprache und juristischer Begriff - Vom richtigen Verstehen des Rechtssatzes (Res Publica 10), Stuttgart 1963.

Herber, M.; *Simons, D.*: Wissenschaftstheorie für Juristen, Frankfurt/M. 1980.

Herberger, M.: Die Relationsmaschine; jur-pc 91, 1259.

Herberger, M.: Textverarbeitung und Wissensrepräsentation Oder: Das nur scheinbar Unschuldige Instrument; jur-pc 89, 116 ff.

Hoffmann, H.: PC-Praxis für Juristen, 2. Aufl., München 1996.

Hoffmann, H.: REF - Datenbank zur Referats-Verwaltung des Richters; jur-pc 91, 901 ff.

Hoppenz, R.: Familiensachen: Kommentar anhand der Rechtsprechung des Bundesgerichtshofs, 4. Aufl., Heidelberg 1992.

Informationstechnische Unterstützung von Richtern, Staatsanwälten und Rechtspflegern (JURISTAR), Köln: Bundesanzeiger 1992.

Kalthoener, E.; Büttner, H.: Rechtsprechung zu Höhe des Unterhalts, 5. Aufl., München 1993.

Karagiannis, D.: Wissensbasierte Datenbanken, Wien 1994.

Kilian, W.: Auswirkungen der Informationstechnologie auf rechtliche Prinzipien. In: Technischer Imperativ und legitimationskrise des Rechts; 1991 S. 353 - 364.

Kirchner, T.: Arbeitsabläufe und Textverarbeitung, CoR 1995, 324 ff.

Korte, W.: Pythia lüftet ihr Geheimnis - Der Entwurf einer Datenstruktur im relationalen Datenbankmodell; jur-pc 91, 1004 ff.

Kowalewski, D. L.; Schneeberger, J.: KOKON: Wissensbasierte Konfigurierung von Verträgen: in Informatikanwendungen Trendes und Perspektiven (Hommel, G.; Schnindler, S. Hrsg.), Informatik Fachberichte 127, Berlin 1986.

Kraft, M.: Alter Traum in neuem Gewande? LCD-Tablett und Windows für Pen–Computer, jur-pc 93, 2331 ff.

Kraft, M.: ObjectVision - Workshop: Automatische Tenorierung der vorläufigen Vollstreckbarkeit, jur-pc 93, 1977 ff.

Kraft, M.: Sophos 1.0 - Eine Wissensdatenbank, jur-pc 90, 496 ff.

Kraft, M.: Windows 3.0 - eine Alternative zur Branchenlösung? jur-pc 91, 971 ff.

Kuhn, M.: Rechtshandlungen mittels EDV und Telekommunikation, München 1991.

Larenz, K.: Methodenlehre der Rechtswissenschaft (Enzyklopädie der Rechts- und Staatswissenschaft; Albach, H.; Helmstädter, E.; Honsell, H.; Lerche, P.; Nörr, D. Hrsg.), 6. Aufl., Berlin 1991.

Lennartz, H.-A.: Rechtliche Steuerung informationstechnischer Systeme, Braunschweig 1993.

Ludwig, Th.; Walter, B.; Ley, M.; Maier, A.; Gehlen, E.: LILOG-DB: Database Support for Knowledge-Based Systems - in Datenbanksysteme in Büro, Technik und Wissenschaft (Härder, T. Hrsg.), Informatik Fachberichte 204, Berlin 1989.

Lüttringhaus, S.: Deduktive Datenbanksysteme: Theoretische und praktische Aspekte, Forschungsbericht Nr. 231 der Abteilung Informatik der Universität Dortmund, Dortmund, 1987.

Mähler, R.: Effektive Organisation und moderne Kommuniktaion in der Anwaltskanzlei, Köln 1989.

Maier, A.: Einbettung von Konzepthierarchien in ein Deduktives Datenbanksystem, St. Augustin 1993.

Mertl, A.: Familien-Sachen - Die familenrechtlichen Programme von Jakob ud Gutdeutsch im Vergleich, CoR 4/91, 17 ff.

Möller, T.; *Weinknecht, J.*: EDV in der Kanzlei - NoRA II, jur-pc 89, 111 ff.

Möller, T.: Juris für Juristen - Optimierte juris–Nutzung mit den Mitteln der Rechtsinformatik unter berücksichtigung überkommener juristischer Methodenlehre, zugleich win Beitrag zur Abbildung des deduktiven Hauptschemas der analytischen Begründungslehre als Computermodell, Dissertation, Saarbrücken 1993.

Morgenstern, R. E.: Tabularius, Programmkomponenten für die kleinere bis mittlere Kanzlei, jur-pc 93, 1904 ff.

Nack, A.: Anforderungen an die Informationstechnik am Arbeitsplatz von Richtern - Die Arbeit am Sachverhalt - in: Informationstechnik am Arbeitsplatz von Juristen..., Köln 1989.

Neske, F.: Daten-Manager, Wie man sich ein internes Informationssystem aufbaut; CoR 3/92, 8 ff.

Neske, F.: Kanzleianalyse - Die Aufgaben des Anwalts und der Computer, CoR 2/88 23 f.

Nilgens, V.: jur-pc Diskettenbeilage: Juristische Programme von Volker Nilgens; jur-pc 93, 2164 ff.

Nilgens, V.: PHANTASY WINDOWS - Software für Juristen? jur-pc 92, 1883 f.

Oechsler, J.: Wissensbasierte Konsultationssysteme im Recht: SUSA - Ein praktisches Anschauungsbeispiel, jur-pc 91, 1205

Palandt: Bürgerliches Gesetzbuch, 55. Aufl., München 1996

Philipps, L.: Der Computer als Hilfsmittel zu einer interessengerechten Normierung, DVR Beiheft 17, München 1984.

Philipps, L.: Naheliegende Anwendungen Neuronaler Netze in der Rechtswissenschaft; jur-pc 90, 820 ff.

Preiß, N.: Ein Konzept für die dedukktive Erweiterung eines relationalen Datenbanksystems, Karlsruhe, 1989.

Raden, L. v.: IT-Unterstützte Richter und Staatsanwaltsarbeitsplätze - Erfahrungen, Erwartungen, Akzeptanz - in Computerintegrierter Arbeitsplatz im Büro, Informatik Fachberichte 156, Berlin 1987.

Raden, L. v.; Strempel, D.: Anleitung zum Weiterfragen - *Juristar* zwischen Erwartung und Enttäuschung; jur-pc 91, 989 ff.

Raden, L. v.; Weihermüller, M. (Hrsg.): Informationstechnik am Arbeitsplatz von Juristen - Neuere Entwicklungen unter besonderer Berücksichtigung von Rechtsinformationssystemen, Köln 1989 (Abk. *Informationstechnik...*).

Rödig, Jürgen: Über die Notwendigkeit einer besonderen Logik der Normen - Hier in: Schriften zur juristischen Logik (Bund, E; Schmiedel, B.; Thieler-Mevissen, G. Hrsg), Berlin 1979.

Rüßmann, H.: Die Grenzen der Anwendung von Denkgesetzen, oder , Der Bundesgerichtshof im elektronischen Notstand, jur-pc 90, 661 ff.

Rüßmann, H.: Restschuldminderung nach § 12 Abs. 2 des Verbraucherkreditgesetzes (VerbrKrG) - Zur Nützlichkeit (Notwendigkeit) des Einsatzes der Tabellenkalkulation zur Bewältigung juristischer Aufgaben, jur-pc 94, 2828 ff.

Schedel, D.: RAMANDATA - Integrierte Kanzlei-EDV zwischen Aktenverwaltung und Rechtsfindung - jur-pc 93, 2212 ff.

Schleicher, K.: Am Anfang war die Information; jur-pc 91, 1216 f.

Schneider, E.: Logik für Juristen, München, 1991.

Schneider, E.: Richterliche Arbeitstechnik, 3. Aufl., München 1991.

Schnelle, H.: Der elektronisch gestützte Zivilprozeß - das neue Stuttgarter Modell, in: DRiZ 1993/3, 97 ff.

Schultze, M.: Der Richter und der Computer - Erfahrungen eines Praktikers; CoR 2/90, 24 ff.

Sirp, W.: Bericht, Gutachten und Urteil, 31. Aufl. München, 1989. (Abk. *Sattelmacher/Sirp).*

Stollenwerk: Die Antragsschrift in Scheidungs- und Folgesachen. 3. Aufl, Köln 1979.

Suhr, D.: Der Computer als juristischer Gesprächspartner, Berlin 1970.

Tettinger, P. J. : Einführung in die juristische Arbeitstechnik, 2. Aufl. München, 1992.

Thomas, H.; *Putzo, H.*: Zivilprozeßordnung, 19. Aufl., München 1995.

Tzschaschel, H.-U.: Vereinbarungen bei Trennung und Ehescheidung, Heidelberg 1992.

Vespermann: Familiensachen, Band 1, Scheidungs- und Scheidungsverbundverfahren, Müchen 1990.

Viefhues, N.;*Viefhues, W.*: Keine Angst vor Baumbach, CoR 1/92, 21 ff.

Viefhues, W.: Methodik der Vergleichenden Untersuchung von Justizsoftware - Am Beispiel der Programme Sijus–Familie und Sojus–FAM

Viefhues, W.: SUPERFMAM - Unterhalt und Versorgungsausgleich, jur-pc 89, 295.

Viefhues, W.: UNT - Neues Unterhaltsberechnungsprogramm, jur-pc 93, 2179.

Viefhues, W.: Unterschiede - Drei Unterhaltsberechnungsprogramme im Vergleich, CoR 2/92, 21 ff.

Walter, G.: Integrierte EDV-Unterstützung für den Arbeitsplatz des Richters, Staatsanwalts und Rechtspflegers, jur-pc 91, 897 ff.

Waltl, P.: Immer Ärger mit der Kollisionskontrolle - Was Anwaltsprogramme leisten sollten, CoR 6/92 18 ff.

Waltl, P.: Stammhalter, Das Anwaltsprogramm MANDANT im Test, CoR 4/93, 15 ff.

Waltl, P.; *Rosenberger, C.*: Beifahrer - Das Anwaltsprogramm Sozius im Test; CoR 2/93,14 ff.

Weinberger, O.: Studien zur Normlogik und Rechtsinformatik, Berlin 1974 (Abk. *Weinberger*).

Wendl, P.; Stautigl, S.: Das Unterhaltsrecht in der familienrechtlichen Praxis, 2. Aufl., München 1990.

KRIMINALWISSENSCHAFTLICHE STUDIEN

Herausgegeben von Prof. Dr. Dieter Meurer, Philipps-Universität Marburg

KS 1 • PERSONALDELIKTE. Parallelen und Abweichungen zum Ladendiebstahl.
Von Dr. Brigitte Schmechtig. 1982. X, 114 Seiten (ISBN 3-7708-0740-5) DM 42,-.

KS 2 • DIE ZULASSUNG DER RECHTSBESCHWERDE IM BUSSGELDVERFAHREN.
Von Dr. Peter Baukelmann. 1983. XVIII, 332 Seiten (ISBN 3-7708-0780-4) DM 98,-.

KS 3 • STRAFE UND ERZIEHUNG NACH DEM JUGENDGERICHTSGESETZ.
Von Dr. Gerhard Wolf. 1984. XXIV, 387 Seiten (ISBN 3-7708-0796-0) DM 98,-.

KS 4 • SEXUALDELINQUENZ UND POLIZEIVERHALTEN unter besonderer Berücksichtigung der Vergewaltigung. Von Dr. Thorsten Kahl. 1985. XII, 155 Seiten (ISBN 3-7708-0830-4) DM 32,-.

KS 5 • DIE RECHTLICHE ZULÄSSIGKEIT POSTMORTALER TRANSPLANTAT-ENTNAHME.
Von Dr. Volker Albrecht. 1986. XV, 135 Seiten (ISBN 3-7708-0862-2) DM 29,70.

KS 6 • STRAFRECHTSPFLEGE AM ENGLISCHEN MAGISTRATES´ COURT.
Von Dr. Achim Lauermann. 1987. XVIII. 227 Seiten (ISBN 3-7708-0868-2) DM 39,50.

KS 7 • DIE NICHTBEACHTUNG DES ZWEI-DRITTEL-ZEITPUNKTS IN DER VOLLSTRECKUNG DES STRAFGERICHTLICHEN FREIHEITSENTZUGS. Ein Beitrag zur Dogmatik der Reststrafenaussetzung zur Bewährung und zur Rechtstatsachenforschung.
Von Dr. Thomas Wolf. 1988. XVI, 169 Seiten (ISBN 3-7708-0877-0) DM 34,-.

KS 8 • ZUR STRAFBARKEIT DER »KETTENANSTIFTUNG«.
Von Dr. Kurt Sippel. 1989. XI, 118 Seiten (ISBN 3-7708-0920) DM 21,-.

KS 9 • RAUSCHMITTEL IM STRASSENVERKEHR. Eine Untersuchung über Medikamente als Rauschmittel im Sinne der §§ 315 c, 316 StGB.
Von Dr. Ellen Ulbricht. 1990. XV, 362 Seiten (ISBN 3-7708-0934-3) DM 98,-.

KS 10 • ZUM BEWEISWERT VON PERSONENIDENTIFIZIERUNGEN. Neuere empirische Befunde.
Von Prof. Dr. Dieter Meurer und Siegfried Sporer, Ph. D. (Hrsg.). 1990. XIV, 196 Seiten, 40 Seiten Anhang (ISBN 3-7708-0942-4) DM 92,-.

KS 11 • DIE »KLARSTELLUNGSFUNKTION« DER IDEALKONKURRENZ. Ein Beitrag zur Dogmatik der Konkurrenzlehre.
Von Dr. Peter Abels. 1991. XVI, 103 Seiten (ISBN 3-7708-0935-1) DM 39,-.

KS 12 • SUMMA CRIMINOLOGICA. Ausgewählte Schriften zur Kriminologie aus den Jahren 1952 bis 1991. Band I: 1952 bis 1979.
Von Prof. Dr. Richard Lange. Neu hrsg. und eingeleitet von Prof. Dr. Dieter Meurer. 1991. X, 321 Seiten (ISBN 3-7708-0952-1) DM 96,-.

KS 13 • SUMMA CRIMINOLOGICA. Band II: 1980 bis l991.
Von Prof. Dr. Richard Lange. Neu hrsg. und eingeleitet von Prof. Dr. Dieter Meurer. 1991. VI, 389 Seiten (ISBN 3-7708-0953-X) DM 96,-.

KS 14 • GERICHTSSPRACHENPROBLEMATIK IM STRAF- UND BUSSGELDVERFAHREN.
Von Dr. Jürgen Weith. 1992. XI. 151 Seiten (ISBN 3-7708-0980-7) DM 45,-.

KS 15 • DOGMATISCHE GRUNDLAGEN DER VERWERTUNGSVERBOTE. Eine Untersuchung über die Strukturen strafprozessualer Verwertungsverbote unter dem Einfluß der Verfassung und der Grundsätze des öffentlichen Rechts.
Von Dr. Rainer Störmer. 1992. XXIII, 306 Seiten (ISBN 3-7708-0989-0) DM 95,-.

KS 16 • DIE ENTSCHEIDUNGSFINDUNG DURCH SCHÖFFEN UND BERUFSRICHTER IN RECHTLICHER UND PSYCHOLOGISCHER SICHT. Empirische, rechtsdogmatische und psychologisch-theoretische Untersuchungen zur Laienbeteiligung an der Strafgerichtsbarkeit.
Von Dr. Christoph Rennig. 1993. XXXIII, 724 Seiten (ISBN 3-7708-0992-0) DM 178,-.

KS 17 • DAS GLÜCKSSPIEL IM STRAFRECHT.
Von Dr. Axel Belz. 1993. XVI, 140 Seiten (ISBN 3-7708-1004-X) DM 48,-.

KRIMINALWISSENSCHAFTLICHE STUDIEN

Herausgegeben von Prof. Dr. Dieter Meurer, Philipps-Universität Marburg

KS 18 • FREIHEIT DURCH ARBEIT. Die Institution der wohltätigen Anrechnung von Arbeitstagen auf die Freiheitsstrafe in Griechenland.
Von Dr. Spiros Frangoulis. 1994. XIII, 137 Seiten (ISBN 3-7708-1024-4) DM 48,-.

KS 19 • DIE BEEINFLUSSBARKEIT VON ZEUGENAUSSAGEN.
Von Prof. Dr. Dieter Meurer und Siegfried Sporer, Ph.D. (Hrsg.) 1994. XV, 320 Seiten
(ISBN 3-7708-1037-6) DM 68,-:

KS 20 • DER MISSBRAUCH VON TITELN, BERUFSBEZEICHNUNGEN UND ABZEICHEN. Rechtsgut, Schutzzweck und Anwendungsbereich des § 132a StGB.
Von Dr. Franz Kahle. 1995. XXIX, 389 Seiten (ISBN 3-7708-1056-2) DM 98,-

KS 21 • SYSTEMATISCHE UNTERSUCHUNGEN ZUR OFFENKUNDIGKEIT IM STRAFPROZESS.
Von Prof. Dr. Eva Graul. 1996. XXVI, 457 Seiten (ISBN 3-7708-1067-8) DM 112,-

KS 22 • DIE ENTSCHEIDUNGEN DES REICHSOBERHANDELSGERICHTS IN STRAFSACHEN.
Von Dr. Axel Weiß. 1997. XXII, 300 Seiten (ISBN 3-7708-1076-7) DM 96,-

KS 23 • GELDWÄSCHE UND TÄTIGE REUE. Eine Untersuchung zu Auslegung und Anwendung der besonderen Rücktrittsregelungen in § 261 Abs. 9 und 10 StGB.
Von Dr. Thomas Fabel. 1997. XXII, 256 Seiten (ISBN 3-7708-1080-5) DM 78,-

COMPUTER IM RECHT

Herausgegeben von Prof. Dr. Dr. Jörg Berkemann, Berlin,
Prof. Dr. Maximilian Herberger, Saarbrücken, Prof. Dr. Dieter Meurer, Marburg

CiR 1 • ANWALTSHAFTUNG FÜR COMPUTERFEHLER. Haftungsrisiken beim Einsatz elektronischer Datenverarbeitung in Anwaltskanzleien und Notariaten.
Von Dr. Axel Benning. 1992. XI, 137 Seiten (ISBN 3-7708-0977-7) DM 44,-

CiR 2 • CELEX. Profil einer Datenbank.
Von Markus Ruffing. 1994. XII, 195 Seiten (ISBN 3-7708-1023-6) DM 48,-

CiR 3 • METHODIK DER VERGLEICHENDEN UNTERSUCHUNG VON JUSTIZSOFTWARE am Beispiel der Programme Sijus-Familie und SOJUS-FAM für die Geschäftsstelle des Familiengerichts und dreier Unterhaltsberechnungsprogramme.
Von Dr. Wolfram Viefhues. 1994. XVIII, 320 Seiten (ISBN 3-7708-1031-7) DM 95,-

CiR 4 • COMPUTERSCHRIFTZEICHENSCHUTZ. Der Schutz typographischer Schriftzeichen - insbesondere Computerschriftzeichen - durch das Schriftzeichengesetz und andere Vorschriften.
Von Dr. Susanne Brinkhoff. 1995. XI, 203 Seiten (ISBN 3-7708-1047-3) DM 49,-

CiR 5 • RICHTERLICHES INFORMATIONSMANAGEMENT UND DATENSCHUTZ.
Von Dr. Franz-Josef Kockler. 1996. XII, 275 Seiten. (ISBN 3-7708-1058-9) DM 68,-

CiR 6 • NICHT-LINEARES INFORMATION-RETRIEVAL IN DER JURISTISCHEN INFORMATIONSSUCHE.
Von Dr. Frank Krüger. 1997. XII, 201 Seiten. (ISBN 3-7708-1081-3) DM 55,-

CiR 7 • VORSTUDIEN ZUR COMPUTERGESTÜTZTEN LOGISCHEN STRUKTURANALYSE DES UN-KAUFRECHTS.
Von Dr. Martin Gitzinger. 1999. XII, 297 Seiten. (ISBN 3-7708-1117-8) DM 69,-

CiR 8 • AUSTAUSCH JURISTISCH RELEVANTER FAKTEN ZWISCHEN HETEROGENEN ANWENDUNGEN. Beschreibung eines theoretischen Ansatzes und Realisierung eines Prototyps mit Beispielen aus dem Familienrecht.
Von Dr. Matthias Kraft. 1999. XVII, 349 Seiten. (ISBN 3-7708-1118-6) DM 97,-

Bei Bezug oder Abonnement der gesamten Reihe(n) gewährt der Verlag einen Preisnachlaß von 10 %.