Entwurf und Implementierung einer Benutzerschnittstelle für ALFRED

Informatikprojekt

am Institut für Informatik und angewandte Mathematik IAM
der Universität Bern

von Roger Blum

im Mai 1997

Betreut durch:

Prof. Dr. Oscar Nierstrasz
Inst. für Informatik (IAM)
Universität Bern
Neubrückstr. 10
3012 Bern

Dipl. Inf. Markus Schlesinger
Inst. für Wirtschaftsinformatik (IWI)
Universität Bern
Engehaldestr. 8
3012 Bern

Zusammenfassung

Aktive Datenbanksysteme erweitern herkömmliche Datenbanksysteme um die Fähigkeit, selbständig auf gewisse Situationen zu reagieren. Am Institut für Wirtschaftsinformatik, Abteilung Information Engineering, der Universität Bern wird die aktive Schicht ALFRED (**A**ctive **L**ayer **F**or **R**ule **E**xecution in **D**atabase Systems) entwickelt. Damit kann prinzipiell jedes beliebige (passive) Datenbanksystem in ein aktives verwandelt werden.

In diesem Projekt wird die Benutzeroberfläche, basierend auf festgelegten funktionellen und systemtechnischen Anforderungen, entworfen und implementiert. Ein besonderes Gewicht wurde dabei auf die Benutzungsfreundlichkeit und die Plattformunabhängigkeit gelegt. Ferner wird ein Konzept für die automatische Ableitung von Regeln für die Gewährleistung von Integritätsbedingungen erarbeitet. Das aktive Verhalten wird in ALFRED somit vollständig durch Regeln realisiert.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Aktive Datenbanksysteme erweitern traditionelle Datenbanksysteme (DBMS) um Konzepte und Mechanismen, mit denen auf das Eintreffen bestimmter Situationen automatisch mit der Ausführung von gewissen Aktionen reagiert werden kann. Aktives Verhalten, wie z. B. Trigger und Integritätsbedingungen, wird durch Regeln festgelegt. Diese beschreiben bestimmte *Situationen* und beinhalten *Aktionen*, die angeben, wie auf diese Situationen reagiert werden soll (vgl. Kapitel 2.2).

Viele aktive Datenbanksysteme unterstützen aber die Administration einer Regelmenge sowie die Möglichkeit das aktive Verhalten mit dem Gewünschten zu vergleichen nur schlecht oder gar nicht. Auch Werkzeuge und eine Entwicklungsumgebung (mit Debugger, Regel-Browser, usw.) fehlen meist. Aktive Datenbanksysteme sind also wegen ihrer begrenzten aktiven Funktionalität zur Zeit nur sehr beschränkt einsetzbar.

1.2 Problemstellung

An der Abteilung Information Engineering des Instituts für Wirtschaftsinformatik der Universität Bern wird ein Konzept für eine aktive Schicht für Datenbanksysteme entwickelt und implementiert. In dieser Schicht können Regeln definiert und verarbeitet werden. Sie trägt den Namen ALFRED (***A**ctive **L**ayer **F**or **R**ule **E**xecution in **D**atabase Systems*). ALFRED soll möglichst alle der oben genannten Nachteile aktiver DBMS beseitigen und berücksichtigt deshalb die meisten in der Literatur [2] angegebenen Merkmale aktiver DBMS. Dazu gehören verschiedene Regelstrukturen, eine umfangreiche Regelsemantik und vor allem diverse Werkzeuge zur Regeladministration und -analyse. Durch die gewählte Architektur kann mit ALFRED prinzipiell jedes (passive) DBMS in ein aktives überführt werden.

Im Rahmen dieses Projekts werden zwei Aufgaben daraus bearbeitet:

1. Entwurf und Implementierung einer Benutzerschnittstelle

Das Konzept von ALFRED sieht ein Menüsystem vor, das dem Benutzer ermöglicht, mit Datenbanken zu arbeiten. Es soll sich durch eine hohe Benutzungsfreundlichkeit auszeichnen und muss u.a. Funktionen bieten für:

- die Implementierung von Datenbankschemata
- die Manipulation von Daten
- die Definition von Regeln
- die Realisierung von Applikationen
- die Definition von (unternehmerischen) Prozessen
- die Simulation von Regeln

Diese Benutzeroberfläche soll möglichst selbsterklärend und leicht erlernbar sein. Sie soll den Benutzer durch die Möglichkeit, aus vordefinierten Werten zu selektieren, bei der Eingabe unterstützen. Die Benutzeroberfläche soll zudem robust und plattformunabhängig sein.

2. Ableitung von Regeln

Der Entwurf eines konzeptionellen Datenmodells wird auf der Grundlage von Entitäts- und Beziehungstypen durchgeführt. Dabei werden implizit Integritätsbedingungen (IB) angegeben. Bei der Realisierung eines Modells ist es erforderlich, solche Konsistenzen sicherzustellen. Bei der Definition von Objekten müssen dafür die entsprechenden (entitätstyp- und beziehungstypbezogenen) IB spezifiziert werden. Da in ALFRED nur Regeln definiert werden können, müssen die IB als solche dargestellt werden. Dabei muss sichergestellt sein, dass die Aufgaben und die Semantik der Regeln korrekt sind. Für die automatische Generierung dieser Regeln ist ein Konzept erforderlich.

1.3 Zielsetzung

Ziel dieses Informatikprojekts ist es, im Rahmen der Entwicklung des Prototypen für ALFRED, eine grafische Benutzerschnittstelle zu realisieren. Dafür sollen Anforderungen definiert werden. Dabei müssen bereits bestehende Konzepte und Anforderungen berücksichtigt werden. Anschliessend soll das Menüsystem entworfen und implementiert werden. Ebenso soll für die automatische Ableitung von Regeln bei der Objektdefinition die Anforderungen festgelegt und ein Konzept erarbeitet werden.

1.4 Gliederung

Der Aufbau dieser Arbeit ist wie folgt: Das erste Kapitel gibt einen Überblick über dieses Informatikprojekt. Im zweiten werden die Grundlagen für aktive Datenbanken und Regeln erläutert. Das Kapitel 3 beschreibt die Konzepte von ALFRED. Im vierten werden die Anforderungen an das Benutzersystem und die automatische Regelableitung festgelegt. Das Kapitel 5 zeigt das Vorgehen bei der Entwicklung des Menüsystems sowie als Beispiele einige der implementierten Masken. Im Kapitel 6 werden das Vorgehen und die Ergebnisse der Regelableitung beschrieben. Die Arbeit endet mit Kapitel 7, in dem eine kurze Zusammenfassung und ein Ausblick gegeben werden.

2 Aktive Datenbanksysteme

Datenbanksysteme bilden heute ein unverzichtbares Werkzeug für alle Unternehmungen aus praktisch allen Branchen. Vor allem Anwendungen in der *Verwaltung* (z.B. *Datenverarbeitung* oder *Informationssysteme*) aber auch in der *Industrie* (z.B. *Computer Integrated Manufacturing* oder *Computer Aided Design*) sind ohne Datenbanksysteme kaum noch denkbar.

Viele dieser Anwendungen erfordern, dass die eingesetzten Systeme in bestimmten Situationen automatisch reagieren können. Da herkömmliche Datenbanksysteme *passiv* sind, muss dieses Verhalten bei ihrer Verwendung auf Applikationsebene realisiert werden. Dies kann prinzipiell auf zwei Arten erfolgen:

- **Integriert in Applikationen**

  In jeder Applikation, in der auf die Datenbank zugegriffen wird, müssen Situationen erkannt werden, damit die notwendigen Aktionen ausgeführt werden können.

  Dies kann zur Verletzung des *Prinzips der Modularität* führen. Dies ist eine grosse potentielle Fehlerquelle, wenn redundant vorhandene Regeln, manipuliert werden müssen. Inkonsistentes Verhalten und eventuell auch inkonsistente Daten können die Folge sein.

- **Polling**

  Die Datenbank wird in regelmässigen Abständen von einer speziellen Applikation darauf untersucht, ob bestimmte Situationen eingetreten sind.

  Eine Schwierigkeit dieses Ansatzes ist, eine adäquate Polling-Frequenz festzulegen. Ist diese zu hoch, kann die Performance des ganzen Systems darunter leiden. Ist diese zu niedrig, kann auf bestimmte Situationen nicht zeitgerecht reagiert werden.

Diese gravierenden Nachteile in passiven Datenbanksystemen haben zu der Erkenntnis geführt, dass neue Konzepte für die Realisierung von aktivem Verhalten in DBMS benötigt werden. Diese Konzepte definieren *aktive Datenbanksysteme* (ADBMS). Darin wird das Verhalten durch Situations-Aktions-Regeln definiert.

2.1 Merkmale

Aktive Datenbanksysteme erweitern herkömmliche (passive) DBMS um Mechanismen zur Realisierung von aktivem Verhalten. Diese Erweiterung der Funktionalität bedeutet, dass ADBMS einige wichtige Anforderungen erfüllen müssen [2]. Ein erstes wichtiges Merkmal aktiver DBMS ist, dass sie selbst Datenbanksysteme sind. Dies bedeutet, dass alle Konzepte passiver DBMS auch in aktiven gültig sein müssen. Zweitens muss ein ADBMS die Definition und Verwaltung von Regeln ermöglichen. Eine weitere Eigenschaft aktiver Datenbanksysteme ist ein Verarbeitungsmodell mit einer wohldefinierten Semantik, das erlaubt, Ereignisse zu erkennen, Bedingungen auszuwerten und Aktionen auszuführen.
Daneben werden auch eine Reihe von optionalen Anforderungen an ADBMS gestellt. Ein Beispiel dafür ist eine integrierte Entwicklungsumgebung mit Werkzeugen, z.B. zum Durchsuchen, Entwerfen und Analysieren der Regelmenge.

2.2 Regelparadigma

Aktives Verhalten wird in ADBMS durch Situations-Aktions-Paare festgelegt. Situationen beziehen sich auf die Datenbank und werden durch Ereignisse und Bedingungen beschrieben. Zusammen mit der in einer Situation auszuführenden Aktion gelangt man so zu der **E**vent-**C**ondition-**A**ction(*ECA*)-Notation [6].

Das Eintreten des Ereignisses löst die Regel aus. Dies kann ein spezieller Zeitpunkt, ein Messwert oder eine Benutzereingabe sein. Anschliessend wird durch die Bedingung überprüft, ob die Datenbank einen bestimmten Zustand hat (z.B. ob ein Tupel in einer Relation vorhanden ist). Die Aktion (z.B. Einfügen eines Tupels) wird ausgeführt, wenn die Regel ausgelöst und die Bedingung erfüllt ist.

2.3 Regelspezifikation

Das aktive Verhalten eines ADBMS wird durch die *Regeldefinitionssprache* definiert. Darin werden die Strukturen, Inhalte und die Ausführungssemantik der Regeln beschrieben [10].

2.3.1 Regelstruktur

In ADBMS können Regeln verschiedene Strukturen wie z.B. ECA, ECAA, CA und CAA haben. Bei einer Struktur mit zwei Aktionen wird in Anlehnung an *if-then-else* auf beide Ausgänge der Bedingungsprüfung reagiert. Die einzelnen Komponenten (Ereignisse, Bedingungen und Aktionen) müssen klar voneinander abgegrenzt werden:

- **Ereignisse**

  Das Ereignis legt fest, wodurch eine Regel ausgelöst wird. Es wird zwischen primitiven und komplexen Ereignissen unterschieden. Primitive Ereignisse sind zum Beispiel

  1. Datenmanipulationen (z.B. `insert`, `update`, `delete` auf einer Tabelle)
  2. Datenselektionen (z.B. `select` auf einer Tabelle)
  3. Absolute Zeitereignisse (z.B. `12/02/97 at 10:00:00`).

  Komplexe Ereignisse sind Kombinationen von primitiven und/oder anderen komplexen Ereignissen, die durch Ereignisoperatoren kombiniert werden. Beispiele für Ereignisoperatoren sind

  1. Bool sche Operatoren (z.B. `and` und `or`)
  2. Verögerungsoperatoren (z.B. `5 seconds after event` E_1)
  3. Sequenzoperatoren (Festlegung einer bestimmten Reihenfolge, in der zwei oder mehrere Ereignisse eintreten müssen).

- **Bedingungen**

  Auch hier wird zwischen primitiven und komplexen Bedingungen unterschieden. Beispiele für primitive Bedingungen sind:

  1. die bool schen Werte `TRUE` und `FALSE`
  2. Prädikate (z.B. `Mitarbeiter.Name = 'Meier'`)
  3. Abfragen (z.B. `retrieve * from Mitarbeiter where Name = 'Meier'`).

  Komplexe Bedingungen werden aus primitiven und/oder anderen komplexen Bedingungen mit den bool schen Operatoren AND, OR und NOT zusammengesetzt.

- **Aktionen**

  Auch Aktionen können primitiv und komplex sein. Beispiele für primitive Aktionen sind:

  1. Datenmanipulationen (z.B. `delete from Mitarbeiter where Name = 'Meier'`)
  2. Ausgabe von Meldungen (z.B. `message (Datensatz existiert bereits!)`).

  Komplexe Aktionen sind Verkettungen von primitiven Aktionen.

2.3.2 Regelausführung

Jede Regeldefinitionssprache muss Konstrukte unterstützen, mit denen die Regelausführung detailliert festgelegt werden kann:

- **Ausführungszeitpunkte**

  Bei der Regelausführung wird für primitive Ereignisse zwischen den zwei Zeitpunkten *pre* und *post* unterschieden. Regeln, die mit pre definiert sind, werden vor der eigentlichen Befehlsverarbeitung ausgelöst (z.B. für eine Kontrolle der Zugriffsrechte). Regeln, die mit post definiert sind, werden nach der eigentlichen Befehlsverarbeitung ausgelöst (z.B. für Prüfungen von komplexen Integritätsbedingungen).

- **Granularitäten**

  Bei der Ausführung von Befehlen wird häufig auf mehrere Datensätze zugegriffen. Manchmal soll aber eine Regel auf jeden selektierten Datensatz angewendet werden. Deshalb wird zwischen *instanz-* und *mengenorientierter* Regelausführung unterschieden. Instanzorientierte Regeln, werden für jeden involvierten Datensatz einmal ausgeführt. Mengenorientierte Regeln hingegen werden genau einmal für den Befehl, ohne Berücksichtigung der Anzahl betroffener Datensätze, ausgeführt.

- **Prioritäten**

  Durch ein Ereignis können mehrere Regeln ausgelöst werden. Mit Prioritäten kann angegeben werden in welcher Reihenfolge diese Regeln ausgeführt werden sollen. Es werden zwei Typen von Prioritäten unterschieden:

  - **Partielle Prioritäten**
    Die Priorität einer Regel wird relativ zu einer anderen angegeben (höher oder niedriger).
  - **Totale Prioritäten**
    Die Priorität einer Regel wird absolut angegeben (z.B. durch Zahlenwerte).

- **Reihenfolgen**

  Werden durch ein einziges Ereignis mehrere Regeln mit gleicher Priorität ausgelöst, können diese *sequentiell* oder *parallel* ausgeführt werden. Bei der sequentiellen Ausführung wird die Reihenfolge entweder zufällig oder dynamisch (z.B. unter Berücksichtigung von Datensperren) bestimmt. Bei der parallelen Verarbeitung von Regeln muss sichergestellt werden, dass diese nicht in Konflikt zueinander stehen.

- **Kopplungsmodi**

  Kopplungsmodi legen die Regelausführung in Bezug auf Transaktionen fest. Diese können sowohl zwischen Ereignissen und Bedingungen (EC), wie auch zwischen Bedingungen und Aktionen (CA) angegeben werden. Ein EC-Kopplungsmodus legt fest, wann die Bedingungsauswertung in Abhängigkeit von der Regelauslösung verarbeitet werden soll.

Es werden verschiedene Kopplungsmodi unterschieden, deren Bedeutung am Beispiel der EC-Kopplungsmodi erläutert werden. Analoges gilt für die CA-Modi.

- **Immediate**
  Die Bedingungsauswertung erfolgt unmittelbar nach dem Eintritt des Ereignisses.
- **Deferred**
  Die Bedingungsauswertung erfolgt in der gleichen Transaktion nach dem letzten Befehl, jedoch vor dem Commit .
- **Detached independent**
  Die Bedingung wird in einer separaten Transaktion, die von der auslösenden vollständig unabhängig ist, ausgewertet. Dies bedeutet, dass die Bedingungsauswertung auch dann durchgeführt wird, wenn die auslösende Transaktion z.B. aufgrund eines Fehlers abbricht.
- **Detached Causally dependent**
  Die Bedingung wird ebenfalls in einer separaten Transaktion ausgewertet. Es bestehen jedoch im Gegensatz zum Kopplungsmodus detached independent Abhängigkeiten zwischen den Transaktionen. Es werden die drei Arten sequential , parallel und exclusive unterschieden [1].

2.4 Architekturen

In der Literatur werden unterschiedliche Architekturen für ADBMS vorgeschlagen. Drei der wichtigsten sind:

- **Schichten (layered architecture)**

  Bei diesem Ansatz wird zwischen die Applikationen und das DBMS eine Schicht eingefügt, die das gesamte aktive Verhalten beinhaltet. Dadurch, dass sämtliche Benutzereingaben an das DBMS und sämtliche Daten vom DBMS an den Benutzer durch die aktive Schicht hindurch gereicht werden, sind dort sämtliche für die Regelausführung notwendigen Informationen vorhanden.

  Diese Architektur hat den Vorteil, dass im Prinzip jedes beliebige (passive) DBMS in ein ADBMS umgewandelt werden kann. Dieser Architekturansatz hat aber auch einige Nachteile. Einer der wichtigsten ist sicher der hohe Kommunikationsaufwand zwischen aktiver Schicht und DBMS, wodurch eine geringe Performance resultieren kann. Auch ist es nicht möglich, mit wichtigen Subsystemen (z.B. Transaktionsmanager) des DBMS zu kommunizieren. Dies kann dazu führen, dass einige Anforderungen, wie zum Beispiel Kopplungsmodi, nicht realisiert werden können.

- **Integriert (built-in architecture)**

  Bei dieser Architektur sind die aktiven Komponenten Bestandteil des DBMS. Dies bedeutet, dass bestehende DBMS weiter- oder aber ganz neu entwickelt werden müssen.

  Die Vorteile einer solchen Architektur sind die problemlose Verwendung von Subsystemen des DBMS sowie die Möglichkeit auch weitere Eigenschaften der Regelausführung, wie zum Beispiel Kopplungsmodi, zu realisieren. Dem stehen die Nachteile von enormen Entwicklungsaufwänden für ein quasi neues DBMS gegenüber sowie die Tatsache, dass nur ein bestimmtes DBMS in ein ADBMS umgewandelt wird.

- **Übersetzt (compiled architecture)**

  Bei diesem Ansatz wird jede Benutzeraktion zur Laufzeit übersetzt (compiliert) und mit zusätzlichem Regel-Code versehen.

  Der Vorteil dieser Architektur ist, dass zur Laufzeit keine Ereigniserkennung und keine Regelverarbeitung stattfinden. Als Nachteile sind zu nennen, dass keine komplexen Ereignisse und Zeitereignisse möglich sind, da der Compiler alle Ereignisse erkennen muss. Auch müssen zyklischen Regelkaskaden verhindert werden, da der (rekursive) Übersetzungsvorgang sonst nicht abbrechen würde. Die Übersetzung zur Laufzeit kann sich zudem nachteilig auf die Performance auswirken.

2.5 Anwendungsgebiete

Regeln können für die unterschiedlichsten Aufgaben eingesetzt werden. Einige Beispiele sind:

- **Integritätsbedingungen**
  In herkömmlichen DBMS können die Überprüfungen der Integritätsbedingungen nur nach Datenbankoperationen oder Transaktionen erfolgen. In ADBMS können diese Überprüfungen im Prinzip zu beliebigen Zeitpunkten erfolgen. Dies ergibt eine viel grössere Flexibilität. Zusätzlich können komplexere Integritätsbedingungen realisiert werden.
- **Trigger**
  Ein Datenbanktrigger ist eine Prozedur, die automatisch ausgeführt wird, wenn Datenmanipulationen auf einer Datenbank durchgeführt werden. Trigger erweitern die Integritätsbedingungen durch eine wesentlich bessere Semantik. Sie können durch Regeln substituiert werden.
- **Autorisation**
  Regeln können Zugriffsrechte sicherstellen. Ein DBMS benötigt also kein spezielles Subsystem für die Autorisation der Benutzer.
- **Kontrolle und Steuerung**
  ADMBS können für die Kontrolle und Steuerung eingesetzt werden, wie z.B. für Workflow-Systeme und zur Inventarkontrolle.

2.6 Überblick

In der Literatur lassen sich viele Beschreibungen von aktiven Datenbanksystemen finden. Diese können grob in die drei Kategorien *relationale Prototypen*, *objektorientierte Prototypen* und *kommerzielle Systeme* eingeteilt werden. Die Prototypen relationaler aktiver Datenbanksysteme basieren auf bestehenden passiven relationalen DBMS. Beispiele dafür sind Ariel und POSTGRES. Bei den Prototypen objektorientierter aktiver DBMS wie z.B. HiPAC, REACH und SAMOS wurde von objektorientierten DBMS ausgegangen. Bei den kommerziellen ADBMS handelt es sich um bereits verfügbare aktive Datenbanksysteme, die um Mechanismen und Konzepte für die Realisierung von aktivem Verhalten erweitert wurden. Beispiele sind Ingres, Oraccle und Sybase. Die folgende Abbildung 2.1 gibt einen Überblick über die wichtigsten Projekte dieser drei Kategorien.

Aktives Verhalten wird auch in den Standards SQL-2 und SQL-3 berücksichtigt. Dies beschränkt sich bei SQL-2 jedoch auf Integritätsbedingungen. Erst im zur Zeit noch nicht veröffentlichten Standard SQL-3 sind auch Trigger vorgesehen.

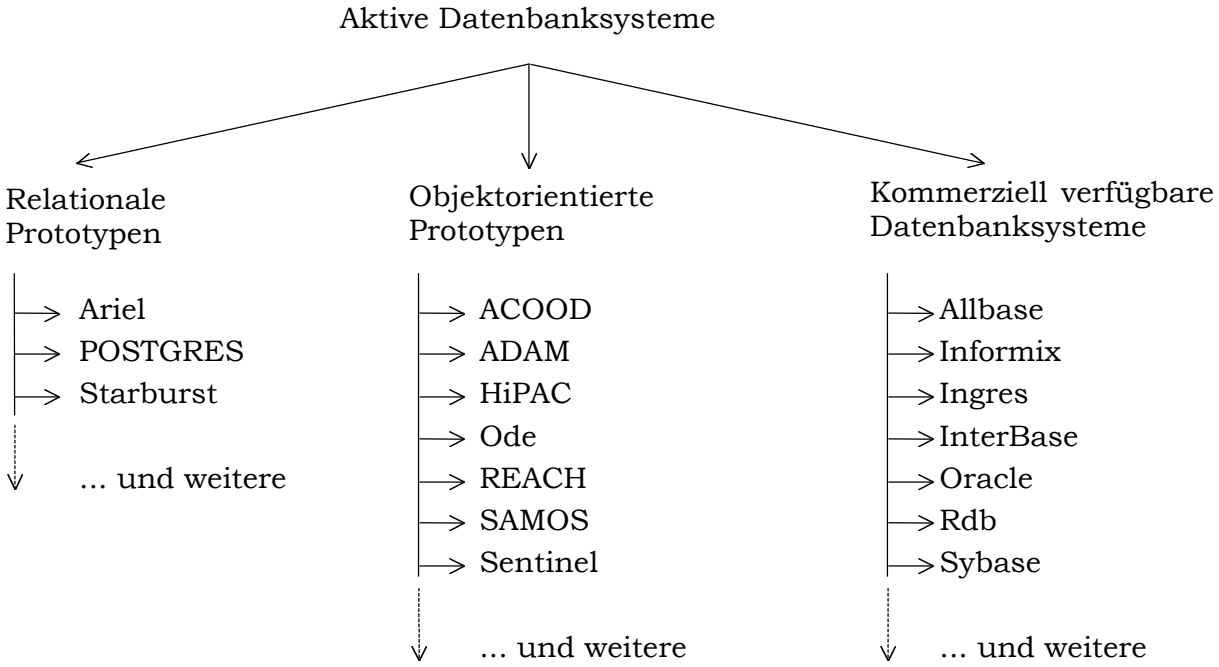

Abbildung 2.1: Überblick über Prototypen und Systeme aktiver Datenbanken

3 ALFRED

ALFRED ist ein Projekt, das am Institut für Wirtschaftsinformatik der Universität Bern seit 1995 bearbeitet wird. Die in diesem Rahmen durchgeführten Arbeiten [5] werden vor allem durch ein Nationalfond-Projekt beeinflusst und motiviert, dessen Ziel eine Untersuchung der Anwendung des Trigger-Konzepts in Bezug auf die Realisierung von Geschäftsregeln ist [4].

3.1 Anforderungen

Neben allgemeinen Anforderungen (wie z.B. *modularer Aufbau*, Kommunikation zwischen den einzelnen Komponenten über *definierte Schnittstellen*, leichte *Erweiter-* und *Wartbarkeit*) werden hier einige wesentliche Anforderungen erläutert, die von ALFRED erfüllt werden müssen. Diese können in folgende Kategorien eingeteilt werden:

- **Benutzer**
  Das Benutzersystem soll, den heutigen Standards entsprechend, *grafikorientiert* sein. Es soll durch möglichst grosse *Selbsterklärbarkeit* eine *leichte Erlernbarkeit* gewährleisten. Eine grosse *Robustheit* soll auch bei Fehlmanipulationen durch den Benutzer konsistente Datenbankzustände und ein stabiles System gewährleisten. Fehlerhafte Eingaben sollen in jedem Fall zu *Fehlermeldungen* führen. Das System soll sich jederzeit so verhalten, wie es der Benutzer erwartet. Eine *Online-Hilfe* soll den Benutzer bei allen Arbeiten in ALFRED unterstützen. Das System soll *Werkzeuge* zur Verfügung stellen, die den Benutzer bei komplexen Aufgaben auf *anschauliche* Weise anleiten. Dies bedeutet insbesondere, dass die Masken nicht zu viel Information auf einmal anzeigen dürfen.
- **Umgebung**
  ALFRED soll auf allen gängigen *Plattformen* (z.B. Microsoft Windows, Unix, usw.) eingesetzt und auf beliebige *Datenbanksysteme* (z.B. Oracle, Ingres, usw.) aufgesetzt werden können.
- **Regelsprache**
  ALFRED soll eine mächtige Regeldefinitionssprache (RDL) unterstützen. Es sollen *sprachliche, primitive und komplexe Komponenten*, mit denen beliebige Geschäftsregeln abgebildet werden können, unterstützt werden.
- **Regelanalyse**
  ALFRED muss gewährleisten, dass die Regelmenge jederzeit *konsistent* ist. *Konflikte* zwischen Regeln müssen verhindert werden. *Nichtterminierende Zyklen* in den Regeln müssen erkannt und verhindert werden. Die *Konfluenz*, das heisst die Unabhängigkeit der Reihenfolge der Regelausführung, wenn ein Ereignis mehrere Regeln auslöst, muss gewährleistet sein.
- **Modell**
  Für Befehle und Ausführungssemantik soll ein *einheitliches Modell*, das eine ganzheitliche Modellierung der Regeln erlaubt, verwendet werden. Dieses soll sowohl für die *Verarbeitung* als auch für *Analyse* und *Simulation* verwendet werden können.

3.2 Architektur

Um alle an ALFRED gestellten Anforderungen abdecken zu können, wurde eine *Schichtenarchitektur* gewählt. Die Abbildung 3.1 zeigt das Modell von ALFRED mit seinen zwei Subsystemen *User-System* und *Processing-System* sowie die Verbindung zu Datenbanksystemen.

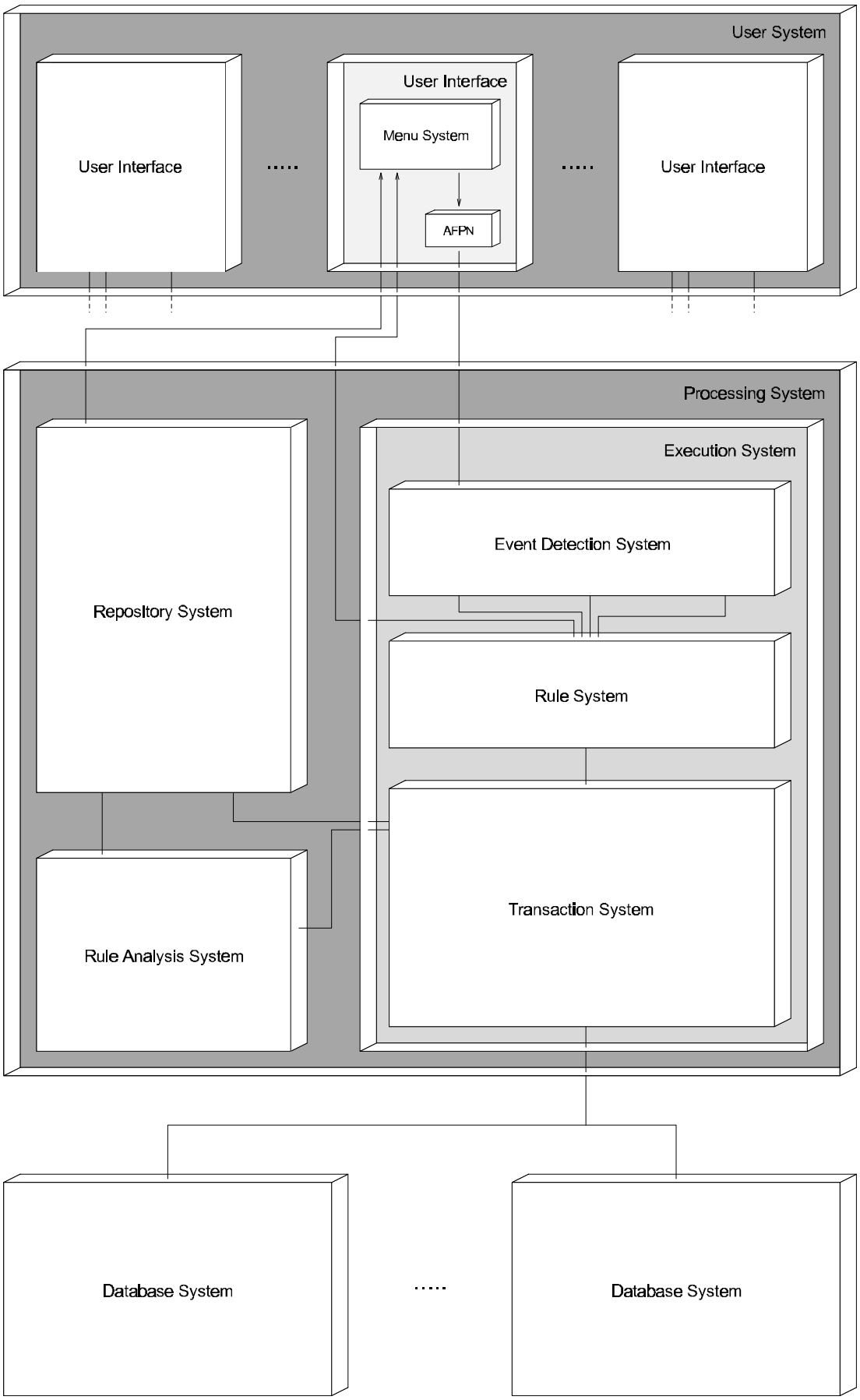

Abbildung 3.1: Architektur von ALFRED

3.2.1 Benutzersystem

Das Benutzersystem beinhaltet zum einen den für den Benutzer sichtbaren Teil von ALFRED, das *Menu-System,* und das *AFPN*, welches aus den Benutzereingaben ein erstes Ablaufmodell in Form eines einfachen Petri-Netzes erstellt.

- **Menu-System**
  Das Menüsystem stellt dem Benutzer eine Oberfläche zur Verfügung, über die er die Funktionalitäten von ALFRED, wie Manipulation von Datenbanken, Objekten und Regeln, einfach nutzen kann. Es hat die Kontrolle über die aktive Schicht und meldet dem Benutzer, ob seine Befehle erfolgreich ausgeführt werden konnten oder nicht.
- **AFPN**
  Im Modul AFPN erfolgt die Transformation der Benutzereingaben in eine vom *Verarbeitungssystem* verständliche Form, die *Action Flow Petri Netze* (AFPN) [5].

3.2.2 Verarbeitungssystem

Im Verarbeitungssystem (Processing System) werden Befehle und Regeln analysiert, modelliert und verarbeitet. Es besteht aus folgenden Subsystemen:

- **Repository**
  Im Repository werden alle Informationen über definierte Regeln, Objekte, Benutzer usw. gespeichert. Diese werden für die Analyse und für Auswertungen verwendet.
- **Regelanalysesystem**
  Im Regelanalysesystem (Rule Analysis System) werden die Regeln, die durch den Benutzer definiert wurden, erzeugt und auf gewisse Eigenschaften wie z.B. Zyklen, Redundanzen und Konflikte hin untersucht.
- **Ausführungsystem**
  Im Ausführungssystem (ExecutionSystem) erfolgt die eigentliche Verarbeitung der Regeln. Es besteht aus drei Modulen, dem *Ereigniserkennungssystem*, dem *Regelsystem* und dem *Transaktionssystem*.

  1. *Ereigniserkennungssystem*

     Im Ereigniserkennungssystem (Event Detection System) werden die definierten Ereignisse erkannt. Es wird unterschieden zwischen *primitiven*, *komplexen* und *temporalen Ereignissen*. Für jeden dieser Ereignistypen gibt es ein eigenes Subsystem.

  2. *Regelsystem*

     Im Regelsystem (Rule System) werden die vom Benutzersystem erzeugten AFPN um auszuführende Regeln, die bei der Verarbeitung ausgelöst werden, erweitert. So entstehen aus den einfachen AFPN die komplexeren ***A**ction **R**ule **F**low **P**etri **N**etze* (ARFPN). Diese werden anschliessend verarbeitet. Dabei müssen Zyklen, die bei der statischen Analyse gefunden wurden, überwacht, und wenn sie nicht terminieren, auf eine bestimmte Art und Weise behandelt werden.

  3. *Transaktionssystem*

     Im Transaktionssystem (Transaction System) werden Bedingungen ausgewertet und Befehle verarbeitet. Ausserdem werden Datenmanipulationsaktionen (insert, update, delete und select) an das unter ALFRED liegende DBMS zur Ausführung weitergegeben.

3.3 Regeldefinitionssprache

Das Spektrum jedes aktiven Verhaltens wird durch die *Regeldefinitionssprache* bestimmt. Diese gibt an, welche Arten von Ereignissen, Bedingungen und Aktionen verwendet werden können. Zusätzlich kann die Regelsemantik festgelegt werden.

Die **A**LFRED **R**ule **D**efinition **L**anguage (*ARDL*) besteht aus zwei Teilen. Mit dem Strukturteil werden die Struktur und die einzelnen Komponenteninhalte definiert. Mit dem Semantikteil wird die Verarbeitung von Regeln festgelegt.

3.3.1 Regelstruktur

Die ARDL unterstützt die Definition von drei *ereignisbasierten* (ECAA, ECA, EA) und zwei *pattern-basierten* (CAA, CA) Regelstrukturen. Jede Regelkomponente kann *sprachlich*, *primitiv* oder *komplex* sein.

Eine ereignisbasierte Regel wird ausgelöst, wenn das entsprechende einfache oder komplexe Ereignis entdeckt wird. Primitive Ereignisse sind Datenbankereignisse (z.B. `create database`), Datendefinitionsereignisse (z.B. `create rule`), Datenmanipulationsereignisse (z.B. `insert`), absolute Zeitereignisse (z.B. `12:00, 10. April, 1997`) oder abstrakte Ereignisse (z.B. `Bestellung ist eingetroffen`). Komplexe Ereignisse bestehen aus primitiven und/oder komplexen Ereignissen, die durch Ereignisoperatoren verknüpft werden. Die ARDL unterstützt die Ereignisoperatoren *Boolean-*, *Auswahl-*, *Sequenz-*, *Wiederholungs-*, *Intervall-* sowie *Zeitoperatoren*.

Bedingungen legen fest, was überprüft werden soll. Beispiele für primitive Bedingungen sind *true*, *false*, *Prädikate* (z.B. `Mitarbeiter.Name = 'Meier'`) und *Abfragen* (z.B. `retrieve * from Mitarbeiter where Name = 'Meier'`). Komplexe Bedingungen bestehen aus primitiven und/oder komplexen Bedingungen, die mit den boole schen Operatoren *and*, *or* und *not* kombiniert werden. Ebenfalls zu den komplexen Bedingungen gehören boole sche Funktionen, die ein *true* oder *false* als Resultat zurückliefern.

Die Aktionskomponente einer Regel legt die Reaktion auf eine bestimmte Situation fest. Unterstützt werden die primitiven Aktionen *Datenmanipulationsaktionen* (z.B. `insert`, `update`, `delete`), *Meldungen* (z.B. `message to all: Server will be down from 11:00 to 15:00!`) und *abstrakte Aktionen* (z.B. `raise 'Bestellung ist eingetroffen'`), die verwendet werden können, um abstrakte Ereignisse auszulösen. Komplexe Aktionen sind *Sequenzen* von primitiven und anderen komplexen Aktionen, wie *Transaktionen*, *Prozeduren* und *Applikationen*.

3.3.2 Regelsemantik

Der Semantikteil der ARDL besteht aus einer Anzahl von Anweisungen, welche die Regelverarbeitung bestimmen und bei der Regeldefinition festgelegt werden. Dazu gehören die *Auslösungszeitpunkte* pre und post, *Meilensteine* (`<rule_timing>`) und *Alternativpläne* (`<contingency_plan>`), falls die Meilensteine nicht eingehalten werden können. Eine weitere Eigenschaft sind *Prioritäten* (`<rule_order>`), welche global sind und als ganzzahliger Wert angegeben werden. Regeln, welche gleichzeitig ausgelöst werden und die selbe Priorität haben, werden gleichzeitig ausgeführt, wenn es keine Konflikte zwischen ihnen gibt. Sonst bestimmt das System die Ausführungsreihenfolge. Schliesslich unterstützt die ARDL die in Kapitel 2.3.2 beschriebenen sechs *Kopplungsmodi* (`<rule_coupling>`), *Granularitäten* und *Parameter-Kontexte*.

Eine vereinfachte Syntax von ECAA-Regeln ist in der Abbildung 3.2 angegeben:

```
<rule> ::=   ‘rule’ <ident>
             [‘belongs’ ‘to’                      <rule_group>]
             [‘has’ ‘priority’                    <rule_order>]
             ‘is’                                 <rule_state>
             [‘has’ ‘to’ ‘be’ ‘executed’ ‘until’  <rule_timing>
             [‘with’ ‘contingency’ ‘plan’         <rule_cont_plan>]]

             ‘on’ ‘event’                         <event>
             ‘check’ ‘condition’                  <condition>
             ‘execute’ ‘true_action’              <action>
             ‘or’ ‘false_action’                  <action>
             [‘with’ ‘couplings’ ‘:’
              ‘event-condition’         ‘:’       <rule_coupling>
              ‘condition-true_action’   ‘:’       <rule_coupling>
              ‘condition-false_action’ ‘:’       <rule_coupling>]
```

Abbildung 3.2: Syntax für eine ECAA-Regel

4 Anforderungen an die Benutzerschnittstelle

An das Benutzersystem von ALFRED werden verschiedene Anforderungen gestellt. Dabei kann zwischen allgemeinen Anforderungen an Benutzerschnittstellen, Menüfunktionalitäten und Anforderungen für die automatische Generierung von Regeln unterschieden werden. Die allgemeinen Anforderungen legen wichtige Eigenschaften von (grafischen) Benutzerschnittstellen fest. Die Menüfunktionalitäten zeigen auf, welche Funktionen benötigt werden, um die in Kapitel 3.1 geforderten Benutzer- und Umgebungseigenschaften von ALFRED zu erfüllen. Die Anforderungen an die Regelableitung legen fest, welche Regeln mit welchen Eigenschaften automatisch generiert werden sollen.

4.1 Allgemeine Anforderungen

Als Richtlinie für die Anforderungen an das Benutzersystem von ALFRED werden die Regeln für den Dialogdesign von Shneiderman [9] verwendet:

1. **Konsistenz**
   Aus ähnlichen Situationen sollen ähnliche Aktionsfolgen resultieren. In Menüs, Hilfeinformationen usw. sollen identische Begriffe, Symbole und Zeichen verwendet werden.
2. **Informatives Feedback**
   Jede Aktion soll eine sichtbare Systemreaktion bewirken. Der Umfang des Feedbacks sollte sich an der Komplexität der Aktion orientieren.
3. **Abschluss von Dialogen**
   Aktionsfolgen sollten einen Beginn, eine Mitte und ein Ende besitzen. Der Benutzer ist erleichtert, wenn er eine solche Aufgabe komplett durchlaufen hat und kann sich auf die nächste Aufgabe konzentrieren.
4. **Fehlerbehandlung**
   Es sollte grundsätzlich nicht möglich sein, schwerwiegende Fehler zu begehen. Falls ein Fehler auftritt, sollte das System diesen erkennen, dem Benutzer anzeigen und eine möglichst einfache Fehlerbehandlung anbieten.
5. **Rücksetzmöglichkeit**
   Aktionen sollten zurückgenommen werden können. Dies gilt insbesondere für Aktionen, durch welche Daten verändert oder gelöscht werden. Dies nimmt dem Benutzer die Angst bei der Arbeit, da jederzeit zum Vorzustand zurückgekehrt werden kann.
6. **Benutzergesteuerter Dialog**
   Benutzer wollen den Dialog (und damit das System) im Griff haben. Dies kann durch unerwartete Systemreaktionen, lange Dateneingabesequenzen, Schwierigkeiten beim Abruf bestimmter Informationen oder der Auslösung gewünschter Aktionen beeinträchtigt werden. Diese und ähnliche Schwierigkeiten sollten dem Benutzer unbedingt erspart werden.
7. **Entlastung des Kurzzeitgedächtnisses**
   Dies lässt sich durch einfache Bildschirminhalte, sowie kontextsensitive Hilfen, Abkürzungen und Codes erreichen. Informationen, die bereits im System gespeichert sind, sollten dem Benutzer bei Bedarf in Form von Auswahllisten zur Verfügung gestellt werden.

4.2 Menüfunktionalitäten

ALFRED ist als aktive Schicht konzipiert, die auf prinzipiell jedes beliebige Datenbanksystem aufgesetzt werden kann. Die Schnittstelle wird durch die Datendefinitions- und Datenmanipulationsfunktionen des DBMS gebildet. Deshalb müssen im Menüsystem von ALFRED Funktionen vorgesehen werden, mit denen Daten definiert und manipuliert werden können. Ausserdem sollen auch die Funktionen zur Realisierung des aktiven Verhaltens von ALFRED durch die Definition von Regeln über die Menüs ausgeführt werden können. Die benötigten Funktionen lassen sich zu folgenden Kategorien zusammenfassen:

- **Datenbankfunktionen**
  Datenbanken sollen erzeugt und nötigenfalls auch wieder gelöscht werden können. Benutzer sollen sich bei der Datenbank, mit der sie arbeiten wollen, anmelden und am Schluss auch wieder abmelden können.
- **Datendefinitionsfunktionen**
  Die Benutzerverwaltung (d.h. Erzeugen, Löschen und Ändern von Benutzern) für ALFRED soll über Menüfunktionen durchgeführt werden können. Diese Benutzer ihrerseits sollen in ALFRED Teile der Realwelt (z.B. einer Unternehmung) als Informationssystem abbilden können. Dazu benötigen sie die Möglichkeit, Objekttypen (*objects*), abgeleitete Daten (*views*) und Regeln (*rules*) zu verwalten. Unter Objekttypen werden in ALFRED aus Gründen der Flexibilität sowohl Klassen (in objektorientierten DBMS) als auch Relationen (in relationalen DBMS) verstanden. Weiter sollen die Benutzer auch komplexere Funktionen und Applikationen erstellen können. Schliesslich müssen die Zugriffsrechte und Privilegien der Benutzer organisiert werden können.
- **Datenmanipulationsfunktionen**
  Diese Kategorie enthält Funktionen zum Verändern des Datenbestandes. Dies beinhaltet das Selektieren und das Mutieren von Datensätzen. Ebenfalls in diese Kategorie gehören Funktionen zum Verändern des Regelstatus (aktivieren und deaktivieren) und Funktionen zum Ausführen von Programm-Code.
- **Simulationen und Prozesse**
  Simulationen und Prozesse sollen jeweils erzeugt, geändert, gelöscht und ausgeführt werden können. Bei Prozessen soll zusätzlich die Möglichkeit bestehen, diese grafisch anzuzeigen.
- **Informationen zu Regeln**
  Diese Kategorie fasst Funktionen zum Auswerten der Regelmenge zusammen. Regeln sollen nach gewissen Aspekten gruppiert, analysiert und ausgegeben werden.
- **Hilfe**
  Zu jeder Funktion soll online eine Hilfe angezeigt werden können. Die dazu vorhandenen Funktionen sollen sich nach den Standards für Benutzerschnittstellen richten. Dazu gehören eine Übersicht der verfügbaren Hilfethemen, eine kontextsensitive Hilfe und Informationen zur Programmversion.

4.3 Ableitung von Regeln

Bei der Realisierung eines konzeptionellen Datenmodells werden Entitäts- und Beziehungstypen erzeugt. Zusätzlich müssen Integritätsbedingungen (IB) definiert werden, welche bei der konzeptionellen Datenmodellierung implizit und explizit festgelegt wurden.

In ALFRED sollen Regeln, welche die Aufgaben der Integritätsbedingungen sicherstellen, automatisch erzeugt werden. Dadurch, dass auch die IB als Regeln dargestellt werden, wird eine einheitliche Realisierung des aktiven Verhaltens erreicht.

Alle diese Integritätsbedingungen lassen sich in zwei Kategorien einteilen:

- **Entitätstypbezogene Integritätsbedingungen**

  Diese beziehen sich immer genau auf einen Entitätstyp. Es lassen sich die folgenden Typen unterscheiden:

  – *Not-Null-IB*
  Attribute, die als Not-Null definiert sind, müssen in jedem Fall einen Wert enthalten.

  – *Schlüssel-IB*
  Ein Schlüssel eines Entitätstypen ist eine Menge von Attributen, deren Werte jeden Datensatz *eindeutig* identifizieren. Für jeden Entitätstypen können mehrere Schlüssel definiert werden.

  – *Primärschlüssel-IB*
  Der Primärschlüssel ist der für den eindeutigen Zugriff auf die Entitäten des Entitätstypen massgebliche Schlüssel.

  – *Benutzerdefinierte IB*
  Mit benutzerdefinierten IB lassen sich bestimmte Eigenschaften für Attribute festlegen. Beispiele sind Minimal- und Maximalwerte. Aber auch Abhängigkeiten von Attributen untereinander können damit gewährleistet werden (z.B. `Von-Datum` ≤ `Bis-Datum`).

- **Beziehungstypbezogene Integritätsbedingungen**

  Diese umfassen die Gewährleistung der referentiellen Integrität und beziehen sich immer auf zwei Entitätstypen und ihre Beziehung zueinander. Bei der Verletzung der referentiellen Integrität aufgrund einer Operation sind grundsätzlich vier verschiedene (Re-) Aktionen möglich:

  – *CASCADE*
  Änderungen werden an die abhängigen Entitätstypen propagiert.

  – *SET NULL*
  Nicht mehr referenzierte Fremdschlüssel werden auf NULL gesetzt.

  – *SET DEFAULT*
  Nicht mehr referenzierte Fremdschlüssel werden auf einen Default-Wert gesetzt.

  – *NO ACTION* oder *RESTRICT*
  Änderungen, welche Integritätsbedingungen verletzen, werden verhindert.

Für die nachfolgenden Betrachtungen wird die heute übliche (min,max)-Notation [8] für Beziehungstypen verwendet. Dabei sind die vier folgenden Kardinalitätsangaben möglich:

- (0,1) choice
- (0,n) multiple choice
- (1,1)
- (1,n) multiple

Diese können für *binäre Beziehungstypen* alle miteinander kombiniert werden, so dass sich insgesamt 16 Möglichkeiten ergeben, zwei Relationen miteinander zu verknüpfen. Davon sind aus Symmetriegründen nur 10 verschieden voneinander. Diese lassen sich in drei Kategorien einteilen:

- Beziehungstypen mit einseitiger Existenzabhängigkeit
  Existenzabhängigkeit bedeutet, dass Entitäten des abhängigen Typs nur dann existieren können, wenn sie über eine Beziehung mit anderen verbunden sind. Bei einer *insert*-Operation in den abhängigen Entitätstyp muss also gewährleistet sein, dass die referenzierte Entität des anderen Typs bereits existiert. Ist dies nicht der Fall, kann die Operation entweder abgebrochen werden, oder der Benutzer wird aufgefordert, die benötigte Entität zu erstellen. Bei *update*-Operationen auf Entitäten des abhängigen Typs muss wie bei insert sichergestellt werden, dass alle neu referenzierten Entitäten existieren. Zusätzlich muss verhindert werden, dass Entitäten des abhängigen Typs existieren, die nicht mehr verbunden sind. Bei beiden Operationen ist zudem unter Umständen (abhängig vom Beziehungstyp) auch die Einhaltung der Maximal-Kardinalität zu überprüfen. Beim Löschen von Entitäten des nicht abhängigen Typs muss nur sichergestellt werden, dass nicht Entitäten des abhängigen Typs existieren, die mit der zu löschenden unabhängigen Entität verknüpft sind.
  Zu dieser Kategorie gehören alle Beziehungstypen mit einer einseitigen Minimal-Kardinalität von 1. Dies sind die Beziehungstypen (1,1)-(0,1), (1,1)-(0,n), (1,n)-(0,1) und (1,n)-(0,n).
  Welche der jeweils möglichen Aktionen zur Sicherstellung der referentiellen Integrität durchgeführt wird, hängt vom Beziehungstyp ab. In manchen Situationen sind mehrere möglich. In diesem Fall entscheidet der Benutzer, welche Aktion durchgeführt werden soll.
- Beziehungstypen mit gegenseitiger Existenzabhängigkeit
  Bei diesen Beziehungstypen müssen die oben beschriebenen Überprüfungen der Existenz jeweils für beide Entitätstypen durchgeführt werden. Zu dieser Kategorie gehören die Beziehungstypen mit gegenseitiger Minimal-Kardinalität 1. Dies sind (1,1)-(1,1), (1,n)-(1,1) und (1,n)-(1,n).
- Beziehungstypen ohne Existenzabhängigkeit
  Zu dieser Kategorie gehören die Beziehungstypen (0,1)-(0,1), (0,1)-(0,n) und (0,n)-(0,n). Bei Einfüge-Operationen muss jeweils nur die Einhaltung der Maximal-Kardinalität sichergestellt sein. Bei Änderungen werden zusätzlich wie beim Löschen nicht mehr referenzierte Fremdschlüssel auf NULL oder einen Standardwert gesetzt.

ALFRED soll alle diese Integritätsbedingungen unterstützen. Wichtig ist dabei, dass die generierten Regeln die gleiche Aufgabe wie die IB übernehmen und die gleiche Semantik haben. Zudem muss sichergestellt werden, dass die Struktur der erzeugten Regel von der ARDL auch unterstützt wird.

5 Menüsystem

5.1 Entwicklungswerkzeuge

In diesem Abschnitt wird das Vorgehen und die im Mittelpunkt stehenden Kriterien für die Wahl des Entwicklungswerkzeuges aufgezeigt. Dazu wird eine Auswahl von typischen Entwicklungsumgebungen kurz beschrieben. Anschliessend wird untersucht, welche Vor- und Nachteile die einzelnen Werkzeuge haben. Der Vergleich zeigt, dass die Benutzeroberfläche von ALFRED mit Tcl/Tk erstellt werden sollte.

5.1.1 Kriterien

Für die Wahl der Entwicklungswerkzeuge sind vor allem die folgenden vier Kriterien massgebend:

- **Portabilität (Plattformunabhängigkeit)**
  ALFRED soll mit möglichst wenig Änderungsaufwand sowohl auf Unix- wie auch auf Windows-basierten Systemen eingesetzt werden können. Dies gilt insbesondere auch für das Menüsystem.
- **Integrationsmöglichkeit in C/C++**
  Da das Verarbeitungssystem mit der Programmiersprache C/C++ realisiert werden soll, ist es wichtig, dass sich das Menüsystem problemlos damit verbinden lässt.
- **Verarbeitungsgeschwindigkeit**
  Das gewählte Entwicklungswerkzeug soll eine möglichst hohe Verarbeitungsgeschwindigkeit (vor allem später im Zusammenhang mit dem Verarbeitungssystem) gewährleisten.
- **Entwicklungsaufwand**
  Das Menüsystem muss sich mit möglichst geringem Aufwand erstellen lassen.

5.1.2 Werkzeuge

Für die Entwicklung von grafischen Benutzeroberflächen gibt es eine Reihe von unterschiedlichen Werkzeugen. Für die Wahl wurde eine Anzahl von herkömmlichen aber auch neueren Entwicklungstools (wie z.B. Java) betrachtet. Um den Rahmen dieser Arbeit nicht zu sprengen, werden hier aber nur die gebräuchlichsten aufgeführt.

- **C/C++**
  C ist eine universell einsetzbare Programmiersprache der 3. Generation, die sich Dank ihrer Mächtigkeit stark verbreitet hat. C ist insbesondere mit Unix-Systemen stark verbunden, da wesentliche Teile von Unix in dieser Sprache programmiert sind. C++ ist eine Erweiterung um objektorientierte Programmiertechniken. Beide Werkzeuge sind aus der heutigen Praxis der System- und Anwendungsentwicklung nicht mehr wegzudenken.
- **Tcl/Tk und Xf**
  Tcl/Tk wurde von John K. Ousterhout von der Computer Science Division des Department of Electrical Engineering and Computer Science der University of California in Berkeley entwickelt. Tcl (tool command language) ist eine einfache interpretierte Skript-Sprache zum Erzeugen von Applikationen. Tk (toolkit) ist eine Erweiterung zu Tcl mit allen benötigten Funktionalitäten zum Erstellen von X-Window-kompatiblen Benutzeroberflächen.

Tcl und Tk sind als C-Bibliotheken implementiert und können dadurch sehr einfach in C-Programme integriert werden. Der Programmierer kann auf einfache Weise die Funktionalität von Tcl erweitern und so die Anforderungen der Anwendung erfüllen.

Tcl/Tk ist auf allen gängigen Arbeitsplatzplattformen verfügbar, was natürlich die geforderte Offenheit von ALFRED stark unterstützt. Es kann lizenzfrei über das Internet bezogen werden (FTP-Site: ftp.neosoft.com/pub/tcl/new).

Für den Entwurf der Dialogfenster steht das Werkzeug Xf, eine Erweiterung zu Tcl/Tk zur Verfügung. Xf wurde von Sven Delmas an der technischen Universität Berlin entwickelt. Es handelt sich dabei um eine integrierte Entwicklungsumgebung, welche die interaktive Entwicklung von grafischen Benutzerschnittstellen ermöglicht. Xf nützt die Möglichkeiten von Tcl/Tk und erlaubt somit Konstruktion und Test einer Applikation. Xf ist ebenfalls frei über das Internet erhältlich (FTP-Site: ftp.neosoft.com/pub/tcl).

- **Borland Delphi**
  Die Entwicklungsumgebung Delphi basiert auf einer objektorientierten Erweiterung der Programmiersprache Pascal auf Windows-Plattformen. Sie besteht aus diversen Werkzeugen, die eine grosse Anzahl von Entwicklungsaktivitäten interaktiv unterstützen. Insbesondere kann die grafische Benutzeroberfläche vollständig interaktiv am Bildschirm erstellt werden. Die Anbindung von praktisch beliebigen Datenbanksystemen ist über die Schnittstelle ODBC[1] möglich.
- **Microsoft Access**
  Access ist ein Windows-basiertes relationales DBMS. Eine wichtige Eigenschaft ist die Möglichkeit über die Schnittstelle ODBC auf anderen DBMS zuzugreifen. Weiter beinhaltet Access ein integrierte Entwicklungsumgebung, mit der vollständige Applikationen erstellt werden können.

5.1.3 Vergleich und Entscheidung

Für einen Vergleich der Entwicklungsumgebungen sind diese im Hinblick auf die oben festgelegten Kriterien zu bewerten. Dabei bedeuten:

- \- Das Werkzeug unterstützt dieses Kriterium nicht oder kann nur mit hohem Aufwand realisiert werden.
- \+ Das Kriterium wird vom Werkzeug unterstützt, jedoch ist zusätzlicher Programmieraufwand nötig (Datenstrukturen / Algorithmen).
- ++ Das Werkzeug unterstützt die Forderung gut. Es ist nur ein geringer Programmieraufwand nötig.

[1] Unter ODBC (Open Database Connectivity) versteht man eine von der Firma Microsoft entwickelte Software-Schnittstelle, die den Zugriff aus Anwendungsprogrammen auf unterschiedliche Datenbanken gewährleisten soll.

Kriterium	C/C++	Tcl/Tk und Xf	Delphi	Access
Portabilität	+	++	-	-
C-Integration	++	++	+	+
Verarbeitungs-geschwindigkeit	++	+	++	+
Entwicklungs-aufwand	-	+	++	+

Tabelle 5.1: Tabellarischer Vergleich der Entwicklungswerkzeuge

Die Portabilität ist eine für ALFRED sehr zentrale Forderung. Aus diesem Grund kommen Werkzeuge, die diese nicht erfüllen, für die Entwicklung nicht in Frage. Dies betrifft die beiden windows-basierten Tools Borland Delphi und Microsoft Access. Für die Integration des Menüsystems in das Verarbeitungssystem muss eine Zusammenarbeit mit C-Funktionen möglich sein. Somit verbleiben nur die beiden Varianten mit C resp. C++. Weil eine Entwicklung der Benutzeroberfläche ohne spezielles Tool nicht mit vernünftigem Aufwand machbar ist, kommt eine reine C/C++-Lösung ebensowenig in Frage.

Daher ist entschieden worden, das Menüsystems Tcl/Tk und Xf auf einem Linux-System zu entwickeln. Es werden die folgenden Programmversionen benützt:

Tcl V. 7.6
Tk V. 4.2
Xf V. 3.1

5.2 Design

Das Menüsystem von ALFRED muss den heutigen Anforderungen an ein modernes Front-End entsprechen. Das heisst, es muss grafisch sein und auf der Fenster-Technik basieren. Eingaben sollen weitgehend mit der Maus erfolgen. Dazu sollen dem Benutzer (überall wo dies möglich und sinnvoll ist) Einträge in Listen zur Auswahl zur Verfügung gestellt werden.

5.2.1 Menüstruktur

Ausgehend von den in Kapitel 4.2 beschriebenen funktionellen Anforderungen an das Menüsystem wurde die in Abbildung 5.1 dargestellte Menüstruktur abgeleitet. Dabei wurde versucht, die logische Zusammengehörigkeit der Funktionen so darzustellen, dass der Benutzer einfach den gesuchten Menüpunkt findet. Die Abbildung 5.1 zeigt schematisch das Ergebnis des Aufbaus der gesamten Menüstruktur.

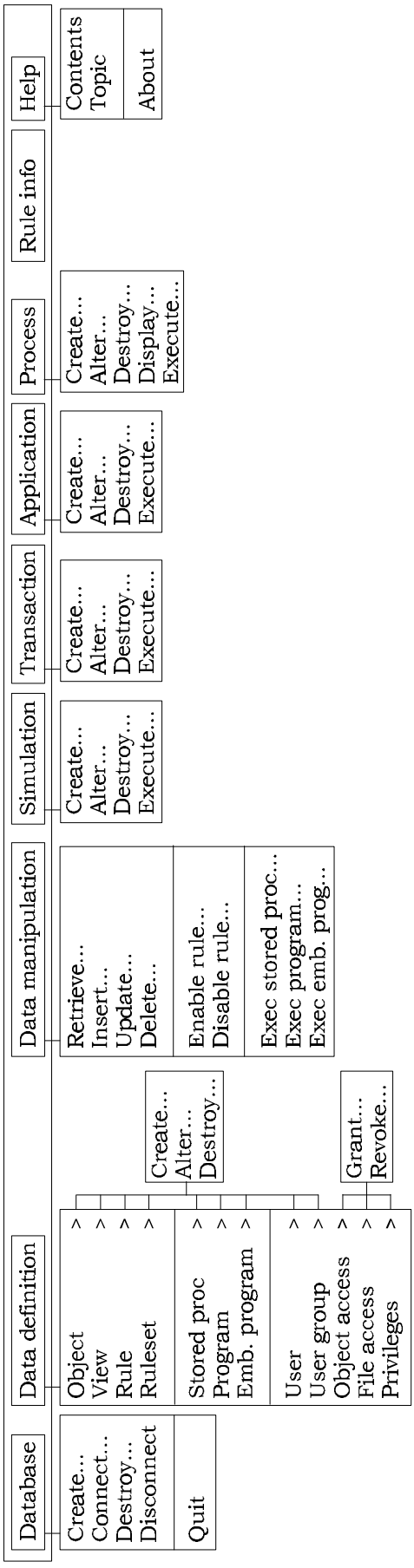

Abbildung 5.1: Das Menüsystem von ALFRED

Das in Abbildung 5.2 gezeigte Untermenü Database umfasst alle Funktionen zur Verwaltung von Datenbanken.

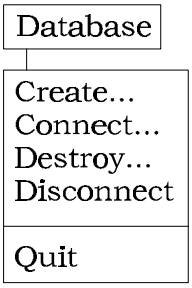

Abbildung 5.2: Submenü "Database"

Das Submenü Data definition umfasst alle Funktionen, mit denen Datenbankelemente verwaltet werden können. Dieses wurde durch Trennlinien weiter in die Bereiche Datenschemata, Programme und Benutzeradministration aufgeteilt. Die effektiven Menübefehle wurden pro Kategorie in einem weiteren Untermenü zusammengefasst (vgl. Abb. 5.3).

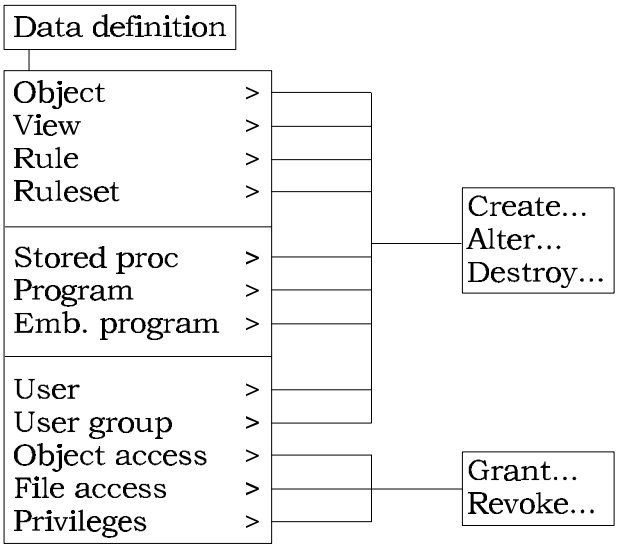

Abbildung 5.3: Submenü "Data definition"

Das dritte Submenü Data manipulation fasst die Funktionen zur Datenmanipulation zusammen (vgl. Abb. 5.4). Diese wurden in die drei Bereiche Datensätze, Regeln und Ausführung von Programmcode eingeteilt. Der Bereich Datensätze enthält die Funktionen, welche direkt Datensätze in Objekten ansprechen. Dies sind *retrieve*, *insert*, *update* und *delete*. Der Bereich Regeln umfasst die Befehle zum Verändern (*enable* und *disable*) des Regelstatus. Im dritten Bereich befinden sich die Funktionen zum Ausführen von Programmen, embedded programs und Prozeduren.

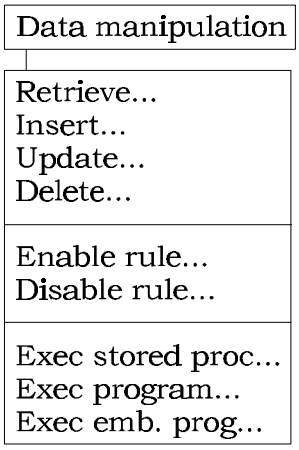

Abbildung 5.4: Submenü "Data manipulation"

Für die Funktionsgruppen Simulation , Transaction , Application und Process wurde je ein eigenes Submenü mit allen benötigten Funktionen erzeugt. Dies sind *create*, *alter*, *destroy* und *execute*. Da alle Submenüs im Wesentlichen den gleichen Aufbau haben, wird in Abbildung 5.5 als Beispiel nur das Submenü für die Simulation dargestellt. Einzig für Prozesse steht ein zusätzlicher Befehl zur grafischen Darstellung zur Verfügung.

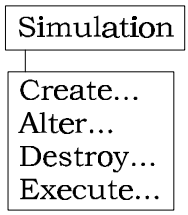

Abbildung 5.5: Submenü "Simulation"

Weitere Submenüs sind für die Regelauswertung und für die Hilfefunktionen vorgesehen. Die für die Regelauswertung benötigten Funktionen sind bis zum jetzigen Zeitpunk noch nicht konkretisiert worden.

5.2.2 Dialogfenster

Dialogfenster werden in ALFRED hauptsächlich für die Spezifikation von Befehlen verwendet. Sie werden aber auch benützt, um dem Benutzer Fehler anzuzeigen und einfache Ja/Nein-Entscheidungen entgegenzunehmen.

5.2.2.1 Richtlinien

Um dem Benutzer ein einheitliches Look and Feel zu bieten, werden für den Entwurf der Dialogfenster die folgenden Richtlinien aufgestellt:

- Dialogfenster bestehen aus den zwei Bereichen *Eingaben* und *Aktionen*.
  - Der Bereich Eingaben befindet sich links im Fenster. Er enthält Felder, Listboxen und Schaltflächen, über die der Benutzer Eingaben tätigen kann. Der Aufbau des Eingabebereichs leitet sich direkt aus der Syntax (vgl. Anhang 9.3) von ALFRED ab.
  - Der Bereich Aktionen befindet sich rechts. Er enthält drei Schaltflächen:
    * Apply überprüft die Eingaben. Sind diese korrekt, werden die Aktionen ausgeführt, sonst wird eine Fehlermeldung angezeigt. Die eingegebenen Daten werden nach der Ausführung der Aktion gelöscht.
    * Ok überprüft ebenfalls die Eingaben, führt Aktionen aus und schliesst zusätzlich das Dialogfenster.
    * Cancel schliesst das Fenster, ohne eine Aktion auszuführen.
    * Nur Apply und Ok im Hauptfenster führen zu einer Verarbeitung des spezifizierten Befehls durch das Verarbeitungssystem.
- Für komplexe Befehle können nicht alle Informationen in einem Fenster dargestellt werden. Die benötigten Daten sollen in diesem Fall logisch zusammengefasst in mehreren Fenstern, die über Schaltflächen im Hauptfenster erreicht werden können, dargestellt werden (vgl. das nachfolgende Beispiel: Erzeugen des Objekts Sachbearbeiter).
- Nicht verfügbare Funktionen sind inaktiv und werden grau dargestellt.

5.2.2.2 Beispiele

Die Anwendung des Menüsystems wird nun anhand von zwei Beispielen illustriert. Für die folgenden Ausführungen wird ein Ausschnitt aus einem Informationssystem einer Bank betrachtet. Im konzeptionellen Datenmodell sind (neben anderen) die zwei Entitätstypen Sachbearbeiter und Konten spezifiziert worden.

Ein Sachbearbeiter wird durch die Attribute Kurzzeichen , Name , Vorname und Gehalt beschrieben. Alle Attribute müssen Werte ungleich NULL enthalten. Es werden zwei Schlüssel festgelegt. Das Attribut Kurzzeichen ist Schlüssel und Primärschlüssel. Die Attribute Name und Vorname bilden zusammen einen weiteren Schlüssel. Ein Sachbearbeiter soll ein Mindestgehalt von Fr. 3000.- erhalten. Wenn ein neuer Sachbearbeiter erfasst wird, soll dem Abteilungsleiter mit einer objektbezogenen Regel eine Nachricht geschickt werden, dass er sich beim Systemadministrator um einen Zugang zum System für den neuen Sachbearbeiter kümmern soll.

Der Entitätstyp Konto enthält die Attribute Nummer , Eröffnungsdatum , Abschlussdatum , Kunde und Saldo . Das Attribut Nummer ist Schlüssel und Primärschlüssel von Konto . Im folgenden wird davon ausgegangen, dass das entsprechende Objekt bereits im System erzeugt wurde.

Zwischen diesen beiden Objekten wird eine Beziehung derart definiert, dass jedes Konto von genau einem Sachbearbeiter verwaltet wird. Jeder Sachbearbeiter kann mehrere Konten verwalten. Dies entspricht dem Beziehungstyp (1,1)-(0,n).

Bei der Erzeugung der Objekte Sachbearbeiter und Konto kann auf die Definition der Fremdschlüsselattribute, über welche die beiden Objekte verbunden werden, verzichtet werden. Das System fügt den im Objekt Konto benötigten Fremdschlüssel Kurzzeichen (des Sachbearbeiters) bei der Erzeugung des Objekts Sachbearbeiter automatisch ein. Weitere Fremdschlüssel werden für diesen Beziehungstyp nicht benötigt.

Für die folgenden zwei Beispiele muss beachtet werden, dass alle nicht verfügbaren Funktionen grau dargestellt sind. Diese Nicht-Verfügbarkeit kann aber zwei Ursachen haben:

- Die Funktion muss abhängig vom Programm-Ablauf gesperrt sein.
- Die Funktion wird im Rahmen des Prototypen nicht implementiert (vgl. Kapitel 5.3.1).

Beispiel: Erzeugen des Objekts Sachbearbeiter

Das in Abbildung 5.6 gezeigte Fenster dient zum Erzeugen eines neuen Objekts. Im obersten Feld wird der Name Sachbearbeiter für das neue Objekt angeben. Für die Spezifikation der vier Attribute muss die Schaltfläche add angeklickt werden. In der Listbox direkt darunter werden bereits definierte Attribute angezeigt. Ein Eintrag setzt sich aus dem Datentyp (in Klammern) und dem Attributnamen zusammen. Einträge in der Liste können über die Schalfläche del wieder gelöscht werden. Wenn kein Attribut ausgewählt ist, wird das erste in der Liste gelöscht, sonst das gewählte.

Als nächstes werden die Integritätsbedingungen festgelegt. Dazu wird die Schaltfläche Integrity constraints angeklickt. Die Beziehung zum Objekt Konto wird über die Schaltfläche Relationships spezifiziert. Um einem neuen Benutzer eine Meldung zu schicken, wird über die Schaltfläche Rules eine objektbezogene Regel definiert.

Das unterste Feld im Fenster dient ausschliesslich der Information des Benutzers. Es zeigt an, in welcher Datenbank das neue Objekt Sachbearbeiter erzeugt wird.

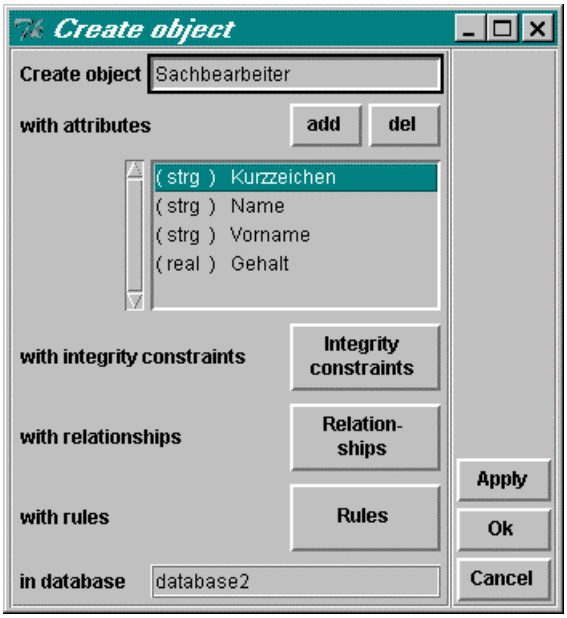

Abbildung 5.6: Fenster zum Erzeugen eines Objekts

Beim Betätigen der Schaltfläche add im Fenster Create object , wird das Fenster zum Eingeben eines neuen Attributs angezeigt (vgl. Abbildung 5.7). Im obersten Feld dieses Fensters wird der Attributname des ersten Attributs Kurzzeichen eingegeben. Es wäre möglich einen bereits definierten Attributnamen durch Anklicken in der Listbox zu übernehmen. Da Kurzzeichen noch nicht in der Liste ist, muss er eingetippt werden. Anschliessend wird durch Anklicken eines Eintrags in der Liste der Datentyp bestimmt. Zur Verfügung stehen die vier Datentypen *integer*, *real*, *boolean* und *string*. Das Kurzzeichen ist vom Typ *string*. Durch drücken der Schaltfläche Apply wird das Attribut im übergeordneten Fenster Create object in die Liste der Attribute eingetragen. Das Fenster Add attribute ist nun für die Definition eines weiteres Attribut bereit.

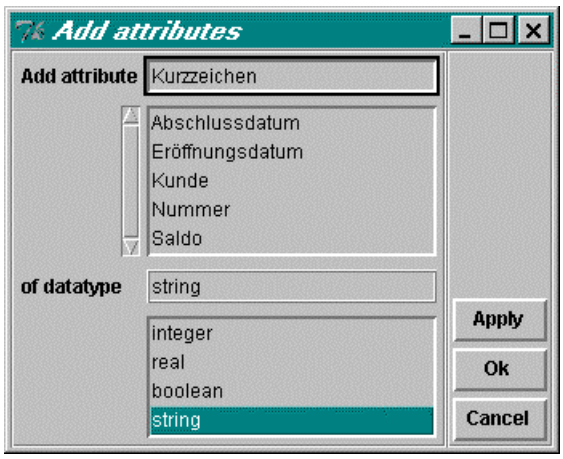

Abbildung 5.7: Fenster zum Eingeben eines neuen Attributs

Durch das Anklicken der Schaltfläche Integrity constraints im Fenster Create object , wird das Fenster zum Eingeben von objektbezogenen Integritätsbedingungen angezeigt (vgl. Abb. 5.8). Dieses besteht aus vier Bereichen. Im ersten Bereich werden über add die Not-Null-IB für alle vier Attribute von Sachbearbeiter hinzugefügt. Mit del können falsch definierte IB wieder gelöscht werden. Analoges gilt für den Bereich define key IC , wo die zwei Schlüssel Kurzzeichen und Name, Vorname definiert werden. Mit select kann der Schlüssel Kurzzeichen als Primärschlüssel ausgewählt werden. Dies ist allerdings erst möglich, wenn der Schlüssel vorher definiert worden ist. Um sicherzustellen, dass jeder Sachbearbeiter ein Mindestgehalt von Fr. 3000.- erhält wird durch Anklicken der Schaltfläche add im untersten Bereich define userdefined IC eine benutzerdefinierte IB festgelegt.

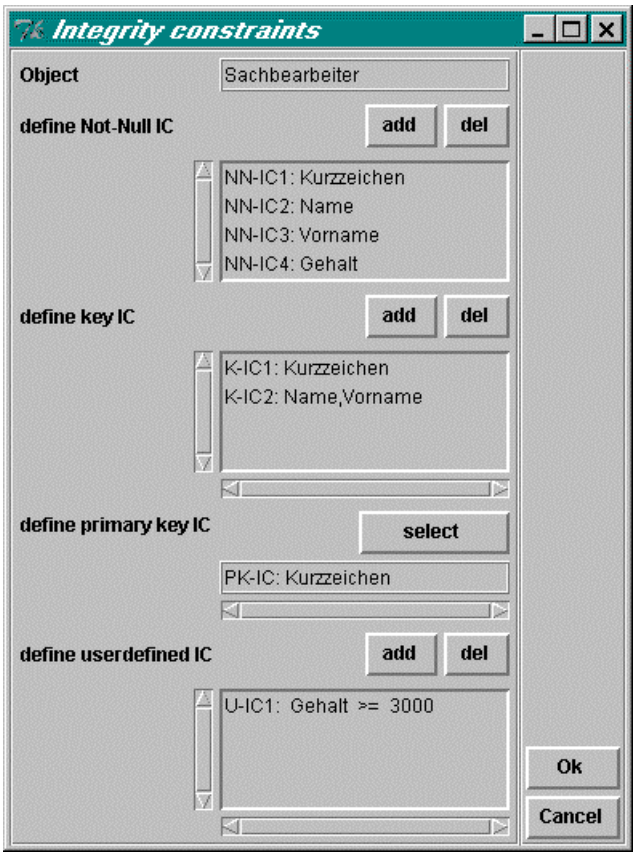

Abbildung 5.8: Fenster zum Eingeben von Integritätsbedingungen

Das Hinzufügen von Not-Null-IB, resp. Not-Null-Attributen erfolgt über das in Abbildung 5.9 gezeigte Fenster. In der Listbox werden alle Objektattribute zur Auswahl angezeigt, für die noch keine Not-Null-IB festgelegt wurde. Um eine Not-Null-IB für das Attribut Kurzzeichen festzulegen, wählt der Benutzer dieses in der Listbox aus. Anschliessend wird die Schaltfläche Apply gedrückt. Dadurch wird einerseits die IB im übergeordneten Fenster Integrity constraints eingetragen und andererseits das Attribut aus der Liste gelöscht. Für die Not-Null-IB der übrigen Attribute wird analog verfahren.

Abbildung 5.9: Fenster zum Festlegen von Not-Null-IB

Die Abbildung 5.10 zeigt das Fenster zum Festlegen von Schlüssel-Integritätsbedingungen. In der Listbox mit allen Objektattributen wird zuerst das Attribut Kurzzeichen angeklickt. Mit Apply wird die Schlüssel-IB im übergeordneten Fenster eingefügt und die Markierung gelöscht. Anschliessend werden die beiden Attribute Name und Vorname angeklickt. Beide erscheinen hervorgehoben. Mit Ok wird diese zweite Schlüssel-IB in Integrity constraints eingetragen und das Fenster Add Not-Null IC geschlossen.

Abbildung 5.10: Fenster zum Festlegen der Schlüssel-Integritätsbedingungen

Um den Schlüssel Kurzzeichen als Primärschlüssel auszuzeichnen, wird dieser im Fenster Select primary key IC (vgl. Abb. 5.11) ausgewählt. Es werden sämtliche definerten Schlüssel zur Auswahl angeboten.

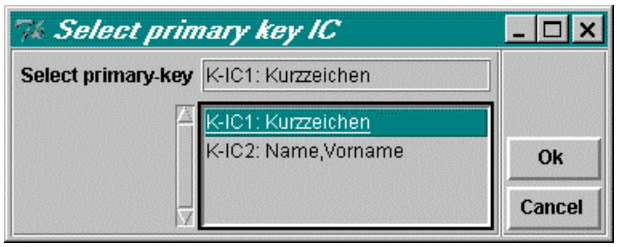

Abbildung 5.11: Fenster zum Auswählen des Primärschlüssels

Die benutzerdefinierte Integritätsbedingungen zur Überprüfung des Mindestgehalts wird im Fenster User defined IC (vgl. Abb. 5.12) festgelegt. Für den linken Operanden wird das Attribut Gehalt aus der Liste aller Attribute angeklickt. Der Vergleichsoperator wird ebenfalls aus einer Liste ausgewählt. Für den rechten Operanden wird die Konstante 3000 eingetippt.

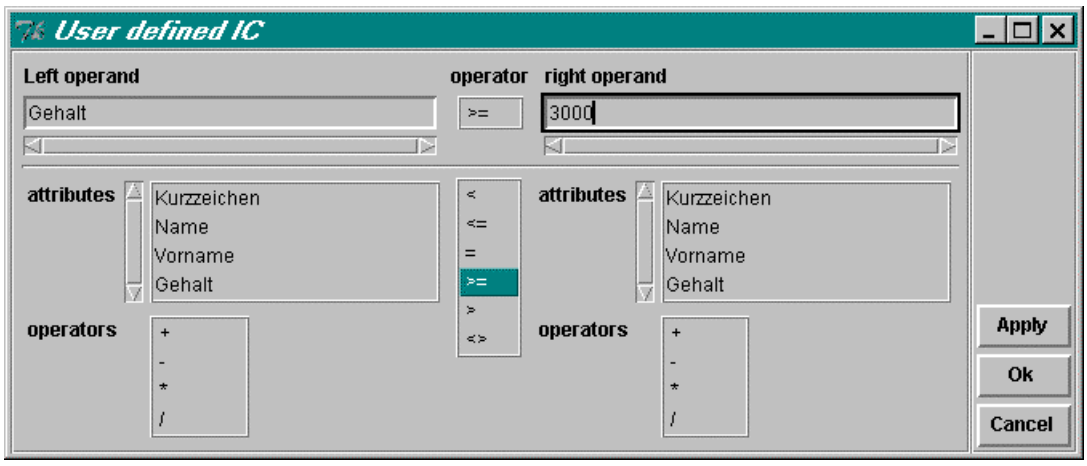

Abbildung 5.12: Fenster zum Eingeben von benutzerdefinierten Integritätsbedingungen

Die Beziehung zum Objekt Konto wird im Fenster Relationships (vgl. Abbildung 5.13) zu einer Liste von Beziehungen hinzugefügt. Die Listbox zeigt alle bereits definierten Beziehungen dieses Typs zu anderen Objekttypen. Jeder Eintrag in der Listbox besteht aus drei Komponenten. Dies sind ein Identifikator (R_ICx), die Kardinalitäten in (min,max)-Notation und der Name des Objekttypen, mit dem es in Beziehung steht. Mit add kann über das in Abbildung 5.14 gezeigte Fenster eine neue Beziehung hinzugefügt werden. Mit del wird eine nicht mehr benötigte oder falsch definierte gelöscht.

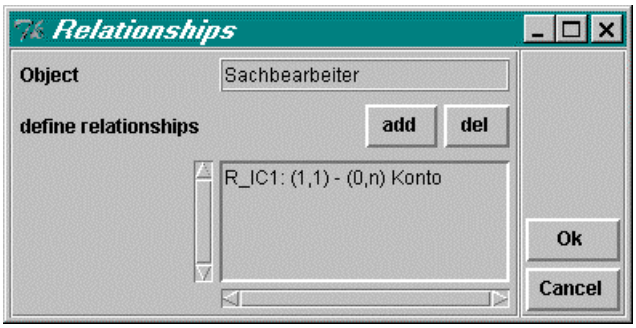

Abbildung 5.13: Fenster der definierten Beziehungstypen

Im Fenster Add relationship (vgl. Abbildung 5.14) wird die Beziehung zwischen den Objekttypen Sachbearbeiter und Konto festgelegt. Über Auswahlschaltflächen wird der Beziehungstyp spezifiziert. Damit jeder Sachbearbeiter mehrere Konten verwalten kann, muss rechts (0,n) angeklickt werden. Jedes Konto soll genau von einem Sachbearbeiter betreut werden. Das heisst, dass links (1,1) gewählt werden muss. Der Name des Objekttyps Konto muss aus der Listbox ausgewählt werden.

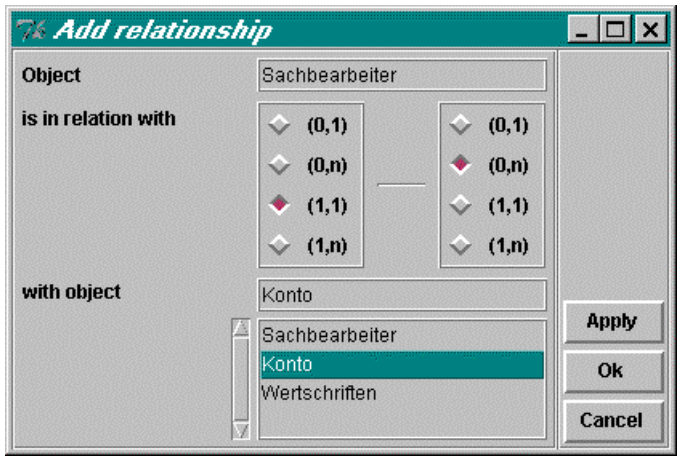

Abbildung 5.14: Fenster zum Festlegen eines neuen Beziehungstyps

Um dem Abteilungsleiter eine Meldung zukommen zu lassen, sobald ein neuer Sachbearbeiter im System erfasst worden ist, wird eine objektbezogene Regel definiert. Dies erfolgt im Fenster Object rules (vgl. Abb. 5.15) wo Regeln hinzugefügt (add) und gelöscht (del) werden können.

Abbildung 5.15: Fenster zum Verwalten der Regeln eines Objekts

Das in Abbildung 5.16 gezeigte Fenster dient dem Erfassen neuer objektbezogener Regeln. Das Vorgehen zum Spezifizieren der Regel wird im nachfolgenden Beispiel Erzeugen der Objektregel Meldung an Abteilungsleiter erläutert.

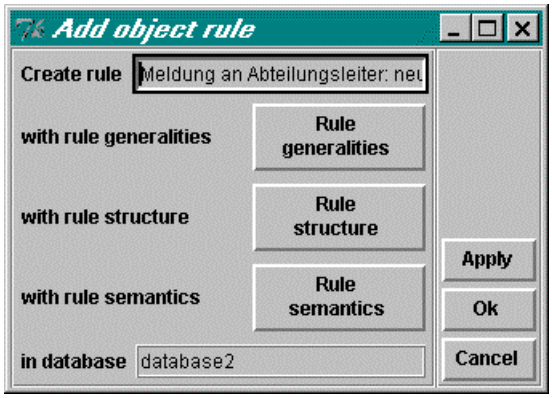

Abbildung 5.16: Fenster zum Eingeben einer neuen Regel für ein Objekt

Beispiel: Erzeugen der Objektregel Meldung an Abteilungsleiter

Das zweite Beispiel zeigt, wie eine neue Objektregel erzeugt wird. Der Ablauf ist für nicht objektbezogene Regeln identisch.

Durch Anklicken der Schaltfläche add im Fenster Object rules wird das Fenster Add object rule (vgl. Abb. 5.16) angezeigt. Hier muss der Regel zuerst im Eingabefeld Create rule ein eindeutiger Namen gegeben werden. Mit Rule generalities werden allgemeine Eigenschaften wie Zugehörigkeit zu Regelgruppen und Status festgelegt. Dies erfolgt mit dem in Abbildung 5.17 gezeigten Fenster. Über die Schaltfläche Rule structure können Angaben zu den Komponenten der Regelstruktur gemacht werden (vgl Abb. 5.19). Durch Anklicken der Schaltfläche Rule semantics kann im gleichnamigen Fenster (vgl. Abb. 5.28) die Regelsemantik festgelegt werden.

Im Fenster Rule generalities (vgl. Abb. 5.17) kann der Benutzer die neue Regel zu Regelgruppen hinzufügen. Dies erfolgt über die Schaltfläche add , welche das in Abbildung 5.18 gezeigte Fenster öffnet. Mit del kann die Zugehörigkeit zu einer Regelgruppe gelöscht werden. Weiter wird der Status der Regel auf aktiv gesetzt (Voreinstellung).

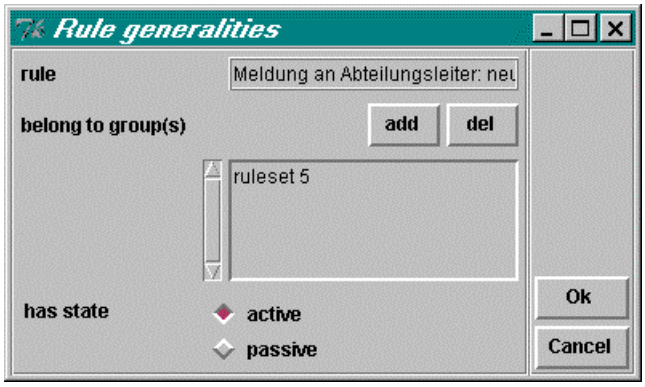

Abbildung 5.17: Fenster zum Festlegen von allgemeinen Regeleigenschaften

Das Hinzufügen von Regelgruppen erfolgt durch die Auswahl der verfügbaren im Fenster Add rule group (vgl. Abb. 5.18). Es werden jeweils nur die Regelgruppen angezeigt, die noch nicht ausgewählt sind. Diese Regel wird nur der Regelgruppe ruleset 5 hinzugefügt.

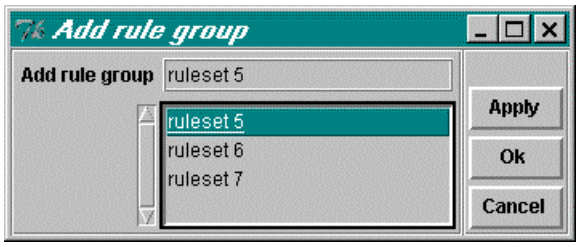

Abbildung 5.18: Fenster zum Hinzufügen von Regelgruppen

Das Festlegen der Strukturkomponenten der Regel erfolgt mit dem in Abbildung 5.19 gezeigten Fenster. Über Auswahlschaltflächen kann die eigentliche Struktur festgelegt werden. Möglich sind ECAA-, ECA-, EA-, CA- und CAA-Regeln. Durch die Wahl einer dieser Optionen wird die Eingabe der Strukturkomponenten gesteuert. Für diese Regel wird eine ECA-Struktur benötigt. Beim Anklicken der entsprechenden Auswahlschaltfläche wird die Schaltfläche rule action bei executes false-action auf inaktiv (grau) gesetzt. Natürlich kann der Strukturtyp jederzeit neu gewählt werden, womit inaktive Schaltflächen wieder verwendet werden können.

Über die Schaltfläche Rule event wird das auslösende Ereignis spezifiziert. Mit Rule condition wird die zu überprüfende Bedingung festgelegt. Über Rule action können auszuführende Aktionen ausgewählt werden.

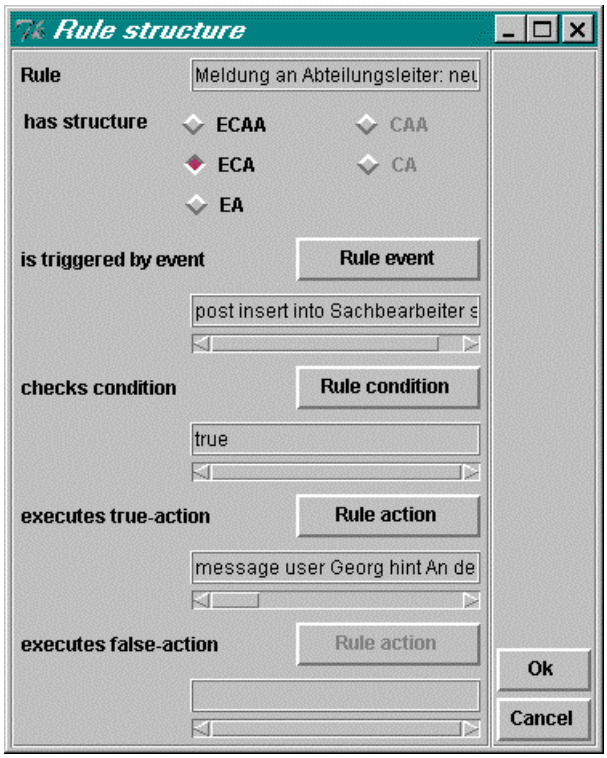

Abbildung 5.19: Fenster zum Festlegen der Strukturkomponenten

Ein Ereignis wird mit dem in Abbildung 5.20 gezeigten Fenster festgelegt. Es kann zwischen sprachlichen, primitiven und zusammengesetzten Ereignissen gewählt werden. In diesem Beispiel wird ein primitives Ereignis verwendet.

Abbildung 5.20: Fenster zum Festlegen des auslösenden Ereignisses

Ein primitives Ereignis kann im Fenster Primitive rule event ausgewählt werden (vgl. Abb. 5.21). Für dieses Beispiel wird ein Datenmanipulationsereignis benötigt. Dieses wird im Fenster Data manipulation events , das durch Anklicken der gleichnamigen Schaltfläche geöffnet wird, festgelegt (vgl. Abb. 5.22). Andere mögliche primitive Ereignisse sind zum Beispiel Datenbank-, Datendefinitions- und abstrakte Ereignisse.

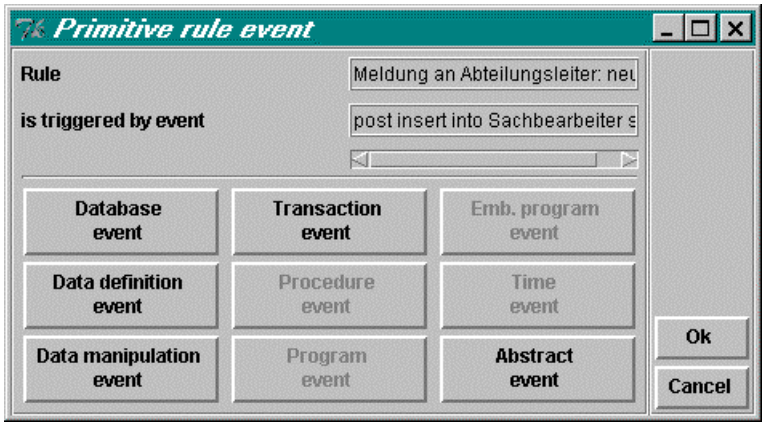

Abbildung 5.21: Fenster zum Auswählen eines primitiven Ereignisses

Das für dieses Beispiel benötigte Datenmanipulationsereignis wird im in Abbildung 5.22 gezeigten Fenster definiert. Zuerst wird der Zeitpunkt auf post , das heisst nach der Ausführung des eigentlichen Datenmanipulationsbefehls gesetzt. Dann wird die Aktion insert into gewählt. Schliesslich muss noch angegeben werden, auf welchen Objekttyp sich das Einfügen beziehen soll. Dazu wird Sachbearbeiter in der entsprechenden Listbox angeklickt.

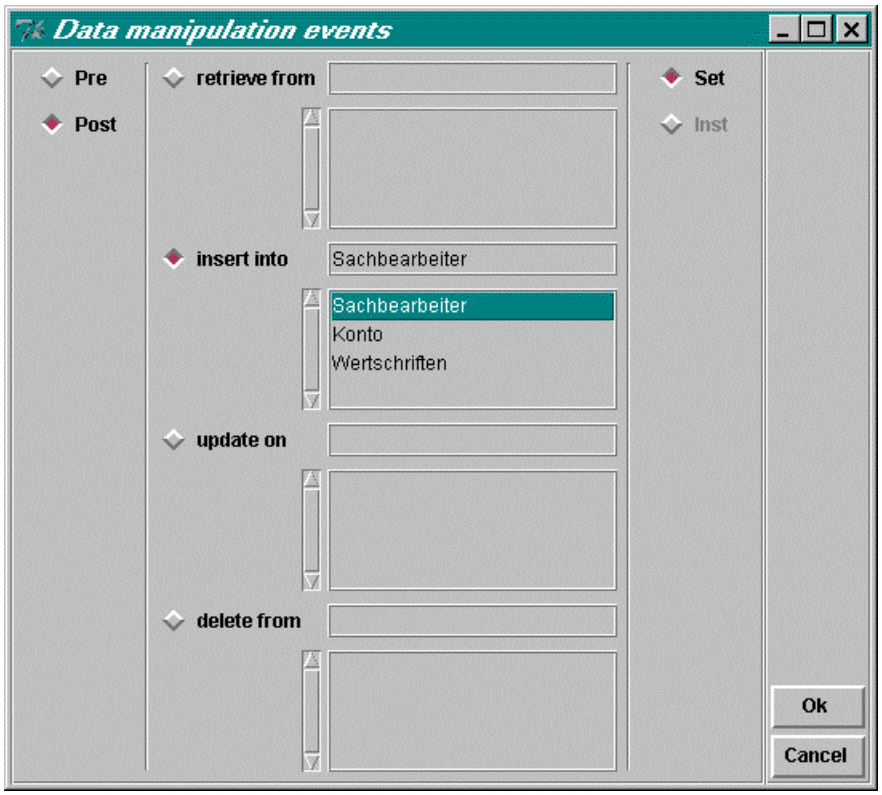

Abbildung 5.22: Fenster zum Definieren eines Datenmanipulationsereignisses

Eine Bedingung wird mit dem in Abbildung 5.23 gezeigten Fenster festgelegt. Es kann zwischen sprachlichen, primitiven und zusammengesetzten Bedingungen gewählt werden. Für dieses Beispiel wird eine primitive Bedingung benötigt.

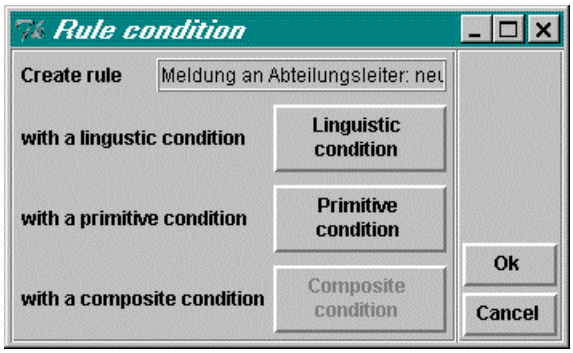

Abbildung 5.23: Fenster zum Festlegen einer Bedingung

Mit dem Fenster in Abbildung 5.24 kann eine primitive Bedingung ausgewählt werden. Da die Meldung an jeden neuen Sachbearbeiter geschickt werden soll, ist die zu wählende Bedingung der boole sche Werte true . Ebenfalls möglich wären false , ein Prädikat, eine Abfrage oder eine Mengenbedingung.

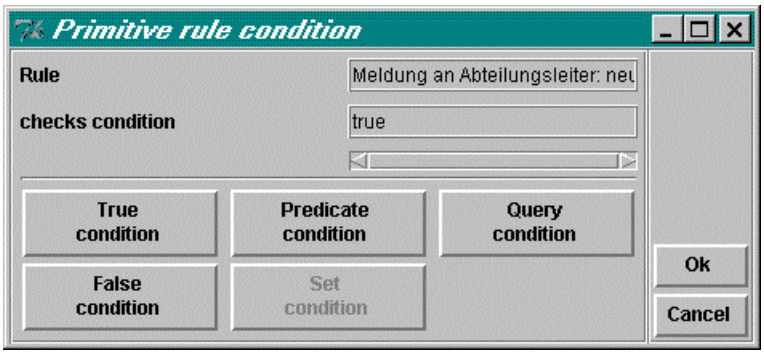

Abbildung 5.24: Fenster zum Definieren einer primitiven Bedingung

Die Aktionskomponente einer Regel wird mit dem in Abbildung 5.25 gezeigten Fenster festgelegt. Eine Meldungsaktion gehört zu den primitiven Aktionen. Ebenfalls möglich sind sprachliche und zusammengesetzte Aktionen.

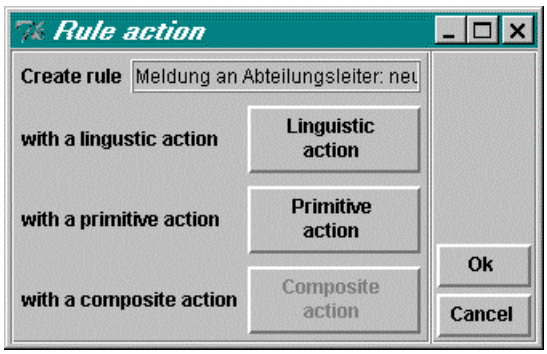

Abbildung 5.25: Fenster zum Festlegen einer Aktion

Im Fenster Primitive rule action (vgl. Abb. 5.26) wird durch Anklicken der Schaltfläche Message das Fenster zum Spezifizieren der Meldung geöffnet.

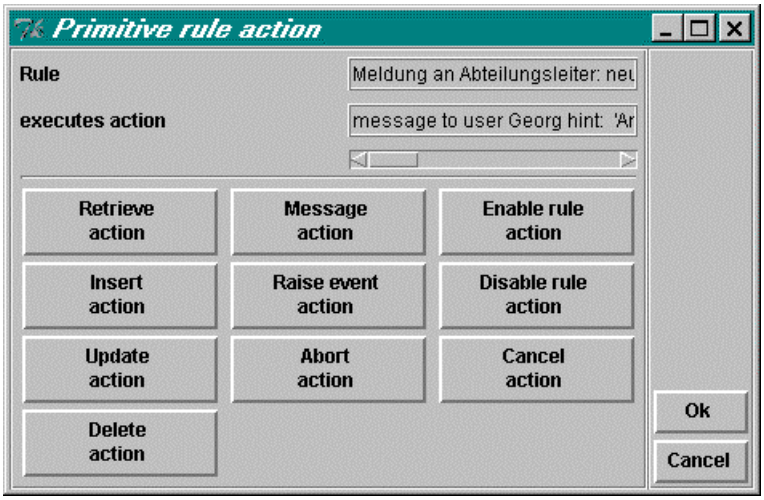

Abbildung 5.26: Fenster zum Auswählen einer primitive Aktionskomponente

Eine Meldung kann an alle Benutzer, an bestimmte Benutzergruppen oder, wie in diesem Fall, an einen einzelnen Benutzer geschickt werden. Dies wird im Fenster Message action (vg. Abb. 5.27) über Auswahlschaltflächen festgelegt. Bei den Benutzergruppen und Benutzern können über die Schaltflächen add neue Einträge zu den Listen hinzugefügt werden. In diesem Beispiel ist der Benutzer Georg Abteilungsleiter. Mit del können nicht gewünschte Einträge aus den Listboxen entfernt werden. Weiter kann für die Meldung ein Präfix wie hint usw. und der Meldungstext eingegeben werden.

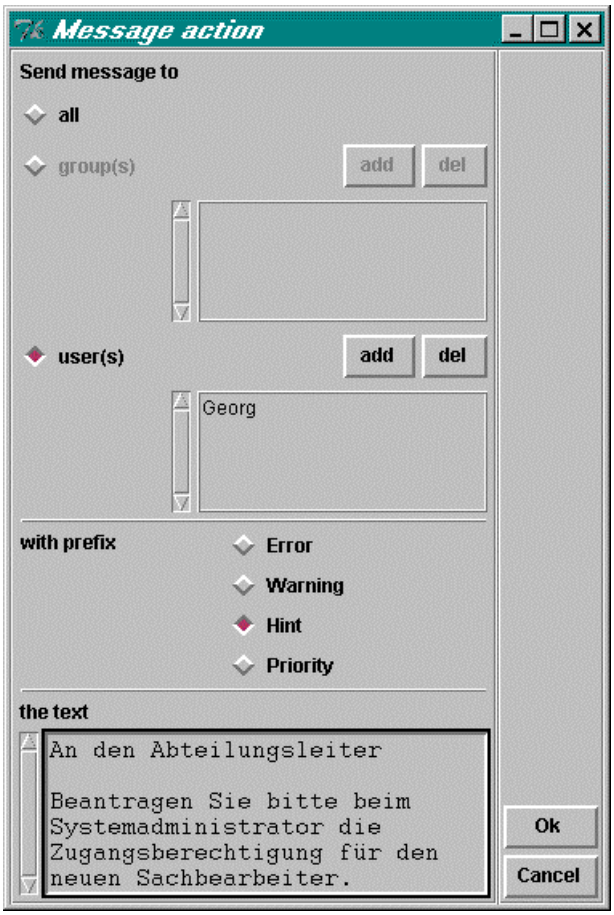

Abbildung 5.27: Fenster zum Eingeben einer Message-Aktion

In dem in Abbildung 5.28 gezeigten Fenster Rule semantics wird die Regelsemantik festgelegt. Dazu gehören die Priorität, Kopplungsmodi, ein Meilenstein und ein Alternativplan. Dieser wird ausgeführt, wenn der Meilenstein nicht eingehalten wird. Es können nur die Kopplungsmodi festgelegt werden, die von der gewählten Regelstruktur benötigt werden. Deshalb ist in diesem Beispiel (ECA-Regel) die Schaltfläche Coupling mode für die Kopplung von Condition und False-Action (C_A coupling) inaktiv. Für beide Kopplungen wird die Voreinstellung *immediate* gewählt.

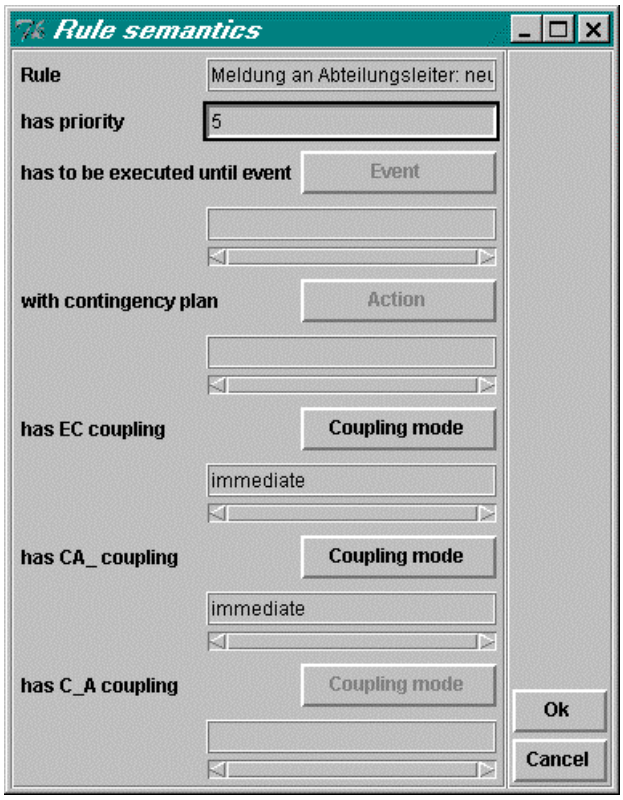

Abbildung 5.28: Fenster zum Definieren der Regelsemantik

Mit dem in Abbildung 5.29 gezeigten Fenster kann ein Kopplungsmodus ausgewählt werden. Beim Öffnen des Fensters ist der im übergeordneten Fenster gewählte ebenfalls bereits ausgewählt.

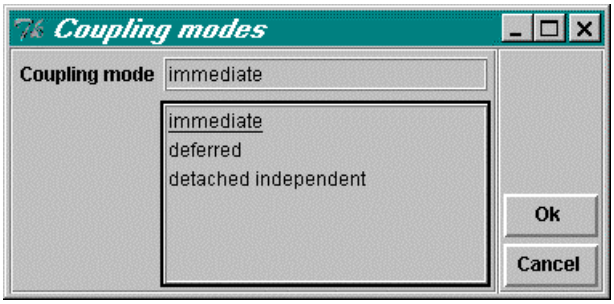

Abbildung 5.29: Fenster zum Auswählen eines Kopplungsmodus

Resümee:

Betrachtet man die oben dargestellten Masken objektiv , so stellt man fest, dass die für die Benutzungsfreundlichkeit von ALFRED wichtigen Eigenschaften (vgl. Kapitel 4) erfüllt sind. Die Konsistenz wird durch einen einheitlichen Aufbau der Masken und Menüs erreicht. Überall wo dies möglich ist, beschränken sich die Eingaben auf die Auswahl von Einträgen aus Listen oder Schaltflächen. Dadurch werden fehlerhafte Eingaben durch den Benutzer weitgehend verhindert. Wo dies nicht möglich ist, werden die Eingaben geprüft und im Fehlerfall eine Meldung angezeigt. Ein besonderes Gewicht wurde auf die Einfachheit der einzelnen Masken gelegt. Dadurch entstand quasi eine sternförmige Benutzerführung, bei welcher der Benutzer immer wieder zur Hauptmaske zurückkommt. So wird es ihm erleichtert, die Übersicht über insbesondere komplexe Eingaben (vgl. Beispiele) zu behalten.

5.3 Prototyp

Um die Machbarkeit der recht vielversprechenden Konzepte von ALFRED zu belegen, wird ein Prototyp mit den wichtigsten Funktionalitäten erstellt. Die hier beschriebene Benutzerschnittstelle ist als Teil des Ganzen (inkl. Prototyp des Verarbeitungssystems) zu sehen.

5.3.1 Funktionen

Im Rahmen der Entwicklung des Prototypen von ALFRED werden nur die wichtigsten Funktionen implementiert. Aus Gründen der Komplexität und des Aufwandes werden nur die Datenbankfunktionen, Teile der Datendefinitionsfunktionen (ohne views, programs, embedded programs und stored procedures) und Teile der Datenmanipulationsfunktionen (wiederum ohne programs, embedded programs und stored procedures) entwickelt. Alles andere (vgl. Kapitel 5.2.1) wird nicht realisiert.

5.3.2 Datenstrukturen

Tcl stellt dem Programmierer nur die komplexen Datentypen Array und Liste zur Verfügung. Da Listen flexibler sind und Tcl starke Listenverarbeitungsfunktionen unterstützt, werden die benötigten Daten in Listenform (Listen von Listen) aufbereitet.

Im Rahmen des Prototyps werden lediglich für die Befehle *create/alter object* und *create/alter rule* komplexere Datenstrukturen benötigt. Diese sehen wie folgt aus (einzelne Listenelemente werden durch { } begrenzt):

- **create/alter object**

  object = {ob_general} {attr_dt_list} {ic_list} {rel_list} {rule_list}
  ob_general = {ob_name} {db_name}
  attr_dt_list = {attr_dt} ... {attr_dt}
  attr_dt = {attribute} {datatype}
  ic_list = {ic_notnull} {ic_key} {ic_pkey} {ic_userdef}
  ic_notnull = {attr_list}
  ic_key = {attr_list} ... {attr_list}
  ic_pkey = {attr_list}
  ic_userdef = {{expression} ... {expression}}
  rel_list = {rel_item} ... {rel_item}
  rel_item = {cardinality} {cardinality} {object_name}
  rule_list = {rule} ... {rule}
  attr_list = {attribute} ... {attribute}
  expression = {left_operand} {operator} {right_operand}

- **create/alter rule**

  rule = {generalities} {structure} {semantics}
  generalities = {rule_name} {ruleset_list} {state}
  ruleset_list = {rs_name} ... {rs_name}
  structure = {structure_type} {event} {condition} {action} {action}
  event = {type} {params}
  condition = {type} {params}
  action = {type} {params}
  semantics = {priority} {event} {action} {c_mode} {c_mode} {c_mode} {c_mode}

5.3.3 Variablen

Für die Übergabe von Parametern zwischen den einzelnen Fenstern wird eine globale Variable verwendet. Diese enthält beim Aufruf eines Unterfensters alle benötigten Angaben wie z.B. den Namen des Objekts, auf das sich die Eingaben im Unterfenster beziehen, oder die Namen aller Attribute, die in einer Listbox dargestellt werden sollen. Beim Schliessen eines Unterfensters werden mit derselben Variablen Werte an das übergeordnete Fenster zurückgegeben. Unter anderem wird auch angegeben, ob das Fenster mit Ok oder mit Cancel geschlossen wurde. Sämtliche auf diese Weise übergebenen Parameter sind als Listen (zum Teil auch als Listen von Listen) organisiert. Der Aufbau der globalen Variablen wird nur durch das aufgerufene Fenster bestimmt.

5.3.4 Implementierung

Die Dialogfenster, die der Benutzer mit dem Menüsystem aufrufen kann, werden je in einem separaten Modul implementiert. Viele der Dialogfenster führen zu weiteren Unterfenstern. Diese werden ebenfalls (ausser einfachen Informations- und Warnungs-Fenstern) in eigenen Tk-Files implementiert. Eine Liste aller erstellten Tk-Files befindet sich in Anhang 9.1.

5.3.5 Anbindung an das Verarbeitungssystem

Durch Anklicken der Schaltflächen Apply oder Ok im Hauptfenster werden die eingegebenen Daten in der Datenstruktur Action-Flow-Petri-Netz (vgl. Kapitel 3.2.1) abgelegt. Dieses einfache Petri-Netz wird anschliessend dem Verarbeitungssystem übergeben. Das Menüsystem wartet dann auf das Resultat der Verarbeitung. Dieses bestimmt die weiteren Aktionen.

6 Ableitung von Regeln

In ALFRED sollen Integritätsbedingungen und die referentielle Integrität durch Regeln sichergestellt werden. Bei der Erzeugung von Objekten werden dazu die entsprechenden Regeln generiert. Durch die Automatisierung der Regelgenerierung wird eine hohe Benutzungsfreundlichkeit erreicht. Der Benutzer muss nur noch ein Minimum an Eingaben vornehmen. Alles andere, insbesondere die Wahl von Ereignissen, Bedingungen und Aktionen der zu erzeugenden Regeln, wird durch das System sichergestellt. Dadurch werden gleichzeitig die zwei Ziele *Einfachheit* und *Sicherheit* (Reduzierung von Fehleingaben) von ALFRED gewährleistet.

Im folgenden wird in einem ersten Teil dargelegt, worauf bei der Regelableitung besonders geachtet werden muss. Der zweite Teil veranschaulicht das Vorgehen zur Regelableitung anhand von zwei Beispielen.

6.1 Konzept

Für die Entwicklung eines solchen Konzepts ist es wichtig, die Mechanismen von Integritätsbedingungen sowie die Kriterien für die Verarbeitung der IB klar darzulegen.

6.1.1 Grundlegende Überlegungen

Unter Integritätsbedingungen wird in der Datenbanktheorie [8] ein Mechanismus für die Eingabekontrolle verstanden. Dieser soll verhindern, dass ungültige Datenwerte in der Datenbank gespeichert werden können. Es werden zwei Typen unterschieden. Die *entitätstypbezogenen* IB beziehen sich immer auf genau einen Entitätstyp. Dieser Typ besteht aus den vier Arten Not-Null-IB, Schlüssel-IB, Primärschlüssel-IB und benutzerdefinierten IB. Die *beziehungstypbezogenen* Integritätsbedingungen betreffen immer zwei Entitätstypen und die Art der Beziehung zwischen diesen.

Die Überprüfung der Integritätsbedingungen muss jeweils bei Datenmanipulations-Operationen durchgeführt werden. Dies sind *insert*, *update* und *delete*. Nicht nötig ist eine Überprüfung bei *select*, da nur lesend auf die Daten zugegriffen wird. Die Überprüfung der Bedingung muss dabei immer sofort (Kopplungsmodus *immediate*) nach dem Auslösen des Ereignisses erfolgen.

Die Bedingungskomponente von Integritätsbedingungen ist abhängig vom Typ. Not-Null-IB beziehen sich auf ein Attribut des Entitätstypen und überprüfen, ob für jedes als Not-Null definierte Attribut ein Wert angegeben wurde. Wichtig ist dabei die Unterscheidung des Null-Werts von 0 und der leeren Zeichenkette , die beide gültige Werte sein können. Schlüssel- und Primärschlüssel-IB überprüfen, ob bereits eine Entität mit der gleichen Wertekombination für die Schlüsselattribute existiert. Dies kann durch einen *select*-Befehl erfolgen. Wird ein Datensatz selektiert, so ist die Integritätsbedingung verletzt. Benutzerdefinierte IB beziehen sich auf eines oder mehrere Attribute einer Relation. Die Bedingungskomponente ist ein boole scher Ausdruck, in dem ausschliesslich Attribute des zugrunde liegenden Entitätstypen und Konstanten verwendet werden dürfen.

Bei referentiellen Integritätsbedingungen wird geprüft, ob jeder Fremdschlüsselwert des abhängigen Entitätstypen mit einem Primärschlüssel des übergeordneten Entitätstypen übereinstimmt.

Bei entitätstypbezogenen IB ist bei einer Verletzung nur ein Abbruch der Aktion möglich. Anders bei beziehungstypbezogenen IB. Dort kann abhängig vom festgelegten Beziehungstyp (und allenfalls der Entscheidung des Benutzers) mit Abbruch (*abort*), Propagieren (*cascade*) oder Initialisieren (*set null*) reagiert werden.

6.1.2 Ableitungskonzept

Wie bereits in Kapitel 4.3 dargelegt, ist es wichtig, dass die generierten Regeln genau den gleichen Inhalt und die gleiche Semantik wie die Integritätsbedingungen haben müssen.

Aus dem Ableitungsprozess sollen zwei Teilergebnisse entstehen. Als erstes wird eine Tabelle mit allen möglichen Ereignissen und Bedingungen erstellt. Es werden die drei Ereignisse *insert*, *update* und *delete* betrachtet. Bei update müssen drei Fälle unterschieden werden: Änderung des Primärschlüssels, Änderung des Fremdschlüssels oder Änderung von Primär- und Fremdschlüssel. Diese Tabelle enthält in Stichworten alle Überlegungen, die im Verlaufe der Erarbeitung des zweiten Teilresultats gemacht wurden. Dabei handelt es sich um die Regel zur Überprüfung der entsprechenden IB in der Syntax der ARDL. Diese soll einfach aus der Tabelle abgeleitet werden können. Damit wird auch gewährleistet, dass sich die benötigte Regel auch wirklich mit den Möglichkeiten der ARDL darstellen lässt.

Ein grosser Vorteil der höheren Flexibilität von Regeln gegenüber herkömmlichen IB ist die Möglichkeit, komplexe Komponenten zu verwenden. So lassen sich z.B. mehrere IB, die durch das gleiche Ereignis ausgelöst werden und gleiche Aktionen zur Folge haben, zu einer einzigen Regel zusammenfassen. Die Bedingungskomponente dieser Regel ist komplex. Sie kann durch eine Oder-Verknüpfung der einzelnen Bedingungen gebildet werden. IB mit gleichen Bedingungs- und Aktionskomponenten lassen sich zu einer Regel mit einer komplexen Ereigniskomponente zusammenfassen. Alle Not-Null-IB eines Entitätstypen lassen sich zum Beispiel so mit einer einzigen Regel mit komplexen Komponenten realisieren. Wird später eine zusätzliche Not-Null-IB hinzugefügt muss nicht eine neue Regel erzeugt, sondern nur die Bedingungskomponente der bestehenden Regel angepasst werden. Bereits definierte Komponenten können so einfach wiederverwendet werden.

6.2 Beispiele

An dieser Stelle wird anhand von je einem Beispiel das Vorgehen bei der Ableitung der Regeln aufgezeigt. Die restlichen Ergebnisse, d.h. Tabellen und Regeln sind im Anhang 9.1 zusammengestellt.

6.2.1 Entitätstypbezogene IB

Am Beispiel der *Not-Null-Integritätsbedingung* wird das Vorgehen zur Bestimmung der abzuleitenden Regeln für die entitätstypbezogenen IB dargestellt.

Im Beispiel von Kapitel 5.2.2 wurde der Entitätstyp Sachbearbeiter beschrieben. Für diesen wurden für die Attribute Kurzzeichen , Name , Vorname und Gehalt Not-Null-IB definiert. Daher ist vor dem Festschreiben (*commit*) beim Einfügen oder Ändern eines Datensatzes zu prüfen, ob alle Attribute einen Wert ungleich NULL enthalten. Ist einer der Werte gleich NULL, so soll dem Benutzer eine Fehlermeldung angezeigt werden und die Verarbeitung abgebrochen werden. Die Tabelle 6.2 zeigt eine Zusammenstellung dieser Tatsachen.

Not-Null-Integritätsbedingungen	
Event:	pre insert
Condition:	Kurzzeichen=NULL or Name=NULL or Vorname=NULL or Gehalt=NULL
Action:	Fehlermeldung, Cancel operation
Event:	pre update
Condition:	Kurzzeichen=NULL or Name=NULL or Vorname=NULL or Gehalt=NULL
Action:	Fehlermeldung, Cancel operation

Tabelle 6.2: Übersicht Not-Null-IB

Aus den in Tabelle 6.2 zusammengestellten Tatsachen lässt sich nun eine Regel ableiten. Für die Bildung des Regelnamens wird der Objektname durch das Kürzel NN (Not-Null) und eine fortlaufende Nummer ergänzt. Als Regelgruppe wird standardmässig *integrity-constraint* gesetzt. Für die Priorität wird ebenfalls ein Defaultwert (0) gesetzt. Der Kopplungsmodus muss für Integritätsbedingungen immer *immediate* sein.

Das Ereignis und die Bedingung in diesem Beispiel sind komplex. So ist es möglich, alle zu beachtenden Eigenschaften mit einer einzigen Regel zu gewährleisten. Ebenso ist es möglich, bei einer zusätzlichen Not-Null-IB diese eine Regel einfach durch eine weitere OR-Klausel im Bedingungsteil zu erweitern.

Die Abbildung 6.1 zeigt, in der Syntax der ARDL, das Ergebnis der Regelableitung aus dem Sachbearbeiter-Beispiel.

```
rule   Sachbearbeiter_NN_1
       belongs to integrity-constraints
       has priority 0
       is active

       on event pre insert in Sachbearbeiter or pre update on Sachbearbeiter
       check condition Sachbearbeiter.Kurzzeichen = NULL or Sachbearbeiter.Name = NULL
          or Sachbearbeiter.Vorname = NULL or Sachbearbeiter.Gehalt = NULL
       execute true_action
       begin
          message 'Error'
             'You have to enter values for Not-Null-attributes.'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 6.1: Regel für Not-Null-IB

6.2.2 Beziehungstypbezogene IB

Im Beispiel von Kapitel 5.2.2 wurde für den Entitätstypen Sachbearbeiter eine (0,n)-(1,1)-Beziehung zum Entitätstyp Konto festgelegt. Das bedeutet, dass jeder Sachbearbeiter eine beliebige Anzahl von Konten betreuen kann. Jedes Konto ist aber genau einem Sachbearbeiter zugeordnet.

Bei einer Realisierung in einem relationalen DBMS benötigt der Entitätstyp Konto für diese Beziehung ein zusätzliches Attribut als Fremdschlüssel zur Referenzierung des zugeteilten Sachbearbeiters. Der Namen dieses Fremdschlüssel-Attributs wird aus dem referenzierten Objekttyp abgeleitet. Dazu werden je die ersten fünf Buchstaben des Objekttyp-Namens und des Primärschlüsselattributs kombiniert. Im Sachbearbeiter-Beispiel ist das Attribut Kurzzeichen als Primärschlüssel definiert. Dies führt zum Fremdschlüssel-Attribut Sachb_Kurzz im Entitätstyp Konto . Dieses Attribut (bei zusammengsetzten Primärschlüsseln sind es mehrere) werden dem Entitätstypen Konto durch das System automatisch hinzugefügt.

Beim Einfügen eines neuen Sachbearbeiters müssen keine referentiellen IB sichergestellt werden, da aus der Sicht dieses Entitätstypen eine (0,n)-Beziehung besteht. Wird das Schlüsselattribut Kurzzeichen geändert, müssen auch alle Fremdschlüssel-Attribute in der abhängigen Relation Konto auf den neuen Wert angepasst werden. Nur so kann die Zuordnung der Konten zu ihrem Sachbearbeiter auch bei einem neuen Kurzzeichen gewährleistet werden. Wird ein Sachbearbeiter gelöscht, sind zwei Aktionen möglich. Entweder werden alle Konten ebenfalls gelöscht (*cascade*) oder die Löschung wird verhindert (*restrict*). Die Entscheidung wird dem Benutzer überlassen.

Beim Einfügen eines neuen Kontos muss geprüft werden, ob der angegebene Sachbearbeiter existiert. Ist dies nicht der Fall, wird dem Benutzer eine Fehlermeldung angezeigt und die Operation abgebrochen. Soll ein Konto einem anderen Sachbearbeiter zugewiesen werden, muss die gleiche Bedingung geprüft werden. Das Löschen eines Kontos ist ohne weiteres möglich.

Die Tabelle 6.3 gibt eine Übersicht aller dieser zu berücksichtigenden Tatsachen.

Sachbearbeiter (1,1) ——— (0,n) Konto			
Event:	insert	Event:	post insert
Condition:		Condition:	not (retrieve * from Sachbearbeiter where Kurzzeichen=Konto.Sachb_Kurzz)
Action:		Action:	Fehlermeldung, Cancel operation
Event:	post update (Kurzzeichen)	Event:	update (Nummer)
Condition:		Condition:	
Action:	update Konto set Konto.Sachb_Kurzz = new.Kurzzeichen where Konto.Sachb_Kurzz = old.Kurzzeichen	Action:	
		Event:	post update (Sachb_Kurzz)
		Condition:	not (retrieve * from Sachbearbeiter where Kurzzeichen=Konto.Sachb_Kurzz)
		Action:	Fehlermeldung, Cancel operation
		Event:	post update (Nummer und Sachb_Kurzz)
		Condition:	not (retrieve * from Sachbearbeiter where Kurzzeichen=Konto.Sachb_Kurzz)
		Action:	Fehlermeldung, Cancel operation
Event:	pre delete	Event:	delete
Condition:		Condition:	
Action:	User-Decision case restrict if (retrieve * from Konto where Sachb_Kurzz = Sachbearbeiter.Kurzzeichen) Fehlermeldung, Cancel operation case cascade delete from Konto where Sachb_Kurzz = Sachbearbeiter.Kurzzeichen	Action:	

Tabelle 6.3: Übersicht (0,n) - (1,1) Beziehung

Aus dieser Übersicht lassen sich nun die in Abbildung 6.2 dargestellten Regeln ableiten:

- **Sachbearbeiter_REFIB_1**

  Nach dem Ändern des Kurzzeichens eines Sachbearbeiters müssen alle von ihm betreuten Konten wieder über sein neues Kurzzeichen referenziert werden können. Das heisst, alle Konten deren Fremdschlüsselattribut Sachb_Kurzz das alte Kurzzeichen des Sachbearbeiters enthälten müssen geändert werden.

- **Sachbearbeiter_REFIB_2**

  Vor dem Löschen eines Tupels in Sachbearbeiter muss der Benutzer entscheiden, wie die abhängige Relation aktualisiert werden soll. Im gewählten Beispiel macht selbverständlich nur die restrict -Variante Sinn. Ein Sachbearbeiter kann also nur gelöscht werden, wenn es keine Konten mehr gibt, die ihm zugeordnet sind.

- **Konto_REFIB_1**

  Nachdem ein neues Tupel in die Relation Konto eingefügt oder Änderungen an einem vorhandenen Tupel durchgeführt wurden, wird geprüft, ob in der Relation Sachbearbeiter der über Konto.Sachb_Kurzz referenzierte Datensatz vorhanden ist. Ist dies nicht der Fall, wird eine Fehlermeldung angezeigt und die (Pseudo-) Transaktion abgebrochen.

```
rule   Sachbearbeiter_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on Sachbearbeiter

       execute action update on Konto set
          Konto.Sachb_Kurzz=new.Kurzzeichen
          where
          Konto.Sachb_Kurzz=old.Kurzzeichen
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   Konto_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post insert in Konto or
          post update on Konto
       check condition not (retrieve * from
          Sachbearbeiter where
          Sachbearbeiter.Kurzzeichen =
          Konto.Sachb_Kurzz)
       execute action begin
          message
             'Error'
             'Relation-constraint violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   Sachbearbeiter_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event pre delete
       check condition (retrieve * from Konto
         where Konto.Sachb_Kurzz =
         Sachbearbeiter.Kurzzeichen)
       execute action begin
          message
             'Error'
             'Record cannot be deleted,
              dependant records exist'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 6.2: Regeln für (0,n) - (1,1) - Beziehungen

Analoge Überlegungen wurden für alle übrigen entitätstyp- und beziehungstypbezogenen Integritätsbedingungen gemacht. Um den Rahmen dieser Arbeit aber nicht zu sprengen werden im Anhang 9.1 nur noch die Tabellen und Regeln dazu dargestellt.

7 Zusammenfassung und Ausblick

Aktive Datenbanken erweitern herkömmliche (passive) DBMS um die Fähigkeit, auf bestimmte Situation selbständig zu reagieren. Am Institut für Wirtschaftsinformatik, Abteilung Information Engineering der Universität Bern wird ALFRED (**A**ctive **L**ayer **F**or **R**ule **E**xecution in **D**atabase Systems) als Prototyp für eine aktive Schicht entwickelt. In dieser Arbeit wird die Benutzerschnittstelle für ALFRED entworfen und als Prototyp implementiert.

Für ALFRED wurden eine Reihe von Anforderungen festgelegt, die auch direkt oder indirekt Einfluss auf das Benutzersystem haben. Dies sind unter anderem:

- **Datenbank- und Plattformunabhängigkeit**
  ALFRED soll auf prinzipiell jedes (passive) Datenbanksystem aufgesetzt werden können.
- **Funktionalität**
  Die obige Anforderung bedeutet unter anderem, dass alle herkömmlichen Datenbankfunktionen unterstützt werden müssen. Zusätzlich werden Funktionen zum Definieren, Verwalten und Steuern des aktiven Verhaltens und weitere benötigt.
- **Benutzungsfreundlichkeit**
  Im Zentrum stehen die Einfachheit und Selbsterklärbarkeit des Systems. Aber auch die Berücksichtigung von Standards (z.B. für die Gestaltung der Oberfläche) ist sehr wichtig.

Aus diesen Anforderungen werden konkrete Anforderungen an das Benutzersystem abgeleitet. Zusätzlich werden systemtechnische Eigenschaften der Komponente Benutzersystem festgelegt. Aus den funktionellen Anforderungen von ALFRED wird das Menüsystem, das aus neun Submenüs besteht, abgeleitet. Für das Spezifizieren der Benutzerbefehle werden Eingabemasken entworfen, mit denen alle benötigten Werte eingeben werden können. Ein besonderes Augenmerk wird dabei darauf gelegt, dass der Benutzer bei der Eingabe bestmöglich unterstützt wird. Wo immer möglich können die Eingaben durch einfaches Anklicken mit der Maus getätigt werden. Realisiert wird das Menüsystem von ALFRED mit Tcl/Tk und Xf. Diese Entwicklungsumgebung bietet vor allem den Vorteil der Plattformunabhängigkeit. Dieses Tool ist als C-Bibliothek implementiert und kann deshalb einfach in eigene Anwedungen integriert werden. Ein weiterer Vorteil ist die lizenzfreie Verfügbarkeit. Verschiedene der erzeugten Masken werden anhand von zwei Anwendungsbeispielen (Erzeugen eines Objekts und Erzeugen einer Regel) gezeigt.

Ferner ist ein Konzept für die automatische Ableitung von Regeln für die Gewährleistung von Integritätsbedingungen erarbeitet worden. Es wird zwischen entitätstyp- und beziehungstypbezogenen IB unterschieden. ALFRED unterstützt Not-Null-IB, Schlüssel-IB, Primärschlüssel-IB und benutzedefinierte IB, sowie alle möglichen binären Beziehungstypen. Für all diese IB werden die zu generierenden Regeln in der Syntax der ARDL angegeben. Durch die automatische Ableitung von Regeln für die Gewährleistung der IB wird erreicht, dass der Benutzer sich bei der Realisierung eines konzeptionellen Datenmodells keine Gedanken machen muss, welche Attribute eines Objekttyps die eines anderen referenzieren. Er muss nur den Beziehungstyp zwischen den beiden Objekttypen festlegen. Das System fügt alle benötigten Fremdschlüssel-Attribute, Zwischentabellen (wo nötig) und Regeln für die Überprüfung der Integritätsbedingungen selbständig ein. So ist es auch einem Nicht-Datenbankspezialisten möglich, ein logisches Datenmodell recht einfach zu erstellen.

Im Laufe der Arbeit hat sich gezeigt, dass der Entwicklungsaufwand mit Tcl/Tk grösser als geplant war. Zwar ist das Erstellen einer Maske mit dem Werkzeug Xf relativ rasch möglich. Der Nachbearbeitungsaufwand um das erzeugte Dialogfenster mit Funktionalität zu füllen, ist jedoch beträchtlich. Trotzdem ist der Entscheid, Tcl/Tk zu verwenden, richtig, weil die für ALFRED zentrale Forderung nach Plattformunabhängigkeit unbedingt erfüllt werden soll. Andere Werkzeuge (z.B. Borland Delphi), die einen ev. geringeren Entwicklungsaufwand ermöglichen würden, erfüllen aber genau diese Forderung nicht. Die Entwicklung eines Prototypen macht aber nur Sinn, wenn dabei das Werkzeug verwendet wird, das später auch bei der Entwicklung des produktiven Systems zum Einsatz kommt. Zudem hat sich Tcl/Tk bei der Entwicklung als äusserst stabil erwiesen.

Im Moment steht das Menüsystem für sich alleine. Für die Listboxen diverser Dialogfenster werden z.B. Funktionen benötigt, die diese Listen aus dem Repository aufbauen. Da dieses noch nicht verfügbar ist, werden diese Funktionen zur Zeit durch Tcl-Prozeduren (Modul ui_funcs.tk) simuliert. In einem nächsten Entwicklungsschritt müssten also diese Schnittstellenfunktionen zum Repository und auch zum Verarbeitungssystem realisiert werden. Weiter fehlen verschiedene Funktionen zur Plausibilisierung von Feldwerten. So ist es zum Beispiel im Moment noch möglich, Buchstaben in Zahlenfelder einzugeben. Diese Funktionen müssen aber in C/C++ realisiert und anschliessend in Tcl eingebunden werden.

Bei der Umsetzung der Syntax der ARDL in die Masken wurde festgestellt, dass noch gewisse Eigenschaften fehlen. So ist es zum Beispiel nicht möglich, den Aktionstyp (z.B. restrict oder cascade) bei der Definition von Beziehungstypen festzulegen. Dies muss aber in konkreten Fällen, wo mehrere Varianten möglich, jedoch nur eine sinnvoll ist, eindeutig festgelegt werden können. Die Behebung dieses Schwachpunkts bedingt jedoch zuerst eine Anpassung der ARDL.

Danke schön!

Ich möchte an dieser Stelle besonders Markus Schlesinger für seine stets hervorragende Betreuung danken. Seine konstruktive Kritik hat mir sehr geholfen.

Danken möchte ich aber auch Susanne Gnepf für ihre Geduld und Unterstützung.

8 Literatur

[1] H. Branding, A. Buchmann, T. Kudrass, J. Zimmermann: Rules in an Open System: The REACH Rule System. In N.W. Paton und M.H. Williams, editors, *Rules in Database Systems*, S. 111 - 126. Springer, London et al., September 1993.

[2] K.R. Dittrich, S. Gatziu, A. Geppert: The Active Database Management System Manifesto: A Rulebase of ADBMS. In T. Sellis, editor, *Rules on Database Systems, Lecture Notes in Computer Science 985*, S. 3 - 17. Springer, Berlin et al., 1995.

[3] Eric F. Johnson: *Graphical Applications with Tcl & Tk*. M&T Books, New York 1996.

[4] G. Knolmayer, H. Herbst, M. Schlesinger: Enforcing Business Rules by the Application of Trigger Concepts. In *Priority Programme Informations Research, Information Conference Module 1, Secure distributed Systems*. S. 28 - 31. SNF, Bern, 1994.

[5] G. Lörincze: *Modellierung, Analyse und Simulation von Regeln in der aktiven Schicht ALFRED*. Diplomarbeit am Institut für Informatik, Universität Bern, 1996.

[6] D.R. McCarthy, U. Dayal: The Architecture Of An Active Data Base Management System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, S. 215 - 224, Portland, Juni 1989.

[7] John K. Ousterhout: *Tcl and the Tk Toolkit*. Addison Wesley, 1994.

[8] G. Schlageter, W. Stucky: *Datenbanksysteme: Konzepte und Modelle*. 2. Auflage, Stuttgart, 1983.

[9] B. Shneiderman: *Designing the User Interface. Strategies for Effective Human-Computer-Interaction*. Reading et al.: AddisonWesley 1987.

[10] J. Widom, S. Ceri: *Active Database Systems*. Morgan Kaufmann, San Francisco, 1996.

9 Anhang

9.1 Ableitung von Regeln bei der Datenmodellierung

Das folgende Kapitel gibt eine Übersicht über die bei der Datenmodellierung zu generierenden Regeln. Zuerst wird jeweils in einer Übersicht dargestellt, was genau beachtet werden muss. Anschliessend daran werden die konkret zu erzeugenden Regeln in der Syntax der ARDL angegeben.

9.1.1 Entitätstypbezogene Integritätsbedingungen

9.1.1.1 Schlüssel-Integritätsbedingungen

Beispiel: Die Attribute *a* und b bilden Schlüssel für das Objekt *x*

Schlüssel-Integritätsbedingungen	
Event:	pre insert
Condition:	retrieve * from x where x.a=a and x.b=b
Action:	Fehlermeldung, Cancel operation
Event:	post update
Condition:	retrieve * from x where x.a=a and x.b=b
Action:	Fehlermeldung, Cancel operation
Event:	delete
Condition:	
Action:	

Tabelle 9.4: Übersicht Schlüssel-IB

```
rule   x_KEY_1
       belongs to integrity-constraints
       has priority 0
       is active

       on event pre insert in x or post update on x
       check condition retrieve * from x where x.a=a and x.b=b
       execute action begin
          message
             'Error'
             'Key integrity-constraint violated.'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.1: Regel für Schlüssel-IB

9.1.1.2 Primärschlüssel-Integritätsbedingung

Beispiel: Die Attribute *a* und *b* bilden den Primärschlüssel für das Objekt *x*

Primärschlüssel-Integritätsbedingungen	
Event:	post insert
Condition:	a=NULL or b=NULL or retrieve * from x where x.a=a and x.b=b
Action:	Fehlermeldung, Cancel operation
Event:	post update
Condition:	a=NULL or b=NULL or retrieve * from x where x.a = a and x.b = b
Action:	Fehlermeldung, Cancel operation
Event:	delete
Condition:	
Action:	

Tabelle 9.5: Übersicht Primärschlüssel-IB

```
rule   x_PKEY_1
       belongs to integrity-constraints
       has priority 0
       is active

       on event post insert in x or post update on x
       check condition a=NULL or b=NULL or retrieve * from x where x.a=a and x.b=b
       execute action begin
          message
             'Error'
             'Primary-Key integrity-constraint violated.'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.2: Regel für Primärschlüssel-IB

9.1.1.3 Benutzerdefinierte Integritätsbedingungen

Beispiel: Für Attribut *a* des Objekts *x* ist ein Maximalwert von 100 festgelegt und das Attribut b muss mindestens so gross sein wie *a*.

Benutzerdefinierte Integritätsbedingungen	
Event:	pre insert
Condition:	a>100 or b<a
Action:	Fehlermeldung, Cancel operation
Event:	pre update
Condition:	a>100 or b<a
Action:	Fehlermeldung, Cancel operation
Event:	delete
Condition:	
Action:	

Tabelle 9.6: Übersicht Benutzerdefinierte IB

```
rule   x_USDEF_1
       belongs to integrity-constraints
       has priority 0
       is active

       on event pre insert in x or pre update on x
       check condition a > 100 or b < a
       execute action begin
          message
             'Error'
             'User-defined integrity-constraint violated.'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.3: Regel für benutzerdefinierte IB

9.1.2 Beziehungstypbezogene Integritätsbedingungen

9.1.2.1 (0,1) - (0,1) - Beziehung

Beide Relationen A und B haben zusätzlich zu ihrem Primärschlüssel pk ein Fremdschlüssel-Attribut fk.

$Relation_A$ (0,1) —— (0,1) $Relation_B$			
Event:	post insert	Event:	post insert
Condition:	a_{fk}≠NULL?	Condition:	a_{fk}≠NULL?
Action:	if not (retrieve * from R_A where $a_{pk}=b_{fk}$ and a_{fk}=NULL) Fehlermeldung, Cancel	Action:	if not (retrieve * from R_B where $b_{pk}=a_{fk}$ and b_{fk}=NULL) Fehlermeldung, Cancel
Event:	post update a_{pk}	Event:	post update b_{pk}
Condition:	a_{fk}≠NULL	Condition:	b_{fk}≠NULL
Action:	update R_B set b_{fk}=new.a_{pk} where $b_{pk}=a_{fk}$	Action:	update R_A set a_{fk}=new.b_{pk} where $a_{pk}=b_{fk}$
Event:	post update a_{fk}	Event:	post update b_{fk}
Condition:	old.a_{fk}=NULL	Condition:	old.b_{fk}=NULL
True-Action:	if not (retrieve * from R_B where $b_{pk}=a_{fk}$ (new) and b_{fk}=NULL) Fehlermeldung, Cancel operation	True-Action:	if not (retrieve * from R_A where a_{pk}=new.b_{fk} and a_{fk}=NULL) Fehlermeldung, Cancel operation
False-Action:	update R_B set b_{fk}=NULL where b_{pk}=old.a_{fk} if new.a_{fk}≠NULL update R_B set $b_{fk}=a_{pk}$ where b_{pk}=new.a_{fk}	False-Action:	update R_A set a_{fk}=NULL where a_{pk}=old.b_{fk} if b_{fk}≠NULL (new) update R_A set $a_{fk}=b_{pk}$ where a_{pk}=new.b_{fk}
Event:	post update (a_{pk} und a_{fk})	Event:	post update (b_{pk} und b_{fk})
Condition:	old.a_{fk}=NULL and new.a_{fk}<>NULL	Condition:	old.b_{fk}=NULL and new.b_{fk}<>NULL
True-Action:	if not (retrieve * from R_B where b_{pk}=new.a_{fk} and b_{fk}=NULL	True-Action:	if not (retrieve * from R_A where a_{pk}=new.b_{fk} and a_{fk}=NULL) Fehlermeldung, Cancel operation
False-Action:	update R_B set b_{fk}=NULL where b_{pk}=old.a_{fk} if new.a_{fk}≠NULL update R_B set $b_{fk}=a_{pk}$ where b_{pk}=new.a_{fk}	False-Action:	update R_A set a_{fk}=NULL where a_{pk}=old.b_{fk} if b_{fk}≠NULL (new) update R_A set $a_{fk}=b_{pk}$ where a_{pk}=new.b_{fk}
Event:	post delete	Event:	post delete
Condition:	a_{fk}≠NULL	Condition:	b_{fk}≠NULL
Action:	update R_B set bfk=NULL where $b_{pk}=a_{fk}$	Action:	update R_A set a_{fk}=NULL where $a_{pk}=b_{fk}$

Tabelle 9.7: Übersicht (0,1)-(0,1) Beziehung

```
rule   RelationA_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA
       check condition RelationA.afk=NULL
       execute action begin
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate

rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA
       check condition (old.apk<>new.apk) and
          (new.afk<>NULL)
       execute action update RelationB set
          RelationB.bfk=new.apk where
          RelationB.bfk=old.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate

rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA
       check condition (old.afk=NULL) and
          (new.afk<>NULL)
       execute true_action begin
          if not (retrieve * from RelationB
          where RelationB.bpk=new.afk and
          RelationB.bfk=NULL
             message
               'error'
               'Ref. Integrity violated'
             cancel operation
       end
       or false_action begin
          update RelationB set
          RelationB.bfk=NULL where
          RelationB.bpk=old.afk
          if new.afk<>NULL
             update RelationB set
             RelationB.bfk=new.apk where
             RelationB.bpk=new.afk
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate

rule   RelationA_REFIB_4
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA
       check condition RelationA.afk<>NULL
       execute action update RelationB set
          RelationB.bfk=NULL where
          RelationB.bpk=RelationA.afk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationB
       check condition RelationB.bfk=NULL
       execute action begin
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate

rule   RelationB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationB
       check condition (old.bpk<>new.bpk) and
          (new.bfk<>NULL)
       execute action update RelationA set
          RelationA.afk=new.bpk where
          RelationA.afk=old.bpk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate

rule   RelationB_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationB
       check condition (old.bfk=NULL) and
          (new.bfk<>NULL)
       execute true_action begin
          if not (retrieve * from RelationA
          where RelationA.apk=new.bfk and
          RelationA.afk=NULL
             message
               'error'
               'Ref. Integrity violated'
             cancel operation
       end
       or false_action begin
          update RelationA set
          RelationA.afk=NULL where
          RelationA.apk=old.bfk
          if new.bfk<>NULL
             update RelationA set
             RelationA.afk=new.bpk where
             RelationA.apk=new.bfk
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate

rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA
       check condition RelationA.afk<>NULL
       execute action update RelationB set
          RelationA.afk=NULL where
          RelationA.apk=RelationB.bfk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.4: Regeln für (0,1) - (0,1) - Beziehungen

9.1.2.2 (0,1) - (0,n) - Beziehung

$Relation_A$ (0,1) ——— (0,n) $Relation_B$			
Event:	insert	Event:	post insert
Condition:		Condition:	$b_{fk} \neq NULL$
Action:		Action:	not retrieve a: $a_{pk} = b_{fk}$ Cancel operation, Fehlermeldung
Event:	post update a_{pk}	Event:	update (b_{pk})
Condition:		Condition:	
Action:	update b_{fk} = new.a_{pk} where b_{fk} = old.a_{pk}	Action:	
		Event:	post update (b_{fk})
		Condition:	$b_{fk} \neq NULL$
		Action:	not retrieve from A where $a_{pk} = b_{fk}$ Cancel operation, Fehlermeldung
		Event:	post update (b_{pk} und b_{fk})
		Condition:	$b_{fk} \neq NULL$
		Action:	not retrieve * from A where $a_{pk} = b_{fk}$ Cancel operation, Fehlermeldung
Event:	delete	Event:	delete
Condition:		Condition:	
Action:	update b_{fk} = NULL where $b_{fk} = a_{pk}$	Action:	

Tabelle 9.8: Übersicht (0,1) - (0,n) Beziehung

Abgeleitete Regeln:

```
rule   RelationA_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA

       execute action update RelationB set
          RelationB bfk=new.apk
          where RelationB.bfk=old.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post insert on RelationB or
         post update on RelationB
       check condition RelationB.bfk<>NULL
       execute action begin
          if not (retrieve * from RelationA
             where RelationA.apk=RelationB.bfk)
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA

       execute action
          update on RelationB set
          RelationB.bfk=NULL
          where RelationB.bfk=RelationA.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.5: Regeln für (0,1) - (0,n) Beziehungen

9.1.2.3 (0,1) - (1,1) - Beziehung

$Relation_A$ (0,1) ——— (1,1) $Relation_B$			
Event:	post insert	Event:	pre insert
Condition:	retrieve from B where $b_{pk} = a_{fk}$ and b_{fk} = NULL	Condition:	b_{fk}≠NULL
True-Action:	Set $b_{fk}=a_{pk}$, $a_{pk}=b_{fk}$	Action:	Fehlermeldung, Cancel operation
False-Action:	Fehlermeldung, Cancel operation		
Event:	post update (a_{pk})	Event:	post update (b_{pk})
Condition:		Condition:	
Action:	update B set b_{fk}=new.a_{pk} where b_{fk}=old.a_{pk}	Action:	update A set a_{fk} = new.b_{pk} where a_{fk}=old.b_{pk}
Event:	post update (a_{fk})	Event:	pre update (b_{fk})
Condition:	retrieve from B where $b_{pk}=a_{fk}$ and b_{fk}=NULL	Condition:	$b_{fk} \neq$ NULL
True-Action:	update B set b_{fk} = new.a_{pk} where b_{pk}=new.a_{fk} update B set b_{fk}=NULL where b_{pk} = old.a_{fk}	True-Action:	delete from A where $a_{pk} = b_{fk}$
False-Action:	Fehlermeldung, Cancel operation	False-Action:	Fehlermeldung, Cancel operation
Event:	post update (a_{pk} und a_{fk})	Event:	pre update (b_{pk} und b_{fk})
Condition:	retrieve from B where $b_{pk}=a_{fk}$ and b_{fk}=NULL	Condition:	$b_{fk} \neq$ NULL
True-Action:	update B set b_{fk}=NULL where b_{pk}=old.a_{fk} update B set $b_{fk}=a_{pk}$ where b_{pk}=new.a_{fk}	True-Action:	update A set a_{fk} = new.b_{pk} where a_{fk}=old.b_{pk} delete from A where $a_{pk} = b_{fk}$
False-Action:	Fehlermeldung,Cancel operation	False-Action:	Fehlermeldung,Cancel operation
Event:	post delete	Event:	post delete
Condition:		Condition:	
Action:	update B set b_{fk} = NULL where $b_{fk}=a_{pk}$	Action:	delete from A where $a_{pk} = b_{fk}$

Tabelle 9.9: Übersicht (0,1) - (1,1) Beziehung

Abgeleitete Regeln:

```
rule   RelationA_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post insert in RelationA
       check condition retrieve * from
          RelationB where
          RelationB.bpk=RelationA.afk and
          RelationB.bfk=NULL
       execute true_action begin
          update on RelationB set
          RelationB.bfk=RelationA.apk
          update on RelationA set
          RelationA.apk=RelationB.bfk
       end
       or false_action begin
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post insert in RelationB
       check condition RelationB.bfk<>NULL
       execute action begin
          message
             'error'
             'Ref. integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA
       check condition retrieve * from
          RelationB where
          RelationB.bpk=RelationA.afk and
          RelationB.bfk=NULL
       execute true_action begin
          update on RelationB set
          RelationB.bfk=RelationA.apk where
          RelationB.bpk=new.afk
          update on RelationB set
          RelationB.bfk=NULL where
          RelationB.bpk=old.afk
       or false_action begin
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationB
       check condition RelationB.bfk<>NULL
       execute action begin
          delete from RelationA where
          RelationA.apk=RelationB.bpk
          update on RelationA set
          RelationA.afk=new.bpk where
          RelationA.afk=old.bpk
       end
       or false_action begin
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA

       execute action update on RelationB set
          RelationB.bfk=NULL where
          RelationB.bfk=RelationA.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationB

       execute action delete from RelationA
          where RelationA.apk=RelationB.bfk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.6: Regeln für (0,1) - (1,1) - Beziehungen

9.1.2.4 (0,1) - (1,n) - Beziehung

$Relation_A$ (0,1) ——— (1,n) $Relation_B$			
Event:	insert	Event:	post insert
Condition:		Condition:	b_{fk}≠NULL
Action:		Action:	not retrieve from A where $a_{pk}=b_{fk}$ Fehlermeldung, Cancel operation
Event:	post update (a_{pk})	Event:	update (b_{pk})
Condition:		Condition:	
Action:	update B set $b_{fk}=a_{pk}$ where b_{fk}=old.a_{pk}	Action:	
		Event:	post update (b_{fk})
		Condition:	b_{fk}≠NULL
		Action:	not retrieve * from A where $a_{pk}=b_{fk}$ Fehlermeldung, Cancel operation
		Event:	post update (b_{pk} und b_{fk})
		Condition:	b_{fk}=NULL
		True-Action:	not retrieve * from B where b_{fk}=old.b_{fk} delete from A where $a_{pk} = b_{fk}$
		False-Action:	not retrieve * from A where $a_{pk}=b_{fk}$ Fehlermeldung, Cancel operation
Event:	post delete	Event:	post delete
Condition:		Condition:	not retrieve * from B where b_{fk}=old.b_{fk}
Action:	update B set b_{fk}=NULL where $b_{fk}=a_{pk}$	Action:	delete from A where $a_{pk}=b_{fk}$

Tabelle 9.10: Übersicht (0,1) - (1,n) Beziehung

Abgeleitete Regeln:

```
rule   RelationA_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA

       execute action update on RelationB set
          RelationB.bfk=new.apk where
          RelationB.bfk=old.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post insert in RelationB or
          post update on RelationB
       check condition RelationB.bfk<>NULL
       execute true_action begin
          if not (retrieve * from RelationA
          where RelationA.apk=RelationB.bfk)
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       or false_action begin
          if (old.bfk<>NULL) and
          not (retrieve * from RelationB where
          RelationB.bfk=old.bfk)
             delete from RelationA where
             RelationA.apk=RelationB.bfk
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA

       execute action update on RelationB set
          RelationB.bfk=NULL where
          RelationB.bfk=RelationA.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationB
       check condition not retrieve * from
          RelationB where
          RelationB.bfk=old.bfk
       execute action delete from RelationA
          where RelationA.apk=RelationB.bfk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.7: Regeln für (0,1) - (1,n) - Beziehungen

9.1.2.5 (0,n) - (1,n) - Beziehung

Für diesen Beziehungstyp muss eine Zwischenrelation AB erstellt werden, die als Primärschlüssel eine Zusammensetzung der Primärschlüssel von A und B enthält.

$Relation_A$ (0,n) ——— (1,n) $Relation_B$	
Event: pre insert Condition: Action: Dialog with user Select existing record from B → insert in AB Insert new record in R_B → insert in AB and B	Event: insert Condition: Action:
Event: post update Condition: Action: update * in AB set $ab_{pk\text{-}a}$=new.a_{pk} where $ab_{pk\text{-}a}$=old.a_{pk}	Event: post update Condition: Action: update AB set $ab_{pk\text{-}b}$=new.b_{pk} where $ab_{pk\text{-}b}$=old.b_{pk}

Event: post delete
Condition:
Action: delete from AB where $ab_{pk\text{-}a}$=a_{pk}
	Event: post delete
	Condition:
	Action: delete from AB where $ab_{pk\text{-}b}$=b_{pk}
	for each deleted record do
	if not (retrieve * from AB where $ab_{pk\text{-}b}$=$ab_{pk\text{-}b}$ (deleted)) then
	delete from A where
	a_{pk}=$ab_{pk\text{-}a}$ (deleted)
	end for each

$Relation_{AB}$	
Event:	post insert
Condition:	not (retrieve * from A where a_{pk}=$ab_{pk\text{-}a}$ and retrieve * from B where b_{pk}=$ab_{pk\text{-}b}$)
Action:	Fehlermeldung, Cancel operation
Event:	post update
Condition:	not (retrieve * from A where a_{pk}=$ab_{pk\text{-}a}$ and retrieve * from B where b_{pk}=$ab_{pk\text{-}b}$)
Action:	Fehlermeldung, Cancel operation
Event:	post delete
Condition:	not (retrieve * from AB where $ab_{pk\text{-}b}$=$ab_{pk\text{-}b}$ (deleted)
Action:	delete from A where a_{pk}=$ab_{pk\text{-}a}$ (deleted)

Tabelle 9.11: Übersicht (0,n) - (1,n) Beziehung

Abgeleitete Regeln:

```
rule   Relation_A_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in Relation_A

       execute action begin
          User-Dialog
          new record for Relation_B
             insert in Relation_B
             insert in Relation_AB (ab_pk_a,
       ab_pk_b)
             values
             (Relation_A.a_pk, Relation_B.b_pk)
          existing record from Relation_B
             insert in Relation_AB (ab_pk_a,
       ab_pk_b)
             values
             (Relation_A.a_pk, Relation_B.b_pk)
        with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   Relation_B_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on Relation_B

       execute action update on Relation_AB set
          Relation_AB.ab_pk_b=new.b_pk where
          Relation_AB.ab_pk_b=old.b_pk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA

       execute action update on RelationAB set
          RelationAB.abpk_a=new.apk where
          RelationAB.abpk_a=old.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA

       execute action delete from RelationAB
          where RelationAB.abpk_a=RelationA.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationB

       execute action delete from RelationAB
          where RelationAB.abpk_b=RelationB.bpk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post insert in RelationAB or
          post update on RelationAB
       check condition not (retrieve * from
          RelationA where
          RelationA.apk=RelationAB.abpk_a and
          retrieve * from RelationB where
          RelationB.bpk=RelationAB.abpk_b)
       execute action begin
          message
             'error'
             'Ref. Integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationAB
       check condition not retrieve * from
          RelationAB where
          RelationAB.abpk_b=old.abpk_b
       execute action delete from RelationA
          where RelationA.apk=old.abpk_a
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.8: Regeln für (0,n) - (1,n) - Beziehungen

9.1.2.6 (1,1) - (1,1) - Beziehung

Beide Relationen A und B haben zusätzlich zu ihrem Primärschlüssel pk ein Fremdschlüssel-Attribut fk.

$Relation_A$ (1,1) ——— (1,1) $Relation_B$			
Event:	pre insert	Event:	pre insert
Condition:		Condition:	
Action:	Dialog with user Insert record in B	Action:	Dialog with user Insert record in A
Event:	post update a_{pk}	Event:	post update b_{pk}
Condition:		Condition:	
Action:	update B set $b_{fk}=a_{pk}$ where $b_{pk}=a_{fk}$	Action:	update A set $a_{fk}=b_{pk}$ where $a_{pk}=b_{fk}$
Event:	pre update a_{fk}	Event:	pre update b_{fk}
Condition:		Condition:	
Action:	Fehlermeldung, Cancel operation	Action:	Fehlermeldung, Cancel operation
Event:	post delete	Event:	post delete
Condition:		Condition:	
Action:	delete from B where $b_{pk}=a_{fk}$	Action:	delete from A where $a_{pk}=b_{fk}$

Tabelle 9.12: Übersicht (1,1) - (1,1) Beziehung

Abgeleitete Regeln:

```
rule   RelationA_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in RelationA

       execute action begin
          User-Dialog
          new record for RelationB
             insert in RelationB
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA
       execute action begin
          if new.apk<>old.apk
             update on RelationB set
             RelationB.bfk=RelationA.apk where
             RelationB.bpk=RelationA.afk
          if new.afk<>old.afk
             message
               'error'
               'Ref. Integrity violated'
             cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA

       execute action delete from RelationB
       where RelationB.bpk=RelationA.afk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in RelationB

       execute action begin
          User-Dialog
          new record for RelationA
             insert in RelationA
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
rule   RelationB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationB
       execute action begin
          if new.bpk<>old.bpk
             update on RelationA set
             RelationA.afk=RelationB.bpk where
             RelationA.apk=RelationB.bfk
          if new.bfk<>old.bfk
             message
               'error'
               'Ref. Integrity violated'
             cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.9: Regeln für (1,1) - (1,1) - Beziehungen

9.1.2.7 (1,1) - (1,n) - Beziehung

Für diesen Beziehungstyp muss eine Zwischenrelation AB erstellt werden, die als Primärschlüssel eine Zusammensetzung der Primärschlüssel von A und B enthält.

$Relation_A$ (1,1) ——— (1,n) $Relation_B$	
Event:	pre insert
Condition:	
Action:	Dialog with user insert record in B → insert b in B and ab in AB
Event:	post update a_{pk}
Condition:	
Action:	update on AB set ab_{pk-a}=new.a_{pk} where ab_{pk-a}=old.ab_{pk-a}
Event:	post delete
Condition:	
Action:	delete * from AB where ab_{pk-a}=a_{pk}

Event:	pre insert
Condition:	
Action:	Dialog with user select record in A → insert ab in AB insert record in A → insert a in A and ab in AB
Event:	post update b_{pk}
Condition:	
Action:	update on AB set ab_{pk-b}=new.b_{pk} where ab_{pk-b}=old.ab_{pk-b}
Event:	post delete
Condition:	
Action:	delete from AB where ab_{pk-b}=b_{pk} if not (retrieve * from AB where ab_{pk-a}=ab_{pk-a} (deleted)) delete from A where a_{pk}=ab_{pk-a}(deleted)

$Relation_{AB}$	
Event:	pre insert
Condition:	not (user= system)
Action:	Fehlermeldung, Cancel operation
Event:	post update ab_{pk-a}
Condition:	not (retrieve * from R_{AB} where ab_{pk-a}=ab_{pk-a} (old))
Action:	Fehlermeldung, Cancel operation
Event:	post update ab_{pk-b}
Condition:	not (retrieve * from R_{AB} where ab_{pk-a}=ab_{pk-a} and ab_{pk-b}=ab_{pk-b} (old))
Action:	Fehlermeldung, Cancel operation
Event:	pre delete
Condition:	not (user=system)
Action:	Fehlermeldung, Cancel operation

Tabelle 9.13: Übersicht (1,1) - (1,n) Beziehung

Abgeleitete Regeln:

```
rule   RelationA_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in RelationA

       execute action begin
          User-Dialog
          new record for RelationB
             insert in RelationB
             insert in RelationAB (abpk_a,
abpk_b)
          values
          (RelationA.apk, RelationB.bpk)
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in RelationB

       execute action begin
          User-Dialog
          new record for RelationA
             insert in RelationA
             insert in RelationAB (abpk_a,
abpk_b)
          values
          (RelationA.apk, RelationB.bpk)
          existing record from RelationA
             insert in RelationAB (abpk_a,
abpk_b)
          values
          (RelationA.apk, RelationB.bpk)
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA

       execute action update on RelationAB set
          RelationAB.abpk_a=new.apk where
          RelationAB.abpk_a=old.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA

       execute action delete from RelationAB
          where RelationAB.abpk_a=RelationA.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationB

       execute action update on RelationAB set
          RelationAB.abpk_ab=new.bpk where
          RelationAB.abpk_b=old.bpk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationB

       execute action begin
          delete from RelationAB where
          RelationAB.abpk_a=RelationA.apk
          if not (retrieve * from RelationAB
          where RelationAB.abpk_a=old.abpk_a)
             delete from RelationA where
             RelationA.apk=old.abpk_a
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in RelationAB
       check condition not (user=system)
       execute action begin
          message
             'error'
             'Ref. integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationAB
       check condition (new.abpk_b<>old.abpk_b)
          and (not (retrieve * from
             RelationAB where
             RelationAB.abpk_a=old.abpk_a and
             RelationAB.abpk_b=old.abpk_b))
       execute action begin
          message
             'error'
             'Ref. integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationAB
       check condition (new.abpk_a<>old.abpk_a)
          and (not retrieve * from
             RelationAB where
             RelationAB.abpk_a=old.abpk_a)
       execute action begin
          message
             'error'
             'Ref. integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre delete from RelationAB
       check condition (not (retrieve * from
          RelationAB where
          RelationAB.abpk_b=old.abpk_b) and
          (user=system))
       execute true_action delete from
          RelationB where
          RelationB.bpk=old.abpk_b
       false_action begin
          message
             'error'
             'Ref. integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.10: Regeln für (1,1) - (1,n) - Beziehungen

9.1.2.8 (1,n) - (1,n) - Beziehung

Für diesen Beziehungstyp muss eine Zwischenrelation AB erstellt werden, die als Primärschlüssel eine Zusammensetzung der Primärschlüssel von A und B enthält.

$Relation_A$ (1,n) —— (1,n) $Relation_B$			
Event:	pre insert	Event:	pre insert
Condition:		Condition:	
Action:	Dialog with user select record in B → insert ab in AB insert record in B → insert a in B and ab in AB	Action:	Dialog with user select record in A → insert ab in AB insert record in A → insert a in A and ab in AB
Event:	post update (a_{pk})	Event:	post update (b_{pk})
Condition:		Condition:	
Action:	update on AB set $ab_{pk\text{-}a}$=new.a_{pk} where $ab_{pk\text{-}a}$=old.a_{pk}	Action:	update on AB set $ab_{pk\text{-}b}$=new.b_{pk} where $ab_{pk\text{-}b}$=old.b_{pk}
Event:	post delete	Event:	post delete
Condition:		Condition:	
Action:	delete * from AB where $ab_{pk\text{-}b}$=$ab_{pk\text{-}b}$ (deleted) if not (retrieve * from AB where $ab_{pk\text{-}b}$=$ab_{pk\text{-}b}$ (deleted)) delete from B where b_{pk}=$ab_{pk\text{-}b}$ (deleted)	Action:	delete from AB where $ab_{pk\text{-}a}$=$ab_{pk\text{-}a}$ (deleted) if not (retrieve * from AB where $ab_{pk\text{-}a}$=$ab_{pk\text{-}a}$ (deleted)) delete from A where a_{pk}=$ab_{pk\text{-}a}$ (deleted)

$Relation_{AB}$	
Event:	insert
Condition:	
Action:	
Event:	post update ($ab_{pk\text{-}a}$)
Condition:	not (retrieve * from AB where $ab_{pk\text{-}a}$=$ab_{pk\text{-}a}$ (deleted))
Action:	Fehlermeldung, Cancel operation
Event:	post update ($ab_{pk\text{-}b}$)
Condition:	not (retrieve * from AB where $ab_{pk\text{-}b}$=$ab_{pk\text{-}b}$ (deleted))
Action:	Fehlermeldung, Cancel operation
Event:	post delete
Condition:	
Action:	not (retrieve * from AB where $ab_{pk\text{-}a}$=$ab_{pk\text{-}a}$ (deleted)) delete from A where a_{pk}=$ab_{pk\text{-}a}$ (deleted) not (retrieve * from AB where $ab_{pk\text{-}b}$=$ab_{pk\text{-}b}$ (deleted)) delete from B where b_{pk}=$ab_{pk\text{-}b}$ (deleted)

Tabelle 9.14: Übersicht (1,n) - (1,n) Beziehung

Abgeleitete Regeln:

```
rule   RelationA_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in RelationA

       execute action begin
          User-Dialog
          new record for RelationB
             insert in RelationB
             insert in RelationAB (abpk_a,
       abpk_b)
             values
             (RelationA.apk, RelationB.bpk)
          existing record from RelationB
             insert in RelationAB (abpk_a,
       abpk_b)
             values
             (RelationA.apk, RelationB.bpk)
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationA
       check condition new.apk<>old.apk
       execute action update on RelationAB set
          RelationAB.abpk_a=new.apk where
          RelationAB.abpk_a=old.apk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationA

       execute action begin
          delete from RelationAB
          where RelationAB.abpk_b=old.abpk_b
          if not retrieve * from RelationAB
          where RelationAB.abpk_b=old.abpk_b
             delete from RelationB where
             RelationB.bpk=old.abpk_b
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event pre insert in RelationB

       execute action begin
          User-Dialog
          new record for RelationA
             insert in RelationA
             insert in RelationAB (abpk_a,
       abpk_b)
             values
             (RelationA.apk, RelationB.bpk)
          existing record from RelationA
             insert in RelationAB (abpk_a,
       abpk_b)
             values
             (RelationA.apk, RelationB.bpk)
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationB
       check condition new.bpk<>old.apk
       execute action update on RelationAB set
          RelationAB.abpk_b=new.bpk where
          RelationAB.abpk_b=old.bpk
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationB_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationB

       execute action begin
          delete from RelationAB
          where RelationAB.abpk_a=old.abpk_a
          if not retrieve * from RelationAB
          where RelationAB.abpk_a=old.abpk_a
             delete from RelationB where
             RelationA.apk=old.abpk_a
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_1
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationAB
       check condition (new.abpk_a<>old.abpk_a)
          and (not retrieve from RelationAB
          where RelationAB.abpk_a=old.abpk_a)
       execute action begin
          message
             'error'
             'Ref. integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationAB_REFIB_2
       belongs to relation-constraints
       has priority 0
       is active

       on event post update on RelationAB
       check condition (new.abpk_b<>old.abpk_b)
          and (not retrieve from RelationAB
          where RelationAB.abpk_b=old.abpk_b)
       execute action begin
          message
             'error'
             'Ref. integrity violated'
          cancel operation
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

```
rule   RelationA_REFIB_3
       belongs to relation-constraints
       has priority 0
       is active

       on event post delete from RelationAB

       execute action begin
          if not (retrieve * from RelationAB)
          where RelationAB.abpk_a=old.abpk_a)
             delete from RelationA where
             RelationA.apk=old.abpk_a
          if not (retrieve * from RelationAB)
          where RelationAB.abpk_b=old.abpk_b)
             delete from RelationB where
             RelationB.bpk=old.abpk_b
       end
       with couplings :
          event - condition       : immediate
          condition - true_action : immediate
```

Abbildung 9.11: Regeln für (1,n) - (1,n) - Beziehungen

9.2 Verzeichnis der Tk-Scripts (alphabetisch)

File-Name	Beschreibung
about.tk	Informationen zum ALFRED-System
add_attr.tk	Auswählen eines Attributnamens
add_rule_group.tk	Auswählen einer Benutzergruppe
add_user.tk	Auswählen eines Benutzernamens
al_obj.tk	Ändern eines bestehenden Objekts
al_rules.tk	Ändern einer bestehenden Regel
al_ruleset.tk	Ändern einer bestehenden Regelgruppe
al_user.tk	Ändern eines Benutzers
alfred.tk	Hauptprogramm von ALFRED
assign_value.tk	Eingeben eines Werts für ein Attribut
co_db.tk	Anmelden an eine Datenbank
condition.tk	Eingeben einer Bedingung
cr_db.tk	Erzeugen einer Datenbank
cr_obj.tk	Erzeugen eines Objekts
cr_obj_add_att.tk	Hinzufügen eines neuen Attributs zu einem Objekt
cr_obj_add_key.tk	Hinzufügen einer Schlüssel-IB
cr_obj_add_notnull.tk	Hinzufügen einer Not-Null-IB
cr_obj_add_relship.tk	Definition eines neuen Beziehungstypen
cr_obj_int_constr.tk	Hinzufügen von Integritätsbedingungen
cr_obj_relship.tk	Hinzufügen eines Beziehungstypen
cr_obj_rules.tk	Hizufügen einer neuen objektbezogenen Regel
cr_obj_rules_add.tk	Erzeugen einer neuen objektbezogenen Regel
cr_obj_sel_pkey.tk	Auswählen einer Schlüssel-IB als Primärschlüssel
cr_obj_userdef_ic.tk	Definieren von benutzerdefinierten IB
cr_rules.tk	Erzeugen von Regeln
cr_ruleset.tk	Erzeugen von Regelgruppen
cr_user.tk	Erzeugen von Benutzern
dc_db.tk	Abmelden von der Datenbank
de_db.tk	Löschen einer Datenbank
de_obj.tk	Löschen eines Objekts
de_rules.tk	Löschen einer Regel
de_ruleset.tk	Löschen einer Regelgruppe
de_user.tk	Löschen eines Benutzers
delete.tk	Löschen von Daten
di_rule.tk	Eine Regel deaktivieren
en_rule.tk	Eine Regel aktivieren
gr_file_access.tk	Dateizugriffsrechte festlegen

File-Name	Beschreibung
gr_object_access.tk	Objektzugriffsrechte festlegen
gr_object_access_grant_oa.tk	Objektzugriffsrechte erteilen
gr_object_access_grant_va.tk	Objektzugriffsrechte entziehen
gr_privileges.tk	Privilegien festlegen
gr_privileges_grant.tk	Privilegien erteilen
insert.tk	Einen Datensatz einfügen
linguistic_rule_action.tk	Festlegen einer sprachlichen Aktionskomponente
linguistic_rule_condition.tk	Festlegen einer sprachlichen Bedingungskomponente
linguistic_rule_event.tk	Festlegen einer sprachlichen Ereigniskomponente
log_on.tk	Anmeldung an das ALFRED-System
new_predicate.tk	Eingeben eines Prädikats
predicate_condition.tk	Eingeben einer Prädikatbedingung
prim_rule_action.tk	Wählen einer primitiven Aktion
prim_rule_action_delete.tk	Festlegen der primitiven Aktion Löschen
prim_rule_action_di_rule.tk	Festlegen der primitiven Aktion Deaktivieren einer Regel
prim_rule_action_en_rule.tk	Festlegen der primitiven Aktion Aktivieren einer Regel
prim_rule_action_insert.tk	Festlegen der primitiven Aktion Einfügen
prim_rule_action_message.tk	Festlegen der primitiven Aktion Meldung
prim_rule_action_raise_event.tk	Festlegen der primitiven Aktion Auslösen eines abstrakten Ereignisses
prim_rule_action_retrieve.tk	Festlegen der primitiven Aktion Daten selektieren
prim_rule_action_update.tk	Festlegen der primitiven Aktion Daten ändern
prim_rule_condition.tk	Festlegen von Bedingungen für eine Regel
prim_rule_condition_predicate.tk	Festlegen einer Bedingungen vom Typ Prädikat
prim_rule_condition_set.tk	Festlegen einer Bedingungen vom Typ Set
prim_rule_event.tk	Auswählen eines Ereignisses
prim_rule_event_db.tk	Festlegen eines Datenbank-Ereignisses
prim_rule_event_dd.tk	Auswählen eines Datendefinitions-Ereignisses
prim_rule_event_dd_object.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ Objekt
prim_rule_event_dd_file_access.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ Dateizugriff

File-Name	Beschreibung
prim_rule_event_dd_object_access.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ Objektzugriff
prim_rule_event_dd_privileges.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ Privilegien
prim_rule_event_dd_rule.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ Regel
prim_rule_event_dd_ruleset.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ Regelgruppe
prim_rule_event_dd_user.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ Benutzer
prim_rule_event_dd_view.tk	Festlegen eines Datendefinitions-Ereignisses vom Typ View
prim_rule_event_dm.tk	Auswählen eines Datenmanipulations-Ereignisses
prim_rule_event_ta.tk	Festlegen eines Transaktions-Ereignisses
query.tk	Festlegen einer Abfrage
re_file_access.tk	Entziehen von Dateizugriffsrechten
re_object_access.tk	Entziehen von Objektzugriffsrechten
re_object_access_revoke_oa.tk	Entziehen eines Zugriffsrechts auf ein Objekt
re_privileges.tk	Entziehen eines Privilegs
retrieve.tk	Datensätze selektieren
rule_action.tk	Wählen der Aktionskomponente einer Regel
rule_condition.tk	Wählen der Bedingungsskomponente einer Regel
rule_coupling.tk	Auswählen der Kopplungsmodi
rule_event.tk	Wählen der Ereignisskomponente einer Regel
rule_general.tk	Festlegen von allgemeinen Eigenschaften einer Regel
rule_semantics.tk	Festlegen der Semantik einer Regel
rule_structure.tk	Festlegen der Struktur einer Regel
set_expression.tk	Eingeben eines Ausdrucks für ein Attribut
ui_funcs.tk	Dummy-Funktionen für Schnittstellen
update.tk	Ändern von Datensätzen

9.3 Syntax der ALFRED Rule Definition Language

Text, Strings und Zeichenketten

```
<lingual>     ::= <string>
<text>        ::= <ident> <separator> [ <text> ]
<separator>   ::= ' ' | ',' | '.' | '!' | '?'
<prefix>      ::= 'ERROR' | 'WARNING' | 'HINT'
<string>      ::= '"' <ident> '"'
<ident>       ::= <letter> [ <ident_rest> ]
<ident_rest>  ::= <letter_rest> [ <ident_rest> ]
<letter_rest> ::= <letter> | '-'
<letter>      ::= 'A' | 'B' | ... | 'Z' | 'a' | 'b' | ... | 'z'
```

Integer- und Dezimalzahlen

```
<value>    ::= <int_real> | <string>
<int_real> ::= [ <add_opr> ] ( <integer> | <real> )
<real>     ::= <integer> '.' <integer>
<num>      ::= <integer>
<integer>  ::= <digit> [ <integer> ]
<digit>    ::= '0' | '1' | ... | '9'
```

Zugriffsrechte und Privilegien

```
<ob_ac> ::= 'retrieve'
          | 'insert'
          | 'update'
          | 'delete'
<fi_ac> ::= 'read'
          | 'write'
          | 'execute'
<pv>    ::= 'create object'
          | 'create view'
          | 'create rule'
          | 'create ruleset'
          | 'create user'
          | 'create procedure'
          | 'create program'
          | 'create embedded program'
          | 'create transaction'
          | 'create simulation'
          | 'create process'

          | 'alter object'
          | 'alter rule'
          | 'alter ruleset'
          | 'alter user'
          | 'alter procedure'
          | 'alter program'
          | 'alter embedded program'
          | 'alter transaction'
          | 'alter simulation'
          | 'alter process'
```

```
          | 'destroy object'
          | 'destroy view'
          | 'destroy rule'
          | 'destroy ruleset'
          | 'destroy user'
          | 'destroy procedure'
          | 'destroy program'
          | 'destroy embedded program'
          | 'destroy transaction'
          | 'destroy simulation'
          | 'destroy process'
```

Operatoren

```
<cmp_opr>  ::= '<' | '<=' | '=' | '>=' | '>' | '<>'
<bool_opr> ::= 'and' | 'or'
<add_opr>  ::= '+' | '-'
<mul_opr>  ::= '*' | '/'
```

Datum, Zeit und Tage

```
<date>      ::= <day> '/' <month> '/' <year>
<time>      ::= <hour> ':' <minute> ':' <second>
<day>       ::= '1' | '2' | ... | '31'
<month>     ::= '1' | '2' | ... | '12'
<year>      ::= '1996' | '1997' | ... | '2100'
<hour>      ::= '0' | '1' | ... | '23'
<minute>    ::= '0' | '1' | ... | '59'
<second>    ::= '0' | '1' | ... | '59'
<week_day>  ::= 'Monday'
              | 'Tuesday'
              | 'Wednesday'
              | 'Thursday'
              | 'Friday'
              | 'Saturday'
              | 'Sunday'
<time_unit> ::= 'Days'
              | 'Weeks'
              | 'Months'
              | 'Quarter_years'
              | 'Half_years'
              | 'Years'
```

Bedingungsvarianten

```
<condition>   ::= <condition> <bool_opr> <condition>
                | [ 'not' ] '(' <condition> ')'
                | <predicate>
<search_cond> ::= <search_cond> <bool_opr> <search_cond>
                | [ 'not' ] '(' <search_cond> ')'
                | <predicate>
                | <pred_query>
<predicate>   ::= <expr> <cmp_opr> <expr>
<pred_query>  ::= <var> ( 'in' | 'all' <cmp_opr> ) '(' <val_list> | <query> ')'
<query>       ::= 'retrieve' ( <att_list> | '*' )
                     'from' <obj_list>
                     'where' <search_cond>
<expr>        ::= <factor> <add_opr> <expr>
                | <factor>
```

```
<factor>      ::= <operand> <mul_opr> <factor>
                | <operand>
<operand>     ::= <value>
                | <var>
                | '(' <expr> ')'
```

Datentypen und Variable

```
<datatype>     ::= 'integer'
                 | 'real'
                 | 'string'
                 | 'boolean'
                 | <ident>
<complex_type> ::= <record>
                 | <array>
                 | <enumerate>
<record>       ::= <ident> '=' 'record' <rec_elem_list> 'end' 'record' ';'
<array>        ::= <ident> '=' 'array' '[' <integer> ']' 'of' <type> ';'
<enumerate>    ::= <ident> '=' 'enum' '(' <val_list> ')' ';'
<var>          ::= <var> '.' <ident> |
                 | <var> '[' <integer> ']'
                 | <ident>
```

Listen

```
<rule_event_list> ::= <rule_event>           [ ',' <rule_event_list> ]
<at_expr_list>    ::= <ident> '=' <expr>     [ ',' <at_expr_list> ]
<at_type_list>    ::= <ident> ':' <type>     [ ',' <at_type_list> ]
<pa_type_list>    ::= <ident> ':' <type>     [ ',' <pa_type_list> ]
<rec_elem_list>   ::= <ident> ':' <type> ';' [ <rec_elem_list> ]
<val_var_list>    ::= ( <value> | <var> )    [ ',' <val_var_list> ]
<ac_list>         ::= <access>    [ ',' <ac_list> ]
<ident_list>      ::= <ident>     [ ',' <ident_list> ]
<ob_list>         ::= <ident>     [ [ 'as' <ident> ] ',' <ob_list> ]
<pv_list>         ::= <privilege> [ ',' <pv_list> ]
<type_list>       ::= <type>      [ ',' <type_list> ]
<val_list>        ::= <value>     [ ',' <val_list> ]
<at_list>         ::= <ident_list>
<pa_list>         ::= <ident_list>
<var_list>        ::= <ident_list>
```

Syntax der ADBL

```
<data_base_lang>    ::= <create_db_cmd>
                      | <destroy_db_cmd>
                      | <connect_db_cmd>
                      | <disconnect_db_cmd>
<create_db_cmd>     ::= 'create' 'database' <ident>
                            [ 'in' 'system' <ident> ] ';'
                      | 'create' 'distributed' 'database' <ident>
                            [ 'in' 'systems' '(' <ident_list> ')' ] ';'
<destroy_db_cmd>    ::= 'destroy' 'database' <ident>
                            [ 'in' 'system' <ident> ] ';'
<connect_db_cmd>    ::= 'connect' 'database' <ident>
                            [ 'in' 'system'    <ident> ]
                            'as' 'user'        <ident>
                            'with' 'password'  <ident> ';'
<disconnect_db_cmd> ::= 'disconnect' 'database' <ident>
                            [ 'in' 'system' <ident> ] ';'
```

Syntax der ADDL

```
<data_definition_lang> ::= <create_ob_cmd>
                         | <create_vi_cmd>
                         | <create_ru_cmd>
                         | <create_rs_cmd>
                         | <create_us_cmd>
                         | <create_sp_cmd>
                         | <create_pg_cmd>
                         | <create_ep_cmd>
                         | <alter_ob_cmd>
                         | <alter_ru_cmd>
                         | <alter_rs_cmd>
                         | <alter_us_cmd>
                         | <alter_sp_cmd>
                         | <alter_pg_cmd>
                         | <alter_ep_cmd>
                         | <destroy_ob_cmd>
                         | <destroy_vi_cmd>
                         | <destroy_ru_cmd>
                         | <destroy_rs_cmd>
                         | <destroy_us_cmd>
                         | <destroy_sp_cmd>
                         | <destroy_pg_cmd>
                         | <destroy_ep_cmd>
                         | <grant_ob_ac_cmd>
                         | <grant_fi_ac_cmd>
                         | <grant_pv_cmd>
                         | <revoke_ob_ac_cmd>
                         | <revoke_fi_ac_cmd>
                         | <revoke_pv_cmd>

<create_ob_cmd>        ::= 'create' 'object' <ident>
                             'with' 'attributes' '(' <at_dt_list> ')'
                             [ 'in' 'system' <ident> ] ';'
<create_vi_cmd>        ::= 'create' 'view' <ident>
                             'on' 'basis' 'of' '(' <ob_vi_list> ')'
                             'with' 'attributes' '(' <ob_vi_at_list> ')'
                             'where' <search_cond>
                             [ 'in' 'system' <ident> ]';'
<create_ru_cmd>        ::= 'create' <rule> ';'
<create_rs_cmd>        ::= 'create' 'ruleset' <ident> ';'
<create_us_cmd>        ::= 'create' 'user' <ident>
                             'with' 'password' <ident> ';'
<create_sp_cmd>        ::= 'create' 'stored' 'procedure' <ident>
                             'with' 'parameters' '(' <pa_dt_list> ')'
                             [ 'in' 'system' <ident> ] ';'
<create_pg_cmd>        ::= 'create' 'program'  <ident>
                             [ 'in' 'system' <ident> ] ';'
<create_ep_cmd>        ::= 'create' 'embedded' 'program' <ident>
                             [ 'in' 'system' <ident> ] ';'
```

```
<alter_ob_cmd>         ::= 'alter' 'object' <ident>
                           [ 'change' 'name' 'to' <ident>              ]
                           [ 'change' attributes'
                                { '(' 'from' <ident> 'to' <ident> ')' } ]
                           [ 'add' 'attributes' '(' <at_dt_list> ')'   ]
                           [ 'delete' 'attributes' '(' <at_list> ')'   ] ';'
<alter_ru_cmd>         ::= 'alter' 'rule' <ident>
                           [ 'change' 'name'
                                'to'  <ident>                          ]
                           [ 'change' 'group'
                                'from' <rule_set>
                                'to'   <rule_set>                      ]
                           [ 'change' 'priority'
                                'from' <integer>
                                'to'   <integer>                       ]
                           [ 'change' 'state'
                                'from' <rule_state>
                                'to'   <rule_state>                    ]
                           [ 'change' 'latest' 'execution' 'point'
                                'from' <rule_event>
                                'to'   <rule_event>                    ]
                           [ 'change' 'contingency' 'plan'
                                'from' <rule_action>
                                'to'   <rule_action>                   ]
                           [ 'change' 'rule' 'event'
                                'from' <rule_event>
                                'to'   <rule_event>                    ]
                           [ 'change' 'rule' 'condition'
                                'from' <rule_condition>
                                'to'   <rule_condition>                ]
                           [ 'change' 'rule' 'true_action'
                                'from' <rule_action>
                                'to'   <rule_action>                   ]
                           [ 'change' 'rule' 'false_action'
                                'from' <rule_action>
                                'to'   <rule_action>                   ]
                           [ 'change' 'rule' 'action'
                                'from' <rule_action>
                                'to'   <rule_action>                   ]
                           [ 'change' 'event-condition' 'coupling'
                               'from'  <rule_coupling>
                               'to'    <rule_coupling>                 ]
                           [ 'change' 'condition-true_action' 'coupling'
                                'from' <rule_coupling>
                                'to'   <rule_coupling>                 ]
                           [ 'change' 'condition-false_action' 'coupling'
                                'from' <rule_coupling>
                                'to'   <rule_coupling>                 ]
                           [ 'change' 'event-action' 'coupling'
                                'from' <rule_coupling>
                                'to'   <rule_coupling>                 ]
                           [ 'change' 'condition-action' 'coupling'
                                'from' <rule_coupling>
                                'to'   <rule_coupling>                 ] ';'
<alter_rs_cmd>         ::= 'alter' 'ruleset' <ident>
                           [ 'change' 'name' 'to' <ident> ] ';'
```

```
<alter_us_cmd>         ::= 'alter' 'user' <ident>
                             [ 'change' 'name' 'to' <ident>     ]
                             [ 'change' 'password' 'to' <ident> ] ';'
<alter_sp_cmd>         ::= 'alter' 'stored' 'procedure' <ident>
                             [ 'change' 'name' 'to' <ident>               ]
                             [ 'change' 'parameters'
                                  { '(' 'from' <ident> 'to' <ident> ')' } ]
                             [ 'add' 'parameters' '(' <pa_dt_list> ')'   ]
                             [ 'delete' 'parameters' '(' <pa_list> ')'   ] ';'
<alter_pg_cmd>         ::= 'alter' 'program' <ident>
                             [ 'change' 'name' 'to' <ident> ]
                             [ 'modify' 'source'            ] ';'
<alter_ep_cmd>         ::= 'alter' 'embedded' 'program' <ident>
                             [ 'change' 'name' 'to' <ident> ]
                             [ 'modify' 'source'            ] ';'

<destroy_ob_cmd>       ::= 'destroy' 'object' <ident> ';'
<destroy_vi_cmd>       ::= 'destroy' 'view' <ident> ';'
<destroy_ru_cmd>       ::= 'destroy' 'rule' <ident> ';'
<destroy_rs_cmd>       ::= 'destroy' 'ruleset' <ident> ';'
<destroy_us_cmd>       ::= 'destroy' 'user' <ident> ';'
<destroy_sp_cmd>       ::= 'destroy' 'stored' 'procedure' <ident> ';'
<destroy_pg_cmd>       ::= 'destroy' 'progam' <ident> ';'
<destroy_ep_cmd>       ::= 'destroy' 'embedded' 'program' <ident> ';'

<grant_ob_ac_cmd>      ::= 'grant' '(' <ob_ac_list> ')'
                             'on' '(' <ob_list> ')'
                             'to' 'users' <ident_list> ';'
<grant_fi_ac_cmd>      ::= 'grant' '(' <fi_ac_list> ')'
                             'on' '(' <sp_pg_list> ')'
                             'to' 'users' <ident_list> ';'
<grant_pv_cmd>         ::= 'grant' '(' <pv_list> ')'
                             'to' 'users' <ident_list> ';'
<revoke_ob_ac_cmd>     ::= 'revoke' '(' <ob_ac_list> ')'
                             'on' ( <ob_list> ')'
                             'from' 'users' <ident_list> ';'
<revoke_fi_ac_cmd>     ::= 'revoke' '(' <fi_ac_list> ')'
                             'on' '(' <sp_pg_list> ')'
                             'from' 'users' <ident_list> ';'
<revoke_pv_cmd>        ::= 'revoke' '(' <pv_list> ')'
                             'from' 'users' <ident_list> ';'
```

Syntax der ADML

```
<data_manipulation_lang> ::= <retrieve_cmd>
                           | <insert_cmd>
                           | <update_cmd>
                           | <delete_cmd>
                           | <message_cmd>
                           | <raise_cmd>
                           | <enable_rule_cmd>
                           | <disable_rule_cmd>
                           | <exec_sp_cmd>
                           | <exec_pg_cmd>
                           | <exec_ep_cmd>
```

```
<retrieve_cmd>          ::= 'retrieve' '(' <att_list> | ( '*' | 'all' ) ')'
                            'from' <ident>
                            [ 'where' <search_cond> ] ';'
<retrieve_cmd>          ::= 'retrieve' '(' <obj_att_list> ')'
                            [ 'where' <search_cond> ] ';'
<insert_cmd>            ::= 'insert' 'in' <ident> [ '(' <att_list> ')' ]
                            'values' '(' <val_list> ')' ';'
<update_cmd>            ::= 'update' 'on' <ident>
                            'set' '(' <att_expr_list> ')'
                            [ 'where' <search_cond> ] ';'
<delete_cmd>            ::= 'delete' 'from' <ident>
                            [ 'where' <search_cond> ] ';'
<message_cmd>           ::= 'message' 'to' 'user'  <ident>
                            '(' [ <prefix> ':' ] <text> ')' ';'
                          | 'message' 'to' 'group' <ident>
                            '(' [ <prefix> ':' ] <text> ')' ';'
                          | 'message' 'to' 'all'
                            '(' [ <prefix> ':' ] <text> ')' ';'
<raise_cmd>             ::= 'raise' 'abstract' 'event' <ident> ';'
<enable_rule_cmd>       ::= 'enable' 'rule' <ident>
                            'set' 'rule' 'state'
                            'to' 'active' [ <rule_tmp_state> ] ';'
<disable_rule_cmd>      ::= 'disable' 'rule' <ident>
                            'set' 'rule' 'state'
                            'to' 'passive' [ <rule_tmp_state> ] ';'
<exec_sp_cmd>           ::= 'exec' 'stored' 'procedure' <ident>
                            '(' <value_list> ')' ';'
<exec_pg_cmd>           ::= 'exec' 'program' <ident> '(' ')' ';'
<exec_ep_cmd>           ::= 'exec' 'embedded' 'program' <ident> '(' ')' ';'
```

Syntax der Regelstruktur

```
<rule>           ::=  'rule' <ident>
                        <rule_general>
                        <rule_structure> ';'
<rule_general>   ::= [ 'belongs' 'to'                    <rule_set>         ]
                     [ 'has' 'priority'                  <rule_order>       ]
                     'is'                                <rule_state>
                     [ 'has' 'to' 'be' 'executed' 'until' <rule_timing>
                       [ 'with' 'contingency' 'plan'     <rule_cont_plan> ] ]
<rule_set>       ::= <ident> [ ',' <rule_set> ]
<rule_order>     ::= <integer>
<rule_state>     ::= ( 'active' | 'passive' ) [ <rule_tmp_state> ]
<rule_tmp_state> ::= 'between' ( 'Now' | <rule_event> ) 'and' <rule_event>
<rule_timing>    ::= <rule_event>
<rule_cont_plan> ::= <rule_action>
<rule_structure> ::= 'on' 'event'                   <event>
                          'check' 'condition'       <condition>
                          'execute' 'true_action'   <action>
                          'or'      'false_action'  <action>
                     [ 'with' 'couplings' ':'
                         'event - condition'       ':' <rule_coupling>
                         'condition - true_action' ':' <rule_coupling>
                         'condition - false_action' ':' <rule_coupling> ]
                   | 'on' 'event'                   <event>
                         'check' 'condition'        <condition>
                         'execute' 'action'         <action>
```

```
                    [ 'with' 'couplings' ':'
                        'event - condition'        ':' <rule_coupling>
                        'condition - action'       ':' <rule_coupling> ]
                  | 'on' 'event'                     <event>
                        'execute' 'action'           <action>
                    [ 'with' 'coupling' ':'
                        'event - action'           ':' <rule_coupling> ]

                  | 'check' 'condition'              <condition>
                        'execute' 'true_action'      <action>
                        'or'      'false_action'     <action>
                    [ 'with' 'couplings' ':'
                        'condition - true_action'  ':' <rule_coupling>
                        'condition - false_action' ':' <rule_coupling> ]

                  | 'check' 'condition'              <condition>
                        'execute' 'action'           <action>
                    [ 'with' 'coupling' ':'
                        'condition - action'       ':' <rule_coupling> ]
<rule_coupling>  ::= 'immediate'
                   | 'deferred'
                   | 'detached' 'sequential' 'dependent'
                   | 'detached' 'parallel'   'dependent'
                   | 'detached' 'exlusive'   'dependent'
                   | 'detached' 'independent'
```

Syntax der Ereigniskomponenten

```
<event>           ::= <rule_event>
                    | <lingual>
<rule_event>      ::= '(' <rule_event> ')'
                    | <primitive_event>
                    | <complex_event> [ <param_context> ]
<param_context>   ::= 'recent'
                    | 'chronological'
                    | 'continuous'
                    | 'cumulative'
```

Syntax der primitiven Ereignisse

```
<primitive_event> ::= <db_event>
                    | <dd_event>
                    | <dm_event>
                    | <canc_event>
                    | <trans_event>
                    | <proc_event>
                    | <prog_event>
                    | <e_prog_event>
                    | <simu_event>
                    | <appl_event>
                    | <process_event>
                    | <time_event>
                    | <abst_event>
<pre_post>        ::= 'pre'
                    | 'post'
<db_event>        ::= <pre_post> <db_cmd>
```

```
<db_cmd>          ::= ( 'create'     'database'         )
                    | ( 'destroy'    'database' <ident> )
                    | ( 'connect'    'database' <ident> )
                    | ( 'disconnect' 'database' <ident> )
<dd_event>        ::= <pre_post> <dd_cmd>
<dd_cmd>          ::= ( 'create' 'object'             )
                    | ( 'create' 'view'               )
                    | ( 'create' 'rule'               )
                    | ( 'create' 'rule_set'           )
                    | ( 'create' 'user'               )
                    | ( 'create' 'transaction'        )
                    | ( 'create' 'stored procedure'   )
                    | ( 'create' 'program'            )
                    | ( 'create' 'embedded' 'program' )
                    | ( 'create' 'simulation'         )
                    | ( 'create' 'application'        )
                    | ( 'create' 'process'            )
                    | ( 'alter'  'object' <ident>             )
                    | ( 'alter'  'rule' <ident>               )
                    | ( 'alter'  'rule_set' <ident>           )
                    | ( 'alter'  'user' <ident>               )
                    | ( 'alter'  'transaction' <ident>        )
                    | ( 'alter'  'stored' procedure' <ident>  )
                    | ( 'alter'  'program' <ident>            )
                    | ( 'alter'  'embedded' 'program' <ident> )
                    | ( 'alter'  'simulation' <ident>         )
                    | ( 'alter'  'application' <ident>        )
                    | ( 'alter'  'process' <ident>            )
                    | ( 'destroy' 'entity' <ident>             )
                    | ( 'destroy' 'view' <ident>               )
                    | ( 'destroy' 'rule' <ident>               )
                    | ( 'destroy' 'rule_set' <ident>           )
                    | ( 'destroy' 'user' <ident>               )
                    | ( 'destroy' 'transaction' <ident>        )
                    | ( 'destroy' 'stored' procedure' <ident>  )
                    | ( 'destroy' 'program' <ident>            )
                    | ( 'destroy' 'embedded' 'program' <ident> )
                    | ( 'destroy' 'simulation' <ident>         )
                    | ( 'destroy' 'application' <ident>        )
                    | ( 'destroy' 'process' <ident>            )
                    | ( 'grant' 'object' 'access' )
                    | ( 'grant' 'file'   'access' )
                    | ( 'grant' 'privilege'       )
                    | ( 'revoke' 'object' 'access' )
                    | ( 'revoke' 'file' 'access'   )
                    | ( 'revoke' 'privilege'       )
<dm_event>        ::= <pre_post> <dm_cmd> ( 'set' | 'inst' )
<dm_cmd>          ::= ( 'retrieve' 'from' <ident> )
                    | ( 'insert'   'in'   <ident> )
                    | ( 'update'   'on'   <ident> )
                    | ( 'delete'   'from' <ident> )
<canc_event>      ::= <pre_post> 'cancel'
<trans_event>     ::= <pre_post> <trans_cmd>
```

```
<trans_cmd>       ::= ( 'begin'  'of' 'transaction'         )
                    | ( 'abort'  'of' 'transaction'         )
                    | ( 'end'    'of' 'transaction'         )
                    | ( 'commit' 'of' 'transaction'         )
                    | ( 'execute'     'transaction' <ident> )
<proc_event>      ::= <pre_post> 'exec' 'stored' 'procedure' <ident>
<prog_event>      ::= <pre_post> 'exec' 'program' <ident>
<e_prog_event>    ::= <pre_post> 'exec' 'embedded' 'program' <ident>
<simu_event>      ::= <pre_post> 'exec' 'simulation' <ident>
<appl_event>      ::= <pre_post> 'exec' 'application' <ident>
<process_event>   ::= <pre_post> 'exec' 'process' <ident>
<time_event>      ::= [ <date> '@' ] <time>
<abst_event>      ::= 'abstract' 'event' <ident>
```

Komplexe Ereignisse

```
<complex_event>   ::= <bool_opr>
                    | <choice_opr>
                    | <seq_opr>
                    | <rep_opr>
                    | <time_opr>
                    | <intvl_opr>
<bool_opr>        ::= <rule_event> <bool_opr> <rule_event>
<choice_opr>      ::= 'weak'   'choice' <num> 'of' '(' <rule_event_list> ')'
                    | 'strong' 'choice' <num> 'of' '(' <rule_event_list> ')'
<seq_opr>         ::= 'weak'   'sequence' '(' <rule_event_list> ')'
                    | 'strong' 'sequence' '(' <rule_event_list> ')'
<rep_opr>         ::= 'every'         <num> 'occurence'  'of' <rule_event>
                    | 'every' 'after' <num> 'occurences' 'of' <rule_event>
<time_opr>        ::= 'timespan' ( <time> | <num> <time_unit> ) 'after'
                         <rule_event>
                    | 'every' <time> <start_pnt>
                    | 'every' [ <num> ] <time_unit> <start_pnt>
                    | 'every' [ <num> ] <week_day> 'at' <time> <start_pnt>
                    | 'every' [ <num> ] ( 'day' | <week_day> ) 'in' 'month' 'at'
                      <time> <start_pnt>
<start_pnt>       ::= 'beginning' 'at' ( 'Now' | <rule_event> )
<intvl_opr>       ::= [ 'not' ]     <rule_event> 'in' 'interval' <interval>
                    | 'the' 'first' <rule_event> 'in' 'interval' <interval>
                    | 'the' 'last'  <rule_event> 'in' 'interval' <interval>
                    | 'the' <num>   <rule_event> 'in' 'interval' <interval>
<interval>        ::= '(' ( 'Now' | <rule_event> ) ';' <rule_event> ')'
```

Syntax der Bedingungskomponenten

```
<condition>           ::= <rule_condition>
                        | <bool_function>
                        | <lingual>
<rule_condition>      ::= '(' [ 'not' ] <rule_condition> ')'
                        | <primitive_condition>
                        | <complex_condition>
```

Syntax der primitiven Bedingungen

```
<primitive_condition> ::= <predicate>
                        | <query>
                        | <pred_query>
                        | 'true'
                        | 'false'
```

Syntax der komplexen Bedingungen

```
<complex_condition>   ::= <rule_condition> <bool_opr> <rule_condition>
<bool_function>       ::= 'function' <ident> '(' [ <par_list> ] ')'
```

Syntax der Aktionskomponenten

```
<action>           ::= <rule_action>
                     | <lingual>
<rule_action>      ::= [ 'instead' 'do' ]
                         ( <primitive_action> | <complex_action> )
```

Syntax der primitiven Aktionen

```
<primitive_action> ::= <dm_action>
                     | <mess_action>
                     | <rule_action>
                     | <canc_action>
                     | <raise_action>
<dm_action>        ::= <retrieve_cmd>
                     | <insert_cmd>
                     | <update_cmd>
                     | <delete_cmd>
<mess_action>      ::= <message_cmd>
<rule_action>      ::= <enable_rule_cmd>
                     | <disable_rule_cmd>
<canc_action>      ::= <cancel_cmd>
<raise_action>     ::= <raise_cmd>
```

Syntax der komplexen Aktionen

```
<complex_action>   ::= <db_cmd_block>
                     | <proc_cmd>
                     | <prog_cmd>
                     | <e_prog_cmd>
                     | <appl_cmd>
                     | <process_cmd>
                     | <dialog_cmd>
<db_cmd_block>     ::= 'begin' ( <seq_block> | <par_block> ) 'end' ';'
<seq_block>        ::= ( <primitive_action> | <transaction> ) <db_cmd_block>
<db_cmd_block>     ::= ( <primitive_action> | <transaction> ) [ <seq_block> ]
<par_block>        ::= 'begin' 'parallel' <db_cmd_block> 'end' 'parallel' ';'
<proc_cmd>         ::= 'exec' 'stored' 'procedure' <ident>
                         '(' <par_list> ')' ';' [ <proc_cmd> ]
<prog_cmd>         ::= 'exec' 'program' <ident> '(' ')' ';'
                         [ <prog_cmd> ]
<e_prog_cmd>       ::= 'exec' 'embedded' 'program' <ident> '(' ')' ';'
                         [ <e_prog_cmd> ]
<appl_cmd>         ::= 'exec' 'application' <ident> ';'
                         [ <appl_cmd> ]
<process_cmd>      ::= 'exec' 'process' <ident> ';'
                         [ <process_cmd> ]
<dialog_cmd>       ::= 'exec' 'dailog' <ident> ';'
                         [ <dialog_cmd> ]
```