Rechnerunterstützung für die konzeptuelle Modellierung

Klaus Gaßner
Abt. Datenbanken und Informationssysteme
Universität Ulm
e-mail: gassner@informatik.uni-ulm.de

Kurzfassung

Ein konzeptuelles Modell ist eine stark abstrahierte Darstellung eines Ausschnittes der realen Welt. Viele der bekannten Modelliertechniken geben nur eine Sicht des Weltausschnittes wieder. Für die Verwendung von konzeptuellen Modellen z.B. in der Analysephase der Software-Entwicklung müssen allerdings mehrere Sichten betrachtet werden. Die Diskussion über verschiedene Basistechniken sowie kombinierte und objekt-orientierte Modelliertechniken ergibt, daß es wünschenswert ist, nacheinander mehrere Modelle mit unterschiedlichen Techniken zu erstellen und die Teilmodelle zu einem konsistenten Gesamtmodell zu integrieren.

Durch den Einsatz rechner-gestützter Werkzeuge (CASE-Tools) kann der Modellierer bei der Anwendung einzelner Modelliertechniken in vielerlei Hinsicht unterstützt werden. Zur Unterstützung des Integrationsprozesses werden mehrere Werkzeuge zu einem Modelliersystem (I-CASE-Systeme) zusammengefaßt, wobei die Art und die Zahl der in einem Modelliersystem verwendeten Werkzeuge festgelegt ist und nicht vom geplanten Verwendungszweck der konzeptuellen Modelle abhängig gemacht werden kann. Durch die Standardisierung von Repository-Schnittstellen (IRDS, Repository Manager, etc.) konnten offene d.h. erweiterbare Modelliersysteme entwickelt werden. Aufgrund der unbekannten Zusammensetzung solcher offenen Systeme ist es aber schwierig, Funktionen für die Integration von (mit verschiedenen Werkzeugen erstellten) Teilmodellen bereitzustellen.

Als Lösungsvorschlag für ein sowohl erweiterbares als auch integrierendes Modelliersystem werden in dieser Arbeit der Aufbau und die Funktionsweise von C-CASE-Systemen (Configurable-CASE-Systeme) vorgestellt. Diese neue Art von Modelliersystemen besteht aus einem generischen Modelliereditor, einem semantischen Repository und einer Integrationsfunktionalität. Alle drei Komponenten müssen vor dem Einsatz des Systems konfiguriert werden. Sie können aber auch nachträglich neu konfiguriert werden, wenn z.B. eine zusätzliche Modelliertechnik eingesetzt werden soll.

1 Einleitung

Die Entwicklung von Applikationen wird im allgemeinen zumindest in eine Analyse-, eine Entwurfs- und eine Implementationsphase unterteilt. In der Analysephase wird festgelegt, welche Probleme mit der Applikation gelöst werden sollen. Um die Anforderungen ermitteln, beschreiben und festlegen zu können, die die Applikation erfüllen muß, werden (meist vom Applikationsentwickler und vom Nutzer gemeinsam) konzeptuelle Modelle erstellt. Für den Entwickler sind diese Modelle die Basis für den Entwurf einer softwaretechnischen Systemarchitektur und die anschließende Implementation der Applikation. Die Qualität der konzeptuellen Modelle ist daher von entscheidender Bedeutung für die Qualität der fertigen Applikation. Ein gutes konzeptuelles Modell muß den Weltausschnitt, den es beschreibt vollständig und richtig wiedergeben. Da diese Übereinstimmung nur von Menschen geprüft werden kann, müssen die Modelle gut lesbar sein und eine gute Kommunikation zwischen Nutzer und Entwickler ermöglichen.

In dieser Arbeit werden in einem Überblick (Kapitel 2) einige der bekanntesten Modelliermethoden und die zur Darstellung der Modelle verwendeten Modelliertechniken vorgestellt, mit denen konzeptuelle Modelle erstellt werden können. Dies dient als Basis, um existierende Weiterentwicklungen wie die kombinierte, die inkrementelle und die objektorientierte Modellierung vorzustellen und zu charakterisieren. Die konzeptuelle Modellierung kann durch den Einsatz von computer-gestützten Werkzeugen (CASE-Tools) unterstützt werden. Für die Entwicklung größerer Modelle müssen meist mehrere Werkzeuge benutzt werden. Um die Interaktion zwischen den Werkzeugen zu ermöglichen, werden diese zu einem Modelliersystem zusammengefaßt. Bei der Charakterisierung (Kapitel 3) verschiedener Arten von Modelliersystemen werden einige Problem herausgearbeitet, die mit existierenden Systemen nicht zufriedenstellend lösbar sind. So muß ein Modelliersystem einerseits eine enge Integration verschiedener Werkzeuge unterstützen, andererseits sollte es aber auch um neue Werkzeuge erweiterbar sein.

Mit C-CASE-Systemen wird die Idee einer neuen Art von Modelliersystemen vorgestellt (Kapitel 4). Ein Bestandteil eines C-CASE-Systems ist ein generischer Modelliereditor, der so konfiguriert werden kann, daß mit ihm die Modellierung mit den verschiedensten Modelliertechniken unterstützt werden kann. Die Modelldaten aller Modell werden unabhängig von der verwendeten Modelliertechnik in einem gemeinsamen Schema eines semantischen Repositories gespeichert. Damit wird die gemeinsame Verwendung von Modelldaten in verschiedenen Modelliertechniken und somit auch die Integration von Teilmodellen zu einem Gesamtmodell ermöglicht. Erweitert werden kann ein C-CASE-System durch die Ergänzung der Konfigurations-Definitionen.

Zur Entwicklung eines geeigneten Rahmens für diese Art von Modelliersystemen müssen ein Reihe von Problemen gelöst werden. Die nähere Beschreibung der Einzelprobleme (Kapitel 5) wird verbunden mit einer eingehenden Diskussion dazu existierender Ansätze. Einige dieser Problemstellungen werden im Rahmen der Dissertation des Autors bearbeitet. In einem Ausblick (Kapitel 6) werden die bereits erarbeiteten eigenen Lösungsansätze sowie die weitere Vorgehensweise kurz vorgestellt.

2 Konzeptuelle Modellierung

Für die konzeptuelle Modellierung sind verschiedene Modelliermethoden und Modelliertechniken entwickelt worden. Die beiden Begriffe können folgendermaßen unterschieden werden. Eine Methode ist eine planmäßig angewandte Vorgehensweise zur Erreichung eines vorgegebenen Zieles. Das Ziel einer Modelliermethode ist die Erstellung von präzisen, aber leichtverständlichen Modellen. Bei Modelliertechniken handelt es sich um Beschreibungsformalismen bzw. Sprachen zur Formulierung von Modellen.

Solche Modelliersprachen bzw. -techniken können genau wie natürliche Sprachen oder Programmiersprache mittels einer Syntax-, einer Präsentations- und einer Semantikbeschreibung definiert werden.

- Die Syntaxbeschreibung legt die zur Sprache gehörenden Sprach- bzw. Modellkonstrukte und deren erlaubte Zusammensetzung fest.
- Die Präsentationsbeschreibung, auch Notation genannt, legt fest, ob die Modellkonstrukte in reiner Textform oder in Textform als Schlüsselworte dargestellt werden, oder in Graphenform mit verschiedenartigen Knoten- und Kantendarstellungen.
- Die Semantikbeschreibung erklärt, wie ein Modell zu interpretieren ist. Jedes Modell beschreibt einige Aspekte eines Weltausschnittes und abstrahiert von vielen anderen. Daher wird im Rahmen der Semantikbeschreibung einer Modelliertechnik festgelegt, welche semantischen Aspekte in den Modellen repräsentiert werden können.

2.1 Einführung

Im folgenden werden einige der bekanntesten und am meisten verwendeten Modelliermethoden und Modelliertechniken vorgestellt. Die Beschreibung gliedert sich jeweils in fünf Abschnitte: Unter der Überschrift "Grundidee und Modelliertechnik" wird die Grundidee der jeweiligen Modelliermethode vorgestellt und die in dieser Methode wichtigste Modelliertechnik. In den nächsten beiden Abschnitten wird die "Syntax und Präsentation" sowie die "Semantik" dieser jeweils wichtigsten Modelliertechnik erläutert. In den Abschnitten "Methodische Unterstützung" werden jeweils Vorgehensweisen zur Erstellung von Modellen in der zuvor besprochenen Modelliertechnik vorgestellt. Die letzten Abschnitte "weiteres Vorgehen" beziehen sich wiederum auf die Modelliermethode. Es werden einige Ideen vermittelt, welche anderen Modelliertechniken noch in dieser Methode eingesetzt werden und wie ein in der vorgestellten Modelliertechnik erstelltes Modell weiterverarbeitet werden kann.

Zur Veranschaulichung der Modelliertechniken wird jeweils ein konzeptuelles Modell einer Bibliothek präsentiert, die verbal wie folgt beschrieben wird. Der Stil dieser Beschreibung entspricht in etwa dem, wie ein Bibliotheksleiter sie formulieren würde.

Beispielanwendung
In einer Bibliothek teilt ein Kunde dem Bibliothekar den Titel eines Buches mit, das er

ausleihen möchte. Ist es vorhanden, prüft der Bibliothekar die Kundendaten, trägt die Kundennummer und die Leihfrist in die Karteikarte des Buches ein und händigt dem Kunden das Buch aus. Existiert dieser Titel noch nicht in der Bibliothek, wird das Buch gekauft. Nach Eintreffen der Lieferung werden die Buchdaten erfaßt, ein passender Standort ermittelt und das Buch ins Regal gestellt. Möchte ein Kunde sein ausgeliehendes Buch verlängern, wird eine neue Leihfrist festgelegt. Ein Buches wird nach der Rückgabe wieder ins Regal gestellt und der Ausleihvermerk gestrichen.

2.1.1 Strukturierte Analyse

Grundidee und Modelliertechnik

In der Modelliermethode "Strukturierten Analyse" (SA) [You89] wird ein System oder eine Applikation ausgehend von Datenflüssen und Prozessen, die diese Datenflüsse verarbeiten oder transformieren, modelliert. Die konzeptuellen Modelle werden als Daten-Fluß-Diagramme (DFD) dargestellt.

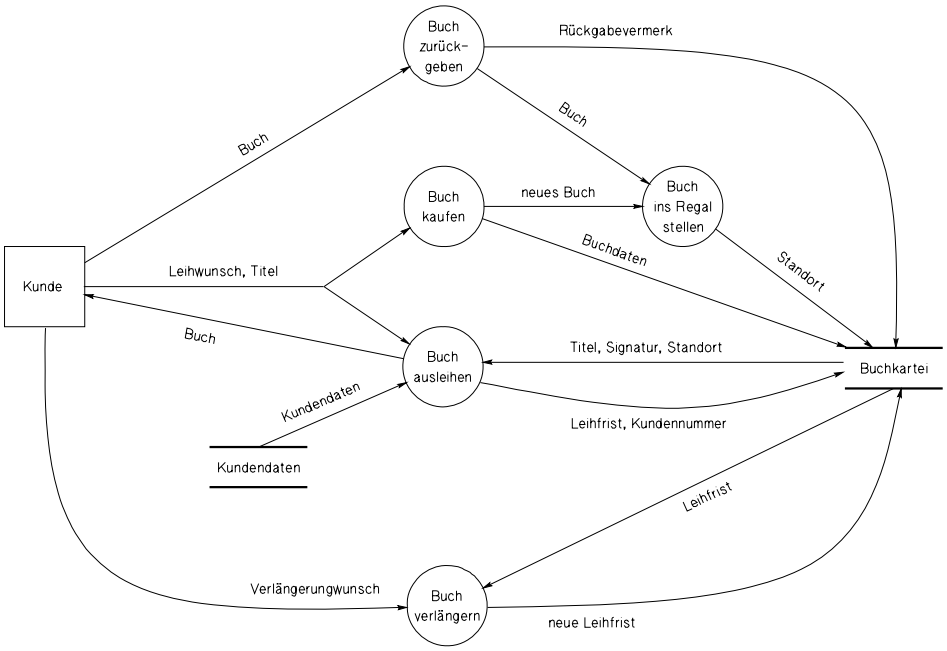

Abbildung 1: Daten-Fluß-Diagramm

Syntax und Präsentation

Ein DFD ist ein gerichteter Graph, der drei Arten von Knoten enthalten kann - Prozeßknoten, Datenspeicher und Endknoten. Prozeßknoten werden als Kreise dargestellt, Datenspeicher durch zwei waagrechte Balken und Endknoten als Quadrate. Die Kanten zwischen den Knoten heißen Datenflußkanten und können beschriftet werden. Jede Kante verbindet einen Prozeßknoten mit einem beliebigen anderen Knoten. Kanten zwischen

zwei Datenspeichern, zwei Endknoten oder einem Datenspeicher und einem Endknoten sind somit nicht erlaubt.

Die hier vorgestellte Präsentation eines DFD (Abb. 1) ist unter dem Namen "Yourdon-DeMarco Notation" bekannt. In der ebenfalls häufig verwendeten "Gane-Sarson Notation" werden Prozesse als Rechtecke mit gerundeten Ecken und Datenspeicher durch rechts offene Rechtecke dargestellt.

Semantik

Mit den vier Konstrukten eines DFD wird allgemein folgende Semantik wiedergegeben:

- Die Datenflußkanten repräsentieren den Fluß von Daten oder Material.
- Die Prozeßknoten repräsentieren Prozesse oder Aktivitäten, die einen oder mehrere Eingangsdatenflüsse in einen oder mehrere Ausgangsdatenflüsse transformieren.
- Datenspeicher dienen zur Repräsentation von Vorrichtungen zur Ablage von Daten. Man kann sich darunter Karteien, Dateien, Datenbanken oder jede andere Art von Informationssammlung vorstellen.
- Durch Endknoten (interface, terminator) werden Quellen und Senken beschrieben, also diejenigen Stellen eines Systems, an denen Informationen oder Gegenstände in das System einfließen oder wieder verlassen. Mit ihnen werden die Schnittstellen des Systems zu seiner Außenwelt dargestellt. Solche Schnittstellen können mit Personen, Organisationen, technischen Prozessen oder anderen Systemen außerhalb des betrachteten Kontextes bestehen.

Diese sehr allgemeine Beschreibung der Semantik kann noch weiter präzisiert werden. Wird mit einem DFD ein DV-System beschrieben, so werden die Endknoten als Computer-ein- und -ausgabegeräte (Tastatur, Bildschirm, Drucker, ...) interpretiert und die Datenflüsse als Beschreibung maschinell verwendeter Daten. Beschreibt ein DFD einen Fertigungsprozeß, so werden mit den Endknoten Mitarbeiter oder Maschinen dargestellt. Mit Datenflußkanten werden Daten in Form von Listen, Aufträgen oder Formularen wie auch Materialflüsse beschrieben.

Methodische Unterstützung

Ein Modell in SA wird mit Hilfe mehrerer DFD beschrieben. Die Diagramme eines Modells werden hierarchisch angeordnet [Ros77] . Als erstes wird das Kontextdiagramm (Abb. 2) erstellt, das an der Wurzel der Hierarchie steht. Es enthält genau einen Prozeß und die Schnittstellen des Systems zur Umwelt. In einem ersten Verfeinerungsschritt wird dieser Prozeß in Teilprozesse aufgeteilt, deren Datenflüsse untereinander sowie zu den Systemschnittstellen in einem Daten-Fluß-Diagramm beschrieben werden.

Diese Teilprozesse können weiter verfeinert werden, wobei jeder Verfeinerungsschritt in einem eigenen DFD dokumentiert wird und die einzelnen Diagramme entsprechend der Verfeinerung eine Diagramm-Hierarchie ergeben. Dabei dürfen Datenflüsse in tieferliegenden Diagrammen nur Verfeinerungen von Datenflüssen sein, die in übergeordneten

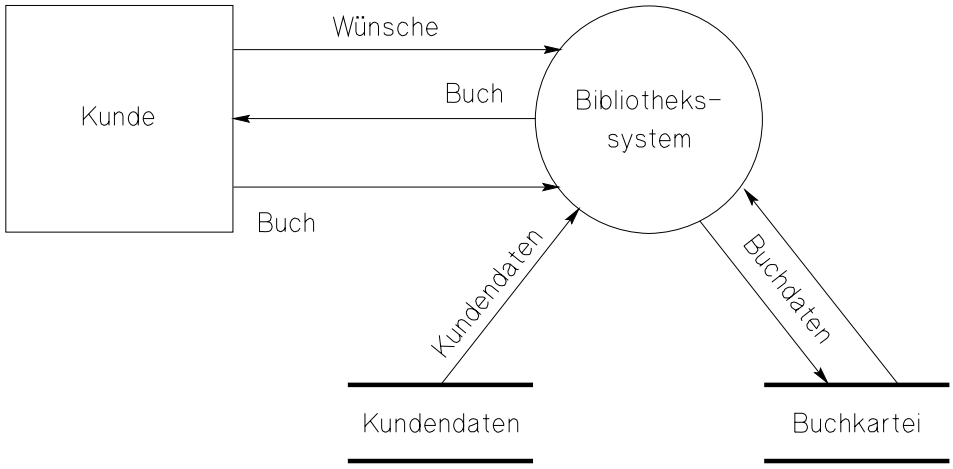

Abbildung 2: Kontext-Diagramm

Diagrammen bereits existieren. Der Verfeinerungsvorgang wird solange fortgeführt, bis das System im gewünschten Detailierungsgrad beschrieben ist.

Weiteres Vorgehen
Nach Fertigstellung der Hierarchie von Daten-Fluß-Diagrammen sind in SA zwei weitere Modellierschritte vorgesehen. Erstens werden mit Hilfe von regulären Ausdrücken alle Datenspeicher und alle Datenflüsse näher beschrieben. Diese Beschreibung wird in Backus-Naur-Form oder der DeMarco-Notation, einer Variante der BNF vorgenommen und in einem sogenannten Datenlexikon (data dictionary) abgelegt. Zweitens werden für jeden Prozeß eine Transformationsbeschreibung ("Minispec" genannt) angefertigt werden. Sie beschreibt wie der Prozeß die eingehenden Datenflüsse in die Ausgabe transformiert. Die Beschreibung selbst erfolgt in Umgangssprache oder in Pseudocode.

Das Datenlexikon und die Transformationsbeschreibungen enthalten bereits sehr detailierte Informationen über den betrachteten Weltausschnitt. Der Modellierer erhält ein Gesamtmodell, das vollständig genug ist, um es als Basis für die Entwurfsphase zu verwenden. Mit der Methode "Structured Design" können SA-Modelle in eine Systemarchitektur weiterentwickelt werden.

2.1.2 Strukturierte Analyse und Design Technik

Grundidee und Modelliertechnik
Die "Strukturierte Analyse und Design Technik" (SADT) [RS77, MM87] ist eine Modelliermethode, die einen ähnlichen Ansatz wie die "Strukturierte Analyse" verfolgt. Mit SADT erstellten Modellen liegt die Auffassung zugrunde, daß sich ein System im wesentlichen durch Aktivitäten und Daten charakterisieren läßt sowie den Wechselbeziehung zwischen Aktivitäten und Daten. SADT-Modelle werden mit Aktigrammen dargestellt.

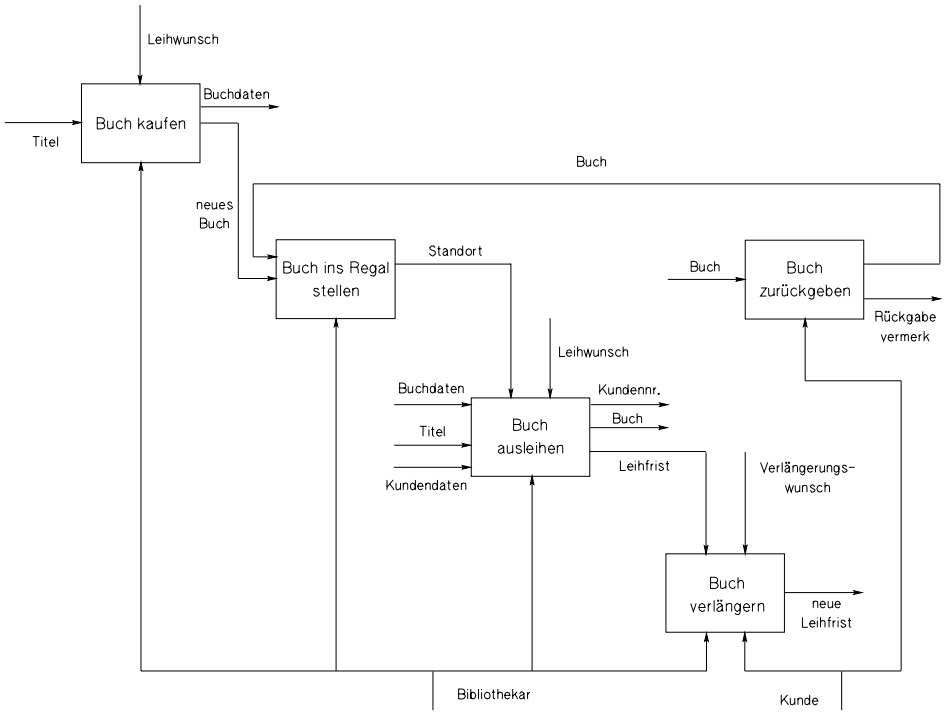

Abbildung 3: Aktigramm

Syntax und Präsentation
Ein Aktigramm (Abb. 3) besteht aus einer Menge von Aktivitäten, die durch Rechtecke dargestellt werden. Mit den Aktivitäten können beliebig viele Pfeile verbunden sein wobei zwischen vier verschiedenen Arten von Pfeilen unterschieden wird. Von links eingehende Pfeile werden Eingabe genannt, von oben eingehende Steuerung, von unten eingehende Mechanismen und von rechts ausgehende Ausgabe. Die Aktivitäten können miteinander verbunden werden, indem der Ausgabe-Pfeil einer Aktivität mit dem Eingabe- oder Steuerung-Pfeil einer anderen Aktivität verbunden wird.

Semantik
Eingabe-Pfeile repräsentieren Daten, die durch die Aktivität in Ausgangsdaten transformiert werden. Die Steuerung-Pfeile beschreiben Daten, die den Transformationsprozeß beeinflussen. Mechanismen-Pfeile repräsentieren Personen oder Programme, die die Aktivität ausführen oder sonstige Hilfsmittel, die zur Durchführung der Aktivität benötigt werden.

Methodische Unterstützung
Die Aktivitäten in SADT lassen sich genau wie die Prozesse in SA verfeinern. Die entstehenden Aktigramme bilden auch in SADT eine Hierarchie. SADT-Modelle werden top-down erstellt, beginnend mit einem Aktigramm das aus genau einer Aktivität und den Schnittstellen besteht.

Weiteres Vorgehen
SADT bietet die Möglichkeit, zwei Sichten eines Weltausschnittes zu modellieren. Die bereits vorgestellten Aktigramme zeigen eine Aktivitäten-Sicht. Die Daten-Sicht wird mit Datagrammen beschrieben. Die Syntax und die Präsentation der Datagramme entspricht exakt der Syntax und Präsentation der Aktigramme. Semantisch werden allerdings die Rollen von Daten und Aktivitäten vertauscht. Ein Rechteck repräsentiert Daten und die Pfeile beschreiben die Aktivitäten, die diese Daten erzeugen (Pfeile von links), verarbeiten (Pfeil von rechts ausgehend) oder steuern (Pfeil von oben). Die Pfeile von unten beschreiben die Speichermedien der Daten. Die Verfeinerung der Daten wird ebenfalls in einer Hierarchie von Datagrammen beschrieben.

2.1.3 Jackson Methoden

Grundidee und Modelliertechnik
In der Methode "Jackson Strukturierte Programmierung" (JSP) [Kil88, Sch90b] wird ein System als Verarbeitungsprozeß angesehen, dessen Struktur sich aus der Struktur der Eingabe- bzw. Ausgabedaten ableiten läßt. Die Methode "Jackson System Development" (JSD) [Cam86] geht davon aus, daß sich ein System als eine Anzahl sequentieller Prozesse darstellen läßt. Dabei werden jeweils die Prozesse zusammengefaßt, die von einem Objekt ausgeführt werden oder die dieses Objekt beeinflußen.

Beide Methoden verwenden zur Beschreibung ihrer Modell Jackson-Diagramme. In JSP wird damit die Struktur von Daten beschrieben; in JSD die möglichen Reihenfolgen von Prozessen.

Syntax und Präsentation
Ein Jackson-Diagramm ist ein Baum, dessen Wurzelknoten ein Objekt festlegt. Mit einem Jackson-Diagramm sind drei verschiedenen Kontroll- bzw. Datenstrukturen beschriebbar:

- Eine Sequenz ist durch eine oder mehrere Komponenten gekennzeichnet, die jeweils genau einmal in einer vorgegebenen Reihenfolge auftreten können. In einem Jachson-Diagramm bilden alle Knoten, die auf derselben Baumebene stehen, eine Sequenz. Die Reihenfolge ist explizit von links nach rechts festgelegt.

- Eine Auswahl besteht aus zwei oder mehr Komponenten, von denen bei jedem Auftreten der Auswahl genau eine gewählt wird. Die alternativen Komponenten werden im Diagramm rechts oben mit einem kleinen Kreis markiert.

- Eine Wiederholung besteht aus einer Komponente, die keinmal oder mehrmals auftreten kann. Diese Komponenten werden mit einem Stern rechts oben markiert.

Semantik
Sowohl in JSP als auch in JSD repräsentiert die Wurzel des Baumes ein Objekt des betrachteten Systems. Alle anderen Knoten im Baum repräsentieren in JSP Datenelemente,

die das Objekt beschreiben. In JSD werden entweder die Prozesse modelliert, die von einem Objekt ausgeführt werden oder die dieses Objekt beeinflußen (Abb. 4). Im zweiten Fall wird der Einfluß von Prozessen in Form von Zustandstransformationen modelliert. Die Knoten eines entsprechenden Jackson-Diagramms (Abb. 5) repräsentieren dann die möglichen Zustände eines Objektes. Mit den Kontrollstrukturen werden die gültigen Zustandsübergänge definiert.

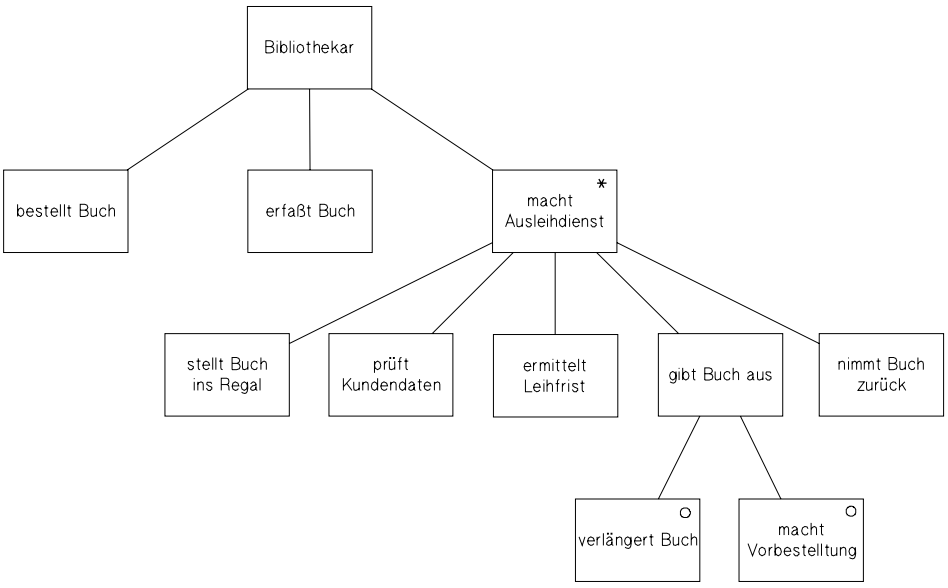

Abbildung 4: Jackson-Diagramm Prozeßfolgensemantik (JD-P)

Methodische Unterstützung

In JSD wird vorgeschlagen, vor der Erstellung eines Jackson-Diagrammes alle relevanten Objekte und alle Aktionen des Systems zu ermitteln und ungeordnet zusammenzustellen. Daraufhin werden die Aktionen den Objekten zugeordnet und durch die passenden Kontrollstrukturen als Jackson-Diagramm beschrieben.

Weiteres Vorgehen

JSP ist eigentlich eine Methode, die in der Entwurfsphase der Softwareentwicklung eingesetzt wird. Sie wurde an dieser Stelle lediglich erwähnt, um ein Beispiel dafür zu geben, daß eine Technik in mehreren Methoden mit unterschiedlicher Semantik verwendet werden kann.

In JSD wird dagegen in mehreren Modellierschritten ein konzeptuelles Modell erstellt, in dem nicht nur Prozesse, Prozeßfolgen und die in diese Prozesse involvierten Objekte modelliert werden. Mit anderen Techniken werden auch noch Schnittstellen zur Umwelt beschrieben sowie Ereignisse, die das System beeinflußen, d.h. Prozesse des Systems aktivieren.

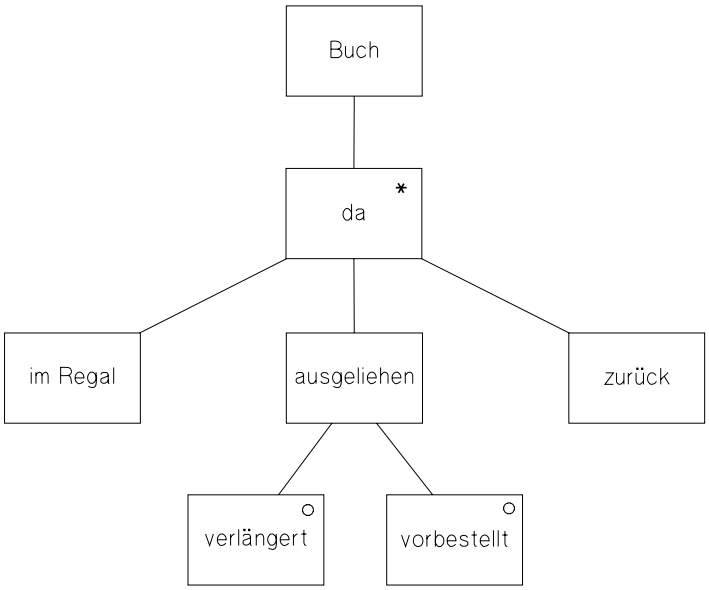

Abbildung 5: Jackson-Diagramm Zustandsfolgensemantik (JD-Z)

2.1.4 Petri-Netze

Grundidee und Modelliertechnik
"Petri-Netze" [Rei90] sind eine Modelliertechnik, mit der sich vorallem die Dynamik von Systemen modellieren, analysieren und simulieren läßt. Sie werden in zahlreichen Modelliermethoden verwendet.

Syntax und Präsentation
Ein Petri-Netz ist ein gerichteter Graph mit zwei verschiedenen Arten von Knoten - Stellen und Transitionen. Eine Stelle wird durch einen Kreis dargestellt, eine Transition durch ein Rechteck. Die Kanten dürfen nur von Knoten einer Art zu Knoten der anderen Art führen. Alle Stellen, von denen aus eine Kante zu einer Transition t führt heißen Vorbereichsstellen der Transition t. Alle Stellen, die mit einer auslaufenden Kante von t verbunden sind werden Nachbereichsstellen genannt. Die Stellen eines Petri-Netzes können mit Marken belegt werden, die als kleine, schwarze Kreise in die Stellen eingetragen werden.

Es gibt viele verschiedenen Arten von Petri-Netzen, die sich vorallem dadurch unterscheiden, mit wievielen Marken und mit welchen Arten von Marken eine Stelle belegt werden kann. Bei dem in Abbildung 6 verwendeten Bedingungs-Ereignis-Netz ist nur eine Marke pro Stelle erlaubt. Bei Stellen-Transitionen-Netzen kann eine Stelle mit beliebig vielen Marken belegt werden. Es kann aber auch eine Höchstanzahl festgelegt werden. Bei den sogenannten CP-Netzen (Coloured Petri Nets) wird zwischen unterschiedlichen Arten von Marken unterschieden.

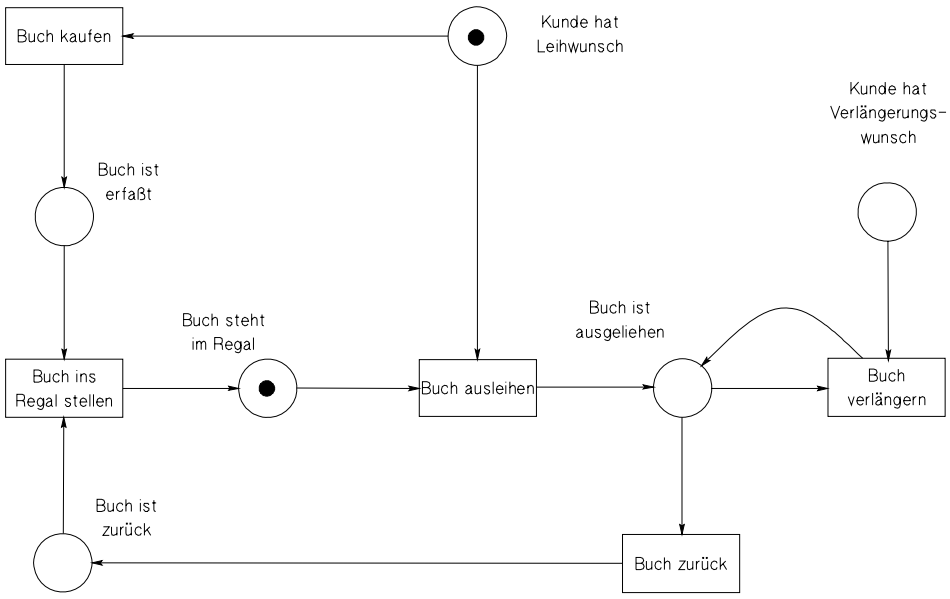

Abbildung 6: Bedingungs-Ereignis-Netz

Zur Simulation der Dynamik eines Systems werden für jede Art von Petri-Netz Schaltregeln definiert, die festlegen, wie sich die Marken durch ein Netz bewegen können. Für ein Bedingungs-Ereignis-Netz lauten diese wie folgt:

- Eine Transition t kann schalten, wenn jede Vorbereichsstelle von t eine Marke enthält.
- Schaltet eine Transition, dann wird aus jeder Vorbereichsstelle die Marke entfernt und jede Nachbereichsstelle mit einer Marke versehen.

Semantik
Eine Stelle beschreibt eine Zwischenablage von Informationen; eine Transition die Verarbeitung von Informationen. Die Marken dienen zur Beschreibung der Dynamik des modellierten Systems. Eine Marke in einer Stelle eines Bedingungs-Ereignis-Netzes bedeutet semantisch, daß die durch die Marke beschriebene Information aktuell vorliegt. Die erste Schaltregeln besagt daher, das eine Transition schalten kann d.h. der beschriebene Verarbeitungsprozeß durchgeführt werden kann, wenn alle hierfür benötigten Informationen vorliegen. Dadurch werden neue Informationen erzeugt. Dies wird durch Markieren der Nachbereichsstellen der Transition dokumentiert (Schaltregel 2).

Damit die Simulation eines Petri-Netzes überhaupt starten kann, müssen bereits einige Stellen markiert sein. Die Menge dieser Marken wird Anfangsmarkierung genannt. Die anfangsmarkierten Stellen repräsentieren den Anfangszustand des Systems und stellen implizit die Schnittstelle zur Umwelt dar.

In dem Beispiel in Abbildung 6 sind die beiden Stellen "Kunde hat Leihwunsch" und "Buch steht im Regal" markiert. Mit dieser Anfangsbelegung kann der Ausleihvorgang eines Buches simuliert werden. Die Transition "Buch ausleihen" kann schalten. Wird der Schaltvorgang durchgeführt, so werden die Marken der Anfangsbelegung abgezogen und die Stelle "Buch ist ausgeliehen" mit einer Marke versehen. Daraufhin können weitere Transitionen schalten und somit nach und nach das gesamte Systemverhalten schrittweise simuliert werden.

Methodische Unterstützung

Zur Erstellung eines Petri-Netzes wird empfohlen [Par91], die Bedingungen zu ermitteln (d.h. mit den Stellen zu beginnen), die den Anfangszustand repräsentieren. Daraufhin sollen alle Ereignisse ermittelt werden (d.h. alle Transitionen modelliert werden), die auf Grund des Anfangszustands eintreten können (d.h. auf Grund der Anfangsbelegung schalten können). Daraufhin werden die Nachbedingungen dieser Ereignisse modelliert, die gleichzeitig wieder Vorbedingungen für weitere Ereignisse sind.

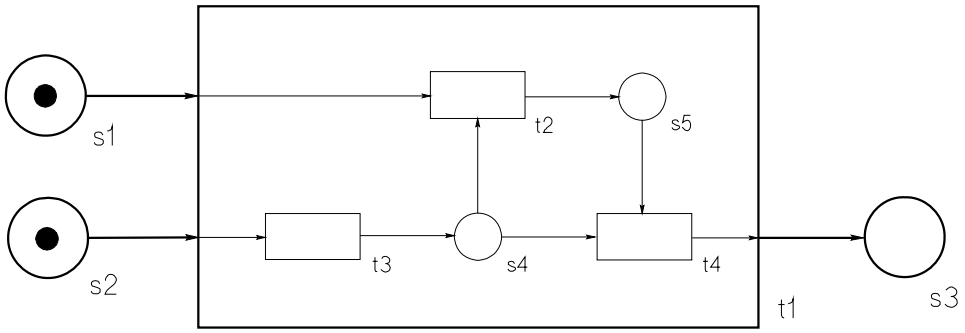

Abbildung 7: Inkonsistente Verfeinerung von Petri-Netzen

Eine Transition beschreibt, wie eine Aktivität in SADT oder ein Prozeß in einem DFD, die Verarbeitung von Informationen. Daher ist es naheliegend zu versuchen, Petri-Netze durch schrittweise Verfeinerung von Transitionen zu erstellen. Dies ist allerdings nicht uneingeschränkt möglich, da die Schaltregeln nicht nur auf das 'oberste' Petri-Netz anwendbar sein müssen, sondern auch auf das Gesamtnetz einschließlich der Verfeinerungen. Abbildung 7 zeigt ein Beispiel, in dem eine Transition vor der Verfeinerung (fett und groß dargestelltes Petrinetz mit der Transition t1 und den Stellen s1-s3) schalten kann, nach der Verfeinerung (das Gesamtnetz mit t2-t4 und s1-s5) aber nicht mehr, da sowohl t2 als auch t4 zum Schalten eine Marke in s4 brauchen. Daher werden z.B. in [Sch90a] Regeln definiert, wie Verfeinerungen von Petri-Netzen korrekt gebildet werden können.

Weiteres Vorgehen

Nach der Beschreibung eines Systems als Petri-Netz kann es unter verschiedenen Gesichtspunkten analysiert werden. Es läßt sich z.B. feststellen, ob jede Transition lebendig ist oder ob das System vermeidbare Verklemmungen enthält. Ein Transition ist lebendig, wenn ausgehend von einer festgelegten Anfangsmarkierung die Transitionen stets so schal-

ten, daß diese Transition im weiteren Verlauf nochmals schalten kann. Eine vermeidbare Verklemmung ist eine Situation, in der keine Transition schalten kann, was aber bei einer anderen Schaltreihenfolge hätte vermieden werden können.

2.1.5 Entity-Relationship-Modellierung

Grundidee und Modelliertechnik
Mit Entity-Relationship-Modellen [EN89] können Objekte bzw. Entitäten und ihre statische Beziehungen zueinander beschrieben werden. Sie werden als Entity-Relationship-Diagramm (ER–Diagramm) dargestellt.

Syntax und Präsentation
Ein ER–Diagramm ist ein zusammenhängender, ungerichteter Graph mit drei verschiedenen Knotenarten – Entitätstypen (E-Knoten), Relationshiptypen (R-Knoten) und Attribute. In einem ER–Diagramm (Abb. 8) werden E-Knoten als Rechtecke, R-Knoten als Rauten und Attribute als Ovale dargestellt. Ein R-Knoten muß mindestens zwei Kanten zu E-Knoten haben. Ein Attribut ist mit genau einem E-Knoten oder einem R-Knoten verbunden.

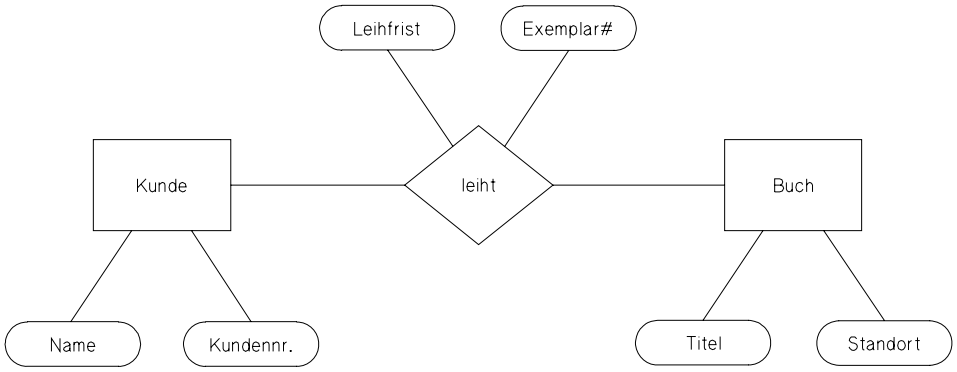

Abbildung 8: Entity-Relationship-Diagramm

Semantik
Eine Entität ist ein individuelles und identifizierbares Exemplar von Dingen, Personen oder Begriffen der realen oder der Vorstellungswelt. Um die Komplexität der betrachteten Welt zu reduzieren wird nach Entitäten gesucht, die strukturell ähnlich sind. Aufgrund dabei festgelegter Ähnlichkeitskriterien, werden Klassen von Entitäten gebildet. Alle Entitäten einer solchen Klasse sind vom selben Typ und werden in einem ERM durch einen Entitätstypen repräsentiert. Durch die Attribute werden die Merkmale repräsentiert, die zur Beschreibung der Ähnlichkeit der einzelnen Entitäten wichtig sind bzw. eine Unterscheidung von Entitäten desselben Typs ermöglichen. Ein Relationsship-Knoten repräsentiert eine Klasse von ähnlichen Beziehungen zwischen zwei Entitäten. Eine Beziehung wird identifiziert durch die beiden Entitäten und ggf. durch weitere Attribute.

Ausgehend von dieser Grundsemantik sind zahlreiche Erweiterungen für die ER-Modellierung entwickelt worden [Mar83, KL87, Sin88, PS89]. Mit Hilfe weiterer Modellkonstrukte können in Modellen dieser Techniken Teilmengen-, Rollen-, Subtypenbeziehungen [SPYA91] und verschiedene Integritätsbedingungen [HW93] dargestellt werden. [STH90] gibt einen Überblick über die wichtigsten Konstrukte.

Methodische Unterstützung
In der Originalarbeit von Chen [Che76] wird keine Methode zum Erstellen von ER-Modellen beschrieben. Seither wurden zahlreiche Vorschläge zu dieser Problematik gemacht, von denen hier nur die prinzipellen Ideen kurz erwähnt werden können.

Ein ERM wird im allgemeinen durch schrittweise Verfeinerung aller drei Modellkonstrukte erstellt. Allerdings besteht ein ERM nur aus einem einzigen Diagramm. Ein verfeinertes Konstrukt wird nicht wie in SA in einem eigenen Diagramm beschrieben, sondern aus dem Diagramm gestrichen und durch seine Verfeinerung ersetzt. Durch diese Transformationen [BCN91] ist es nicht mehr möglich, den Modellierprozeß nachzuvollziehen. Da die Größe der Modelle und der Diagramme schnell wächst, wird häufig in einem ersten Modellierschritt auf die Modellierung der Attribute verzichtet.

Eine andere Möglichkeit ist es, zuerst ein ERM zu erstellen, das den ganzen betrachteten Weltausschnitt auf sehr abstrakten Niveau beschreibt. In diesem ERM werden Teilausschnitte identifiziert und anschließend durch mehrere zusätzliche ER-Modell beschrieben. Diese Teilmodelle müssen nach ihrer Fertigstellung wieder zu einem Gesamtmodell (Gesamtdiagramm) zusammengesetzt werden. Die damit verbundene Problematik ist unter dem Stichwort "Sichten-" oder "Schemaintegration" beschrieben [BLN86].

Es gibt weitere Vorschläge [FM86, CJA89, ZNG+90] zu Methoden, die durch die Einführung neuer Modellkonstrukte Möglichkeiten schaffen, ER-Modelle zu strukturieren.

Weiteres Vorgehen
Vor allem um die Weiterverarbeitung in der Entwurfsphase zu ermöglichen, können die ER-Modelle zum Beispiel durch die Angabe von Schlüsselattributen, Wertebereichen für die Attribute und Komplexitätsgraden [EN89] erweitert werden. In der Entwurfsphase können dann aus den ER-Modellen Datenbankschemata der verschiedenen Datenmodelle (hierarchisch, Netzwerk, relational, objekt-orientiert) abgeleitet werden [BCN91].

2.2 Modelliertechniken

Im den vorangegangenen Abschnitten wurden einige Modelliertechniken vorgestellt — Daten-Fluß-Diagramme, Data-Dictionaries, Jackson-Diagramme, Petri-Netze und Entity-Relationship-Diagramme. Diese Techniken werden Basistechniken genannt, da sie sich nicht weiter auf andere Basistechniken reduzieren lassen. Als weitere Basistechniken werden in [Bal91, Par91] z.B. Entscheidungstabellen, Zustandsübergangsbeschreibungen und Programmablaufpläne erwähnt. Diese Basistechniken sind besonders wichtig, da sie, zum

Teil mit anderer Syntax-, Präsentations- oder Semantikbeschreibung, in den meisten Modelliermethoden verwendet werden.

2.2.1 Basistechniken

Ein Vorteil der erwähnten Basistechniken ist, neben der einfachen Syntax und der meist grafischen Präsentation, daß sie sich auf die Modellierung weniger Aspekte konzentrieren. Dadurch erhält man einfache, kompakte und leicht verständliche Modelle. Allerdings läßt sich mit einem einzelnen Diagramm nur ein kleiner Ausschnitt der realen Welt beschreiben. Je nach geplanter Verwendung der konzeptuellen Modelle (Software-Entwicklung, Datenbank-Entwurf, Simulationen, Metadaten-Informationssystem, etc.) muß zur Beschreibung eines Weltausschnittes eine Vielzahl von Aspekten berücksichtigt und beschrieben werden.

Für den Entwurf einer Datenbank müssen zum Beispiel Objekte und deren Beziehungen zueinander modelliert werden. In diesen Fällen genügt es unter Umständen, nur ein Entity-Relation-Modell (ERM) zu erstellen. Für die Entwicklung einer Datenbank-Applikation müssen auch Prozesse modelliert werden, die Daten erzeugen, in der Datenbank speichern oder Datenbankanfragen durchführen. Zur Überprüfung der Konsistenz einer Datenbank ist die Beschreibung von Objektzuständen und der gültigen Zustandsübergänge wichtig.

Aspekt	DFD	AG	DG	JD-P	JD-Z	B/E-Netz	ERD
Objekt			x		x		x
Beziehung							x
Datensammlung	x		x				
Schnittstelle	x						
Prozeß	x	x		x		x	
Prozeßfolge				x		x	
Aktor		x		x			
Zustand					x	x	
Zustandsfolge					x		

Tabelle 1: semantische Aspekte

Bevor also mit der Modellierung begonnen werden kann, sollte festgelegt werden, welche Aspekte des Weltausschnittes modelliert werden müssen. Tabelle 1 gibt eine kleine Übersicht über verschiedene Basistechniken und die wichtigsten, mit ihnen beschreibbaren, semantische Aspekten. Bei der Vorstellung von JSD wurden zwei verschiedene Semantiken mit Jackson-Diagrammen dargestellt. JD-P bezieht sich auf die Semantik alle Prozesse und Prozeßabläufe darzustellen, die ein Objekt (Aktor) ausführt. JD-Z bezieht sich auf die Zustandssemantik. "AG" und "DG" sind die Abkürzungen für die beiden Modelliertechniken von SADT – Aktigramme und Datagramme.

2.2.2 Kombinierte Techniken

Für die Modellierung einer Datenbank-Applikation muß eine Menge von Aspekten modelliert werden, die, wie aus Tabelle 1 ersichtlich ist, mit keiner der Basistechniken darstellbar sind. Daher wurden zahlreiche kombinierte Techniken entwickelt (ACM/PCM [BS82], BIER [EKTW86, KS88], ORM [Per90]). Sie verwenden Konzepte aus mehreren Basistechniken und verbinden sie zu einer eigenständigen Technik. BIER ist zum Beispiel eine Kombination aus Petri-Netzen und Entity-Relationship Modellen; ORM verbindet Zustands-Übergangs-Diagramme und ER-Modelle.

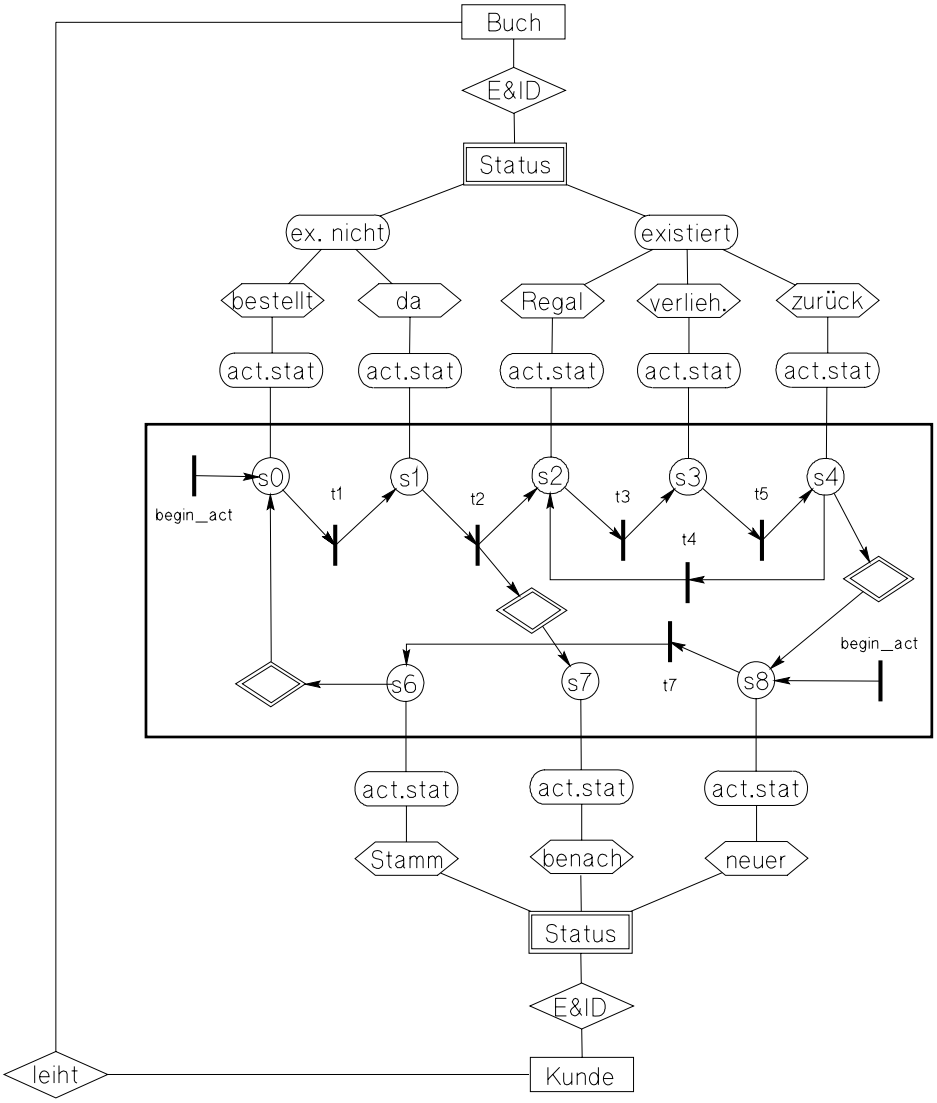

Abbildung 9: Behavior-Integrated-Entity-Relationship-Diagramm

Als Beispiel für eine kombinierte Technik wird im folgenden der BIER-Ansatz (Behavior Integrated Entity-Relationship Approach) [EKTW86] kurz umrissen [1]. Ein BIER-Diagramm (Abb. 9) ist aufgeteilt in einen äußeren und einen inneren Bereich. Im äußeren Bereich wird der statische Aufbau eines Systems als erweitertes Entity-Relationship Diagramm dargestellt. Als Modellkonstrukte stehen zur Verfügung :

- Unabhängige Entitytypen, semantische Relationship-Typen und Attribute mit derselben Syntax und Semantik wie die Basistechnik.
- Existenzabhängige Entity-Typen (weak entity), die über eine identifizierende Beziehung mit einem unabhängigen Entity-Typen (strong entity) verbunden sind. Hiermit werden Objekte beschrieben, die ohne die Zugehörigkeit zu einem anderen Objekt für das System nicht von Interesse sind (die Kinder eines Angestellten).
- Sechs weitere Arten von Relationshiptypen, mit denen Teilmengen-, Rollen- und Untertypenbeziehungen zwischen Objekten repräsentiert werden können, sowie
- Zustände, mit denen die verschiedenen Rollen eines Objektes näher beschrieben werden können.

Im inneren Bereich wird die Dynamik des Systems beschrieben. Für jeden Entitytypen wird ein eigenes Petri-Netz modelliert. Dazu wird für jeden Zustand eines Entitytypen eine Petri-Netz-Stelle erzeugt und durch eine "ist Zustand von" Beziehung mit der Rollenbeschreibung des statischen Modellteils verbunden. Die Transitionen der Petri-Netze repräsentieren Prozesse, die die entsprechenden Zustandsübergänge vornehmen. Zur Verknüpfung der einzelnen Petri-Netze stehen drei weitere Arten von Relationshiptypen zur Verfügung. Mit diesen Beziehungen können die Zusammenhänge zwischen den Zuständen verschiedener Objekte beschrieben werden.

Mit kombinierten Techniken können somit semantisch umfangreichere Modelle erstellt werden, da eine größere Zahl von Aspekten darstellbar ist. Allerdings ist die Handhabung dieser Modelle, wie bereits dieses kleine Beispiel zeigt, aufgrund der wesentlich komplexeren Syntax und Semantik deutlich schwieriger als bei Modellen von Basistechniken.

2.3 Modelliermethoden

Für viele Anwendungen müssen bei der Modellierung so viele Aspekte berücksichtigt werden, daß ein Modell einer Einzeltechnik (Basistechnik oder kombinierte Technik) den Weltausschnitt nicht ausreichend beschreibt. In solchen Fällen müssen verschiedene Techniken ausgewählt und angewendet werden. Die dabei entstehenden Modelle müssen alle für die Beschreibung der Anwendung benötigten Aspekte enthalten. Ein Problem stellt hierbei die Konsistenz der verschiedenen Einzelmodelle dar. In Tabelle 1 fällt auf, daß

[1] Auf die explizite Erläuterung des Beispiels (Abb. 9) wird an dieser Stelle aus Platzgründen verzichtet. Eine ausführliche Beschreibung der verschiedenen Modellkonstrukte ist in [EKTW86] zu finden.

es zahlreiche Überlappungen von Aspekten zwischen den verschiedenen Techniken gibt. Wurden zu einem Weltausschnitt zwei Modelle mit verschiedenen Techniken erstellt, die überlappende Aspekte beinhalten, so bedeutet das, daß gewisse Teile der Einzelmodelle inhaltlich denselben Teil des Weltausschnittes beschreiben. Daher muß die Semantik bzgl. dieser Aspekte in beiden Einzelmodellen gleich sein. Das Semantik des Gesamtmodells ergibt sich damit aus der 'Vereinigung' der Interpretationen der verschiedenen Einzelmodelle. Die Einzelmodelle selbst sind lediglich verschiedenartige Sichten des Gesamtmodells und werden im folgenden als Teilmodelle bezeichnet.

2.3.1 Inkrementelle Modellierung

Eine einfache Möglichkeit, ein konsistentes Gesamtmodell zu erstellen, ist die inkrementelle Modellierung. Hierbei werden mehrere Modelliertechniken nacheinander angewendet. Für jeden Übergang von einer Technik zur nächsten wird beschrieben, welche Aspekte der bis dahin erarbeiteten Modelle in das nächste übernommen und dort näher beschrieben werden. Damit wird eine mehrfache und damit potentiell inkonsistente Modellierung derselben Aspekte in den verschiedenen Modelliertechniken vermieden. Die Ergebnisse der verschiedenen Modellierschritte werden sukzessive zu einem konsistenten Gesamtmodell zusammengesetzt. Es kann bei der inkrementellen Modellierung eigentlich nicht von einer Erstellung von Teilmodellen und deren Integration gesprochen werden. Vielmehr werden durch die Anwendung weiterer Techniken gewisse Aspekte zum bereits existierenden Gesamtmodell hinzugefügt.

In einem vorausgegangenen Abschnitt wurde die Methode "SA" vorgestellt. Hierbei handelt es sich um eine inkrementelle Modelliermethode mit drei Modellierschritten. Zuerst wird eine Hierarchie von Daten-Fluß-Diagrammen erstellt. In zwei weiteren Modellierschritten werden alle DFD's in den Blättern der Diagrammhierarchie betrachtet und einige ihrer Modellkonstrukte näher beschrieben. Die Prozeßknoten, die in diesen DFD's vorkommen repräsentieren Elementarprozesse, die nicht weiter verfeinert werden können. Die Verarbeitungslogik dieser Elementarprozesse wird in einem weiteren Modellierschritt in Form von Pseudocode oder rein verbal beschrieben. In einem anderen Modellierschritt wird die Struktur der Daten bzw. Objekte, die durch die Datenflüsse und durch die Datenspeicher repräsentiert werden, in Form von regulären Ausdrücken beschrieben.

In SA können also Objekte und deren Struktur modelliert werden, allerdings keine Beziehungen zwischen den verschieden Objekten. Daher wurde mit "SA-ER" [CN86] eine Modelliermethode entwickelt, mit der zusätzlich zu den in SA modellierbaren Aspekten auch noch dieser Aspekt modelliert werden kann. In SA-ER wird wie in SA zuerst eine Hierarchie von Daten-Fluß-Diagrammen erstellt. Als nächster Modellierschritt wird wiederum die Struktur der durch die Datenflüsse repräsentierten Objekte beschrieben. Allerdings nicht in Form von regulären Ausdrücken, sondern in Form von ER-Modellen. Es wird also für jeden Datenfluß in einem Blattdiagramm ein kleines ER-Modell erstellt. In einem dritten Modellierschritt werden die im zweiten Schritt entstandenen ER-Modelle zu einem Gesamt-ER-Modell integriert. Dafür wird die DFD-Hierarchie von den Blättern zur Wur-

zel Ebene für Ebene durchlaufen. Zu jedem DFD gehört eine Menge kleiner ER-Modelle, die mit Techniken der Schemaintegration [BLN86] zu einem einzigen Modell zusammengefaßt werden. Da die Datenflüsse eines untergeordneten DFD's nur eine Verfeinerung von Datenflüssen der übergeordneten Diagramme sein dürfen, kann die Integration von ER-Teilmodellen auf allen Ebenen der Hierarchie bis zur Wurzel durchgeführt werden. Resultat dieses Integrationsprozesses ist ein ER-Modell, das alle für dieses System relevanten Objekte (Entity-Typen) und ihre Beziehungen (Relationship-Typen) zueinander beschreibt. Mit den in SA-ER vorgeschlagenen Modellierschritten entsteht ein konsistentes Gesamtmodell eines Systems, in dem mehr semantische Aspekte berücksichtigt sind als in einem SA-Modell.

Ein semantisch ähnlich umfassendes Modell läßt sich mit einer Methode entwickeln, die in [Bai89] vorgestellt wird. In dieser Methode wird allerdings zuerst ein ER-Modell erstellt und dann in sechs weiteren, zum Teil mehrfach auszuführenden, Modellierschritten eine Hierarchie von speziellen Daten-Fluß-Diagrammen. In dieser Arbeit werden noch weitere Vorschläge, wie das durch die ersten sieben Schritte entstandene Modell um weitere semantische Aspekte wie Zustände, Ereignisse, die durch eine Umweltschnittstelle auf das System einwirken (events), Zustandsübergänge etc. erweitert werden kann.

Die vorgestellten Methoden zeigen jeweils Beispiele, wie durch die inkrementelle Modellierung mit mehreren Basistechniken semantisch ausdrucksstärkere Modelle erstellt werden können.

2.3.2 Objekt-orientierte Ansätze

Die meisten objekt-orientierten Modelliermethoden [CJ90, RBP+91, SM92] basieren ebenfalls auf der Idee der inkrementellen Modellierung. Sie unterscheiden sich von den bereits vorgestellten Methoden hauptsächlich in der Sicht oder der Vorgehensweise [RBP+91]. Bei vielen klassischen Methoden (SA,SADT,PN) wird ein System als Prozeß dargestellt, der nach dem Verfeinerungsprinzip immer detailierter beschrieben wird. Bei den objekt-orientierten Methoden wird eher Bottom-up vorgegangen. Es werden Objekte gesucht, die zum System gehören [RG92]. Diese Objekte werden statisch einerseits durch zugehörige Attribute beschrieben. Andererseits werden aber auch die Beziehungen zwischen den verschiedenen Objekten modelliert. Diese Vorgehensweise entspricht ziemlich genau der ER-Modellierung und wird in vielen objekt-orientierten Methoden wie OMT [RBP+91] oder OOA [CJ90] auch ähnlich dargestellt. Zur Beschreibung der Dynamik des Systems werden den Objekten Methoden zugeordnet. Diese Methoden bewirken meist eine Zustandsänderung des Objekts [Boo86]. In OMT und in OOA werden Zustands-Übergangs-Diagramme zur Beschreibung der Interaktion zwischen den Objekten verwendet. In ORM [Per90] werden alle gültigen Zustände (dort Rollen genannt, um zum Ausdruck zu bringen, daß ein Objekt auch mehrere Rollen gleichzeitig haben kann) eines Objektes aufgelistet und mit Regeln die gültigen Zustands-Übergänge festgelegt. In JSD [MP86, Cam86] als klassischer Methode werden, wie bei den objekt-orientierten Methoden, die relevanten Objekte und Prozesse gesucht und jedem Objekt die Prozesse zugeordnet, die es ausführen kann.

3 Computer-gestützte Modellierung

3.1 Modellierwerkzeuge

Zur Unterstützung der konzeptuellen Modellierung sind zahlreiche computer-gestützte Werkzeuge entwickelt worden und werden ständig weitere entwickelt [Bal91]. Haupteinsatzgebiet der Werkzeug ist die Software-Entwicklung. Daher werden diese Werkzeuge meist unter dem Begriff CASE-Tools (Computer Aided Software Engineering-Tools) zusammengefaßt. Im folgenden werden einige Funktionen von CASE-Tools beschrieben und diskutiert,welche Erleichterung sie für einen Software-Entwickler bieten. Dabei wird hauptsächlich von Werkzeugen ausgegangen, die in der Analysephase der Software-Entwicklung eingesetzt werden und genau eine Modelliertechnik unterstützen. Werkzeuge, mit denen Modelle in verschiedenen Techniken erstellt werden können, werden als Modelliersysteme gezeichnet; deren Funktionalität wird in Abschnitt 3.2 beschrieben.

3.1.1 Grafische Editoren

In der Analysephase werden Modelle der geplanten Applikation erstellt. Die Beschreibung dieser Modelle wird mit Hilfe von Editoren in den Computer eingegeben. Die Art des Editors hängt von der Notation der verwendeten Technik ab; so kann z.B. für die Eingabe eines "Data-Dictionaries" ein Texteditor verwendet werden, wie er auch zur Eingabe des Quellcodes eines Programmes verwendet wird. Viele der zuvor vorgestellten Modelliertechniken verfügen über eine grafische Notation. Die hierfür benötigten Grafikeditoren haben ähnliche Funktionen wie Grafikprogramme zum Erstellen von Schaubildern oder CAD-Programmen.[2]

Ein grafischer Modelliereditor ermöglicht dem Benutzer die Eingabe von grafischen Grundsymbolen und deren Plazierung auf dem Bildschirm des Rechners. Als Symbole werden nur die angeboten, die in der Notation der unterstützten Modelliertechnik vorgesehen sind. Zum Zeichnen der Diagramme steht meist eine 'unendlich' große Zeichenfläche zur Verfügung von der der Rechnerbildschirm lediglich einen Ausschnitt anzeigt. Damit entfällt die Aufteilung eines Diagramms auf mehrere Blatt Papier. Die Größe und die Lage des angezeigten Ausschnittes kann beliebig gewählt werden. Modelliereditoren, die auf Basis einer grafischen Benutzeroberfläche (Windows, Motif, etc.) arbeiten, ermöglichen die Anzeige von mehreren Ausschnitten des Diagramms.

Eine einfache aber sehr wichtige Funktion ist das Verschieben von Symbolen. Während der Eingabe eines Diagramms kommt es immer wieder vor, daß ein Symbol an eine Stelle plaziert werden soll, die bereits von einem oder mehreren anderen Symbolen belegt ist. In

[2] Um Mißverständnissen vorzubeugen sei darauf hingewiesen, daß in dieser Arbeit exakt zwischen einer Grafik und einem Graphen unterschieden wird. Eine Grafik ist die Darstellung eines Sachverhalts mit Hilfe von Linien, Kurve und Zeichnungen wogegen es sich bei einem Graphen um eine mathematische Struktur handelt, die z.B. mit Kreisen und Linien grafisch dargestellt werden kann.

diesen Fällen müssen zuerst einige Symbole auf der Zeichenfläche verschoben werden, um den 'Platz' für das neue Symbol frei zu machen. Dieser Vorgang kann vorallem bei großen Diagrammen sehr aufwendig werden. Daher wäre es wünschenswert, wenn der grafische Editor eine Funktion zum automatischen Darstellen von Diagrammen hätte. Es gibt einige Forschungsarbeiten [BTT84, BNT86, PSTS91] die Algorithmen für solch ein Funktion vorschlagen. Diese Algorithmen sind sehr komplex, da zahlreiche Kriterien berücksichtigt werden müssen [BFN85, NT90], wenn ein Diagramm gut, d.h übersichtlich und gut lesbar dargestellt werden soll. Aus diesem Grund wird die Funktion in kommerziellen Systemen bisher nicht eingebaut.

3.1.2 Prüffunktionen

CASE-Tools sollten über Funktionen verfügen, mit denen sich die Richtigkeit von Modellen prüfen läßt. Die Möglichkeiten ein Modell auf Richtigkeit zu prüfen sind stark von der gewählten Modelliertechnik abhängig. Im folgenden werden beispielhaft einige Eigenschaften von Modellen genannt und erläutert, wie sie von einem CASE-Tool geprüft werden können.

- Grapheigenschaften
  ER-Modelle, Petri-Netze und DFD sind Graphen mit verschiedenen Knoten- und Kantenarten. Für jede dieser Techniken ist genau festgelegt (siehe Syntaxbeschreibungen in Abschnitt 2.1), welche Knotenarten durch Kanten miteinander verbunden werden dürfen. Zusätzlich ist auch noch die Gesamtstruktur des Modellgraphen definiert. ER, PN und DFD verlangen, daß der Modellgraph zusammenhängend ist. Zur Prüfung dieser Eigenschaft genügt es nicht, den Modellgraphen auf isolierte Knoten[3] zu untersuchen, da in einem zusammenhängenden Graphen jeder Knoten über mindestens einen Kantenzug mit jedem anderen Knoten verbunden sein muß. Weitere Graphstrukturen sind z.B Bäume, wie sie für Jackson-Diagramme verwendet werden oder gerichtete, azyklische Graphen, wie sie zur Darstellung von Typ bzw. Vererbungshierarchien in objekt-orientierten und erweiterten Entity-Relationship Modelliertechniken verwendet werden.

- Redundanzfreiheit
  In den meisten Modelliertechniken wird während der Modellierung jedem einzelnen Modellkonstrukt ein Bezeichner (Prozeß "Buch kaufen", Objekt "Buch", etc.) zugeordnet, an Hand dessen es identifiziert werden kann. Damit die Identifikation eindeutig ist, darf ein Bezeichner (zumindest für eine Modellkonstruktart) nur einmal verwendet werden. Bei Modellen mit hunderten oder tausenden von Modellkonstrukten ist eine maschinelle Überprüfung dieser Eindeutigkeit von großer Bedeutung. Werden im Rahmen dieser Überprüfung mehrfach verwendete Bezeichner entdeckt, so ist dies häufig auch ein Hinweis dafür, daß ein Aspekt mehrfach modelliert wurde. Solche Redundanzen können und müssen aus einem Modell entfernt werden.

[3] ein isolierter Knoten ist mit keinem anderen Knoten durch eine Kante verbunden

- Vollständigkeit und Konsistenz
  Die semantische Vollständigkeit eines Modells, d.h. ob ein Modell den betrachteten Weltausschnitt vollständig beschreibt, kann letztendlich nur vom Modellierer und nicht maschienll geprüft werden. In vielen Techniken können aber Bedingungen formuliert werden, die dem Modellierer auf mögliche Unvollständigkeiten hinweisen; in einem ER-Modell z.B. sollte jeder Entitytyp mindestens mit einem Attribut verbunden sein, da zu jedem modellierte Objekttyp auch seine Struktur beschrieben werden muß. In einem Daten-Fluß-Diagramm sollte die Schnittstelle des modellierten Systems zu seiner Umwelt beschrieben werden. Dazu muß es mindestens einen Endknoten, eine Datenflußkante zu einem Endknoten sowie eine von einem Endknoten ausgehende Kante geben. Die Endknoten repräsentieren die Schnittstellen, die auslaufenden Kanten beschreiben Daten, die vom System verarbeitet werden und die einlaufenden Kanten beschreiben Ergebnisse, die das System für seine Umwelt erzeugt hat.

  [CPR91] stellen ein Werkzeug vor, das versucht, den Modellierer bei der Entdeckung und Beseitigung von semantischen Unvollständigkeiten und Inkonsistenzen zu unterstützen. In diesem Werkzeug werden zuerst natürlichsprachliche Modellspezifikationen in ein vorläufiges semantisches Netz [Rei91] umsetzt. Die Knoten- und Kantentypen des Netzes sowie dessen Aufbau sind fest vorgegeben. Mit ihnen können ähnlich viele semantische Aspekte repräsentiert werden wie mit einer kombinierten Modelliertechnik (siehe Abschnitt 2.2.2). Zusätzlich verfügt das Werkzeug über eine Faktenbasis, in der ähnliche Bedingungen, wie die zuvor beschiebenen abgespeichert sind. Durch die Überprüfung des vorläufigen semantischen Netzes bezüglich dieser Bedingungen wird eine interaktive Sitzung gesteuert, die den Modellierer leitet, das Netz zu vervollständigen oder beim Umsetzen entstandene Inkonsistenzen zu korrigieren.

  Etwas andere Arten von semantischen Inkonsistenzen können in einem Modell entdeckt werden, wenn ein System mit einem Petri-Netz beschrieben wird. Wie bereits erwähnt (siehe Abschnitt 2.1.4) kann das dynamische Verhalten eines Systems in einem Petri-Netz simuliert werden. Dabei können Prozesse entdeckt werden, die sich gegenseitig verklemmen können oder solche die nie zur Ausführung kommen. Diese Überprüfungen können sehr gut durch Werkzeuge vorgenommen werden [LE89].

3.1.3 Wiederverwendung

Die in einem Unternehmen entwickelten Applikationen hängen inhaltlich oft eng zusammen. In einem Modell einer neue Applikation sind daher häufig Aspekte des Unternehmens oder anderer Applikationen beschrieben, die bereits bei der Entwicklung früherer Applikationen modelliert wurden. Durch die Wiederverwendung von Teilen existierender Modell kann nicht nur viel Modellieraufwand gespart werden. Ein weiterer Vorteil liegt darin, daß die wiederverwendeten Modellteile einen Weltausschnitt beschreiben, auf

dem die existierenden Applikationen basieren. Damit kann die Integration der neuen Applikation in das existierende Umfeld schon in der Analysephase berücksichtigt werden.

Technisch ist das Wiederverwenden von Modellteilen, die mit demselben Werkzeug erstellt wurden, im allgemeinen kein Problem. Problematisch ist es allerdings, in einer größeren Menge von existierenden Modellen die für die neue Applikation relevanten zu finden. Mit RECAST [FP90, FGP91] wird z.B. ein Applikationen-Entwurf-System vorgestellt, bei dem besonderer Augenmerk auf die Wiederverwendbarkeit von Entwurfsergebnissen gelegt wird. Alle Modelldaten werden bei RECAST in einer Software-Information-Base (SIB) gespeichert. Zusätzlich baut ein SIB-Administrator ein semantisches Netz [Rei91] auf, in das er sowohl die Modelle selbst, als auch Metadaten, wie den zugehörigen Anwendungsbereich, die verwendete Modelliertechnik und das Modellierwerkzeug, einbindet. Ein Applikationsentwickler kann nun die relevanten Modellteilen zusammensammeln, indem er durch das semantische Netz navigiert und sich das mit jedem Knoten des Netzes verbundene Modell ansieht. Ein Zusatzwerkzeug [FP90] unterstützt ihn dabei, indem es Wege durch das Netz vorschlägt oder Fragen generiert, deren Beantwortung die Navigation durch das Netz beeinflußen.

3.2 Modelliersysteme

Wie bereits in Abschnitt 2.3 erwähnt, ist es für viele Anwendungen nicht ausreichend, ein Modell nur mit einer einzigen Technik zu erstellen. Als Alternative wurde das Prinzip der inkrementellen Modellierung (Abschnitt 2.3.1) vorgestellt, bei dem mehrere Techniken nacheinander angewendet werden. Für die Modellierung in den verschiedenen Techniken müssen dementsprechend auch mehrere Werkzeuge benutzt werden. Die Konsistenz des Gesamtmodells wird dadurch gesichert, daß in einer nachfolgenden Technik jeweils auf Aspekte der bereits existierenden Modelle zurückgegriffen wird und diese Aspekte nicht nochmals modelliert werden. Dazu müssen Modelldaten von den bereits erstellten Modellen in das jeweils nachfolgende Modell übernommen werden. Dieser Vorgang ist vor allem bei größeren Modellen sehr aufwendig und es ist nur schwer zu gewährleisten, daß alle Daten übertragen werden. Da sich dieser Vorgang jedoch sehr gut automatisieren läßt wurden computer-gestützte Modelliersysteme entwickelt, die dem Modellierer diese Arbeit abnehmen und dabei auch noch die Vollständigkeit der Datenübertragung garantieren können.

Die Übernahme von Daten eines Werkzeugs in ein anderes Werkzeug ist ein sehr allgemeines Problem, das nicht nur bei der inkrementellen Modellierung oder bei den objektorientierten Ansätzen auftaucht. Dasselbe Problem stellt sich auch, wenn zu einem, in der Analysephase einer Applikationsentwicklung erstellten, konzeptuellen Modell in der Entwurfphase eine Systemarchitektur entwickelt werden muß.

Analog zur Analyse gibt es für den Entwurf von Systemen Entwurfstechniken und Werkzeuge, die diese Techniken unterstützen. Ausgangspunkt einer Entwurfstechnik ist ein konzeptuelles Modell, das entsprechend weiterverarbeitet wird. So wird z.B in der Ent-

wurfstechnik "Strukturierter Design" [YC79] ein mit Hilfe der "Strukturierten Analyse" entwickeltes Modell verwendet, um eine Modulstruktur und die Schnittstellen zwischen den Modulen zu entwickeln. In der Methode "Jackson Strukturierte Programmierung" [Kil88, Sch90b] wird aus einem Jackson-Diagramm (siehe Abschnitt 2.1.3) die Struktur des Programmes abgeleitet. In [Kat90] wird ein Werkzeug vorgestellt, das den Applikationentwicklern die Entwurfsentscheidung erleichtet, ob es sich bei den im System verwendeten Daten um globale oder lokale Daten handelt. Dazu wird sowohl das für das System entwickelte ER-Modell als auch das Daten-Fluß-Modell metrisch ausgewertet.

Für jedes in der Entwurfsphase verwendete Werkzeug stellt sich wiederum das Problem, Daten von Modellierwerkzeugen zu übernehmen. Die im folgenden beschriebenen Vorschläge zur Lösung dieses Problems sind also unabhängig davon, ob Daten zwischen verschieden Modellierwerkzeugen ausgetauscht werden sollen (horizontale Integration) oder zwischen Modellierwerkzeugen und Entwurfswerkzeugen (vertikale Integration). Da der Hauptaugenmerk dieser Arbeit bei der konzeptuellen Modellierung liegt, wird im folgenden ('ohne Beschränkung der Allgemeinheit') davon ausgegangen, daß mit einem Modelliersystemen die inkrementelle Modellierung unterstützt werden soll.

3.2.1 Aufbau von Modelliersystemen

Unter einem Modelliersystem wird in dieser Arbeit ein computer-gestütztes Softwaresystem verstanden, das besteht aus :

- einer Anzahl von computer-gestützten Modellierwerkzeugen (CASE-Tools),
- einem Repository und
- einer Integrationsfunktionalität.

Mit jedem Modellierwerkzeug können Modelle genau einer bestimmten Technik erstellt werden. Alle CASE-Tools speichern die mit ihnen erzeugten Modelldaten in einer gemeinsamen Datenbank. Diese Datenbank wird im folgenden Repository [Sag89, Win89] genannt. Der Name "Repository" (als Synonyme tauchen auch folgende englische Begriffe auf: encyclopedia, catalog, dictionary, directory) weist lediglich darauf hin, daß es sich bei den in diesen Systemen gespeicherten Daten um Modelldaten handelt. Mit Hilfe der Integrationsfunktionalität können Daten zwischen den zum Modelliersystem gehörenden Werkzeugen ausgetauscht werden. Bezogen auf die Modellierung bedeutet dies, das mit dieser Funktion die mit den verschiedenen Werkzeugen erstellten Teilmodelle zu einem Gesamtmodell integriert werden können. Die Art der Integrationsfunktionalität hängt davon ab, ob die Modelldaten im Repository redundanzfrei gespeichert werden oder nicht.

Bei der Beschreibung von Modelliersystemen muß prinzipiell zwischen "offenen" und "geschlossenen Modelliersystemen" unterschieden werden. Bei geschlossenen Systemen ist die Art, die Zahl und der Hersteller der Modellierwerkzeuge fest vorgegeben, wogegen offene Systeme um zusätzliche Werkzeuge erweiterbar sind. Welchen Einfluß dieser Architekurunterschied auf das Repository und die Integrationsfunktionalität hat, wird im Folgenden näher beschrieben.

3.2.2 Geschlossene Modelliersysteme

Für geschlossene Systeme kann sowohl die redundanzfreie wie auch redundant Speichertechnik verwendet werden. Bei der redundanten Speicherung (Abb. 10) wird für jedes Modellierwerkzeug ein eigenes Schema im Repository angelegt. Die mit einem Werkzeug erzeugten Daten werden ausschließlich in dem zugehörigen Schema gespeichert.

Als Integrationsfunktionalität werden bei dieser Art der Schemadefinition Funktionen benötigt, die beim Wechsel der Modelltechnik bzw. des Modellwerkzeuges die Übernahme der Daten vornehmen. Diese Funktionen suchen die bereits existierenden und für diese Technik relevanten Daten in den verschiedenen Schemata und kopieren sie in das zum aktuell gewählten Werkzeug gehörige Schema. Je größer die Überlappung der mit den verschiedenen Techniken jeweils modellierbaren Semantik ist, desto größer ist bei dieser Speichertechnik die Redundanz der Modelldaten im Repository.

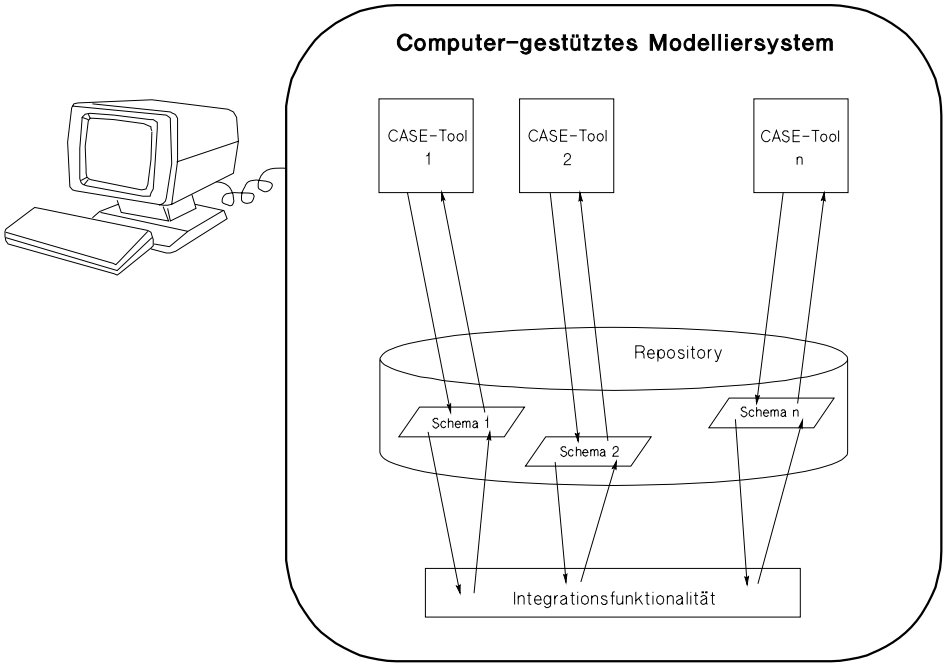

Abbildung 10: Getrennte Schemata

Für eine redundanzfreie Speicherung (Abb. 11) wird lediglich ein einziges Schema definiert, in dem alle Werkzeuge ihre Daten speichern. Als Integrationsfunktionalität muß in diesem Fall jedes Werkzeug um eine zusätzliche Funktion erweitert werden. Diese Funktion extrahiert aus dem globalen Schema alle für die jeweilige Modelliertechnik relevanten Daten. Dabei können auch Daten gelesen werden, die von einem anderen Werkzeug im Repository gespeichert wurden. Die Modelldaten werden also bei einem Wechsel der Modelliertechnik nicht von einem Werkzeug zum anderen kopiert, sondern lediglich von der Integrationfunktionalität des anderen CASE-Tools gelesen.

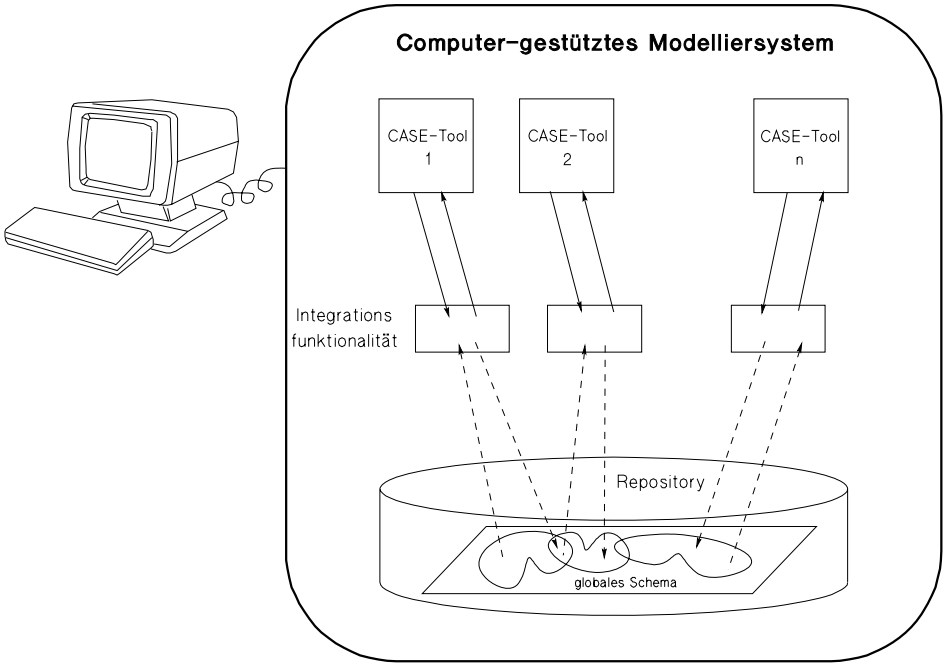

Abbildung 11: Globales Schema

Bei den heute kommerziell verfügbaren Modelliersystemen [Bal91] (häufig auch wegen der Integration mehrere Werkzeuge I-CASE-System genannt) handelt es sich meist um geschlossene Systeme. Sie stellen keine Schnittstellen zur Verfügung, die einen Datenaustausch zwischen dem Modelliersystem und anderen Programmen, wie zum Beispiel Modellierwerkzeuge anderer Hersteller, ermöglichen. Auf Grund der fehlenden Erweiterbarkeit kann der Modellierer nur Modelle mit den im Modelliersystem zur Verfügung stehenden Modellierwerkzeuge erstellen. Erfordert die geplante Verwendung des Gesamtmodells die Modellierung von Aspekten, die mit den systemeigenen Werkzeugen nicht modellierbar sind, so kann das Modelliersystem nicht verwendet werden.

3.2.3 Offene Modelliersysteme

Im Gegensatz zu den nicht erweiterbaren, geschlossenen Systemen verwenden die offenen Modelliersysteme Repositories mit standardisierten Schnittstellen [ISO90a, Lef91, HL93]. Für die Modellierung sind alle CASE-Tools verwendbar, die über eine Schnittstelle zum verwendeten Repository verfügen. Das Modelliersystem ist daher jederzeit um neue Werkzeuge erweiterbar. Damit die Werkzeuge ihre Modelldaten in diesem Repository speichern können, müssen zuvor entsprechende Speicherschemata angelegt werden. Hierfür gibt es, wie bei den geschlossenen Systemen, mehrere Möglichkeiten, deren Auswahl großen Einfluß auf Art und Mächtigkeit der Integrationsfunktionalität hat.

Repository-Schemata

1. Werkzeug-spezifische Schemata

   In diesem Fall wird für jedes Modellierwerkzeug ein eigenes Schema im Repository angelegt und die Daten der einzelnen Werkzeuge in separaten Schemata gespeichert. Zur Integration von Teilmodellen, die mit verschiedenen Werkzeugen bzw. mit den von ihnen unterstützten Techniken erstellt wurden, ist es allerdings notwendig, daß ein Austausch von Daten zwischen den Werkzeugen stattfinden kann. Durch die Verwendung eines gemeinsamen Repositories für alle Werkzeuge und der standardisierten Schnittstelle ist dies technisch kein Problem. Ein Problem ist allerdings die Anzahl der Programme, die den Transfer von Daten eines Schemas in ein anderes durchführen sollen. Bei der Ergänzung des Systems um ein weiteres Werkzeug (d.h. die Erweiterung des Repositories um ein weiteres Schema) muß für jedes bereits existierende Werkzeug (sofern semantische Überlappungen exisiteren) ein Transferprogramm entwickelt werden, das den Datenaustausch zwischen den alten und dem neuen Werkzeug durchführt.

   Zusätzlich stellt sich hierbei die Frage, wer diese Transferprogramme realisiert. Bei den geschlossenen Systemen werden alle Werkzeuge von einem Hersteller entwickelt. Dieser Hersteller kennt sowohl die Syntax (und damit die Definition der Speicherschemata) als auch die Semantik der Techniken. Daher ist es ihm möglich, die überlappenden semantischen Aspekte von zwei oder mehreren Techniken zu analysieren und die entsprechenden Datentransferprogramme zu realisieren. In einem offenen System werden dagegen Werkzeuge von unterschiedlichen Herstellern verwendet. Da kein Hersteller wissen kann, welche Werkzeuge zu einem offenen Modelliersystem gehören, muß die Integrationsfunktionalität für jedes Modelliersystem individuell realisiert werden.

2. Modelliertechnik-spezifische Schemata

   Eine andere Art der Schemadefinition geht von der Erkenntnis aus, daß viele der existierenden CASE-Tools dieselben Techniken unterstützen. Die Unterschiede zwischen den Werkzeugen liegen in der Handhabung, in den Weiterverarbeitungsmöglichkeiten oder in der Präsentation der Modellkonstrukte. Daher wurde mit CDIF[4] [Imb91] versucht, für einige Basistechniken (ER, DFD, ...) je ein Schema zu entwerfen und zu standardisieren, das alle zu dieser Grundtechnik bekannten, syntaktischen Konstrukte beinhaltet. Die Modellierwerkzeuge werden derart geändert, daß sie ihre Daten nicht mehr in einem werkzeug-spezifischen Schema ablegen, sondern in den für alle Werkzeuge zugänglichen technik-spezifischen. Hierdurch erhält man eine gelungene Entkopplung der syntaktischen Modellierdaten von den Modellierwerkzeugen, mit denen die Daten erzeugt wurden. Der Modellierer erhält damit völlige Flexibilität bei der Wahl des Werkzeugs, das ihn bei der Modellierung (mit einer bestimmten Technik) am besten unterstützt.

[4] CASE-Data-Interchange-Format

Für die Werkzeughersteller sind diese Schemata standardisierte Schnittstellen, die eine direkte Übernahme von Daten aus anderen Werkzeugen erlauben. Voraussetzung dafür ist, daß die Werkzeuge dieselbe Technik unterstützen. Der Integration von Teilmodellen im Sinne des methodischen Ansatzes, d.h. die Übernahme von Modelldaten einer Technik in eine andere Technik, wird aber auch mit dieser Art der Schemadefinition nicht vereinfacht.

3. Globales Schema

   Ein weiterer Lösungsansatz ist die Entwicklung eines einzigen, globalen Schemas, indem die Daten aller (zum Modelliersystem gehörenden) Modelliertechniken bzw. -werkzeuge gespeichert werden. Hierzu reicht es nicht, die Vereinigung über z.B. alle CDIF-Schemata zu bilden, da in diesen lediglich die Syntax der Modelle gespeichert wird. Da jede Modelliertechnik ihre eigene Syntax hat, ist eine Integration von Teilmodellen auf Basis syntaktischer Konstrukte nicht möglich. Stattdessen müßte ein Schema entworfen und standardisiert werden, in das sich die Semantik alle existierender und zukünftiger Techniken abbilden ließe. Die Integrationsfunktionalität wäre dann dieselbe, wie in den geschlossenen Systemen. Zu jedem Modellierwerkzeug müßte eine eigene Integationsfunktionalität existieren, die die Semantik des, in der Modelliertechnik des Werkzeugs beschriebenen, Modells im globalen Schema speichert bzw. die für die Modelliertechnik relevanten Daten extrahiert. Die Entwicklung eines solchen Schemas scheint allerdings auf Grund der Vielfältigkeit der semantischen Aspekte nicht realisierbar zu sein.

3.2.4 Resümee

Diese Ausführungen haben gezeigt, daß es in offenen Modelliersystemen unabhängig von der gewählten Schemadefinition des Repositories schwierig ist, eine adäquate Integrationsfunktionalität für ein offenes Modelliersystem zu realisieren. Demgegenüber stehen die wichtigen Aspekte der Erweiterbarkeit und der Flexibilität.

Um also die Vorteile der geschlossenen und der offenen Systeme zu vereinen, müßte ein Art von Modelliersystem entwickelt werden, das zumindest folgende Anforderungen erfüllt :

1. Es muß das Modellieren mit unterschiedlichsten Basistechniken und kombinierten Techniken unterstützen.

2. Es muß den Modellierern große Flexibilität bei der Wahl der Techniken und der Reihenfolge ihrer Anwendung geben.

3. Es muß leicht um weitere Techniken bzw. Werkzeuge erweiterbar sein.

4. Es muß eine Integration von Teilmodellen durch die flexible Übernahme von Daten einer Technik in eine andere vornehmen können.

Mit diesen Anforderungen wurde selbstverständlich noch kein vollständiger Anforderungskatalog [DK88, DHS+91] für ein Modelliersystem bzw. eine Software-Entwicklungs-Umgebung (SEU) definiert. In der Diskussion über die Anforderungen an eine SEU zur Integration verschiedener CASE-Tools [BGMT89] ist die in dieser Arbeit diskutierte Datenintegration allerdings ein besonders wichtiger Aspekt. Weitere Aspekte sind [Was89, Kel91a] die Integration von eventuell unterschiedlichen Betriebssystemen oder Benutzerschnittstellen sowie die Steuerung, Automation und Überwachung des Software-Entwicklungsprozesses. Da die Art der Lösung dieser Aspekte keinen Einfluß auf die Problematik der Datenintegration hat [Kel91b], werden sie auch im Rest dieser Arbeit nicht weiter berücksichtigt.

Im folgenden Abschnitt werden der Aufbau und vor allem die Funktionsweise einer neuen Art von Modelliersystemen vorgestellt, die alle vier vorweg gestellten Anforderungen erfüllen. Diese Systeme werden im weiteren C-CASE-Systeme (Configurable-CASE) genannt, da sie auf einer konfigurierbaren Systemarchitektur basieren. Der große Unterschied zu den bisher beschriebenen, existierenden Modelliersystemen liegt in der Tatsache, daß sich ein C-CASE-System nicht aus mehreren, verschiedenen Modellierwerkzeugen (plus einem Repository und einer Integrationsfunktionalität)[5] zusammensetzt, sondern lediglich einen einzigen generischen Modelliereditor beinhaltet. Dieses generische Werkzeug kann derart konfiguriert werden, daß es die Modellierung in den verschiedensten Modelliertechniken sowie den Austausch von Modelldaten zwischen den Techniken ermöglicht.

Zur Realisierung eines C-CASE-Systems mit der im folgenden Abschnitt beschriebenen Funktionalität und den damit verbundenen Vorteilen gegenüber existierenden Systemen müssen noch einige grundlegende Probleme gelöst werden. Daher folgt im Abschnitt 5 eine ausführliche Diskussion der zu lösenden Probleme sowie einiger bereits existierender Lösungsansätze.

4 Konfigurierbare Modelliersysteme

4.1 Konfigurieren eines C-CASE-Systems

C-CASE-Systeme sind computer-gestützte Modelliersysteme, die individuell auf die Anforderungen der Modellierer sowie ihrer zu modellierenden Applikationen und Weltausschnitte angepaßt werden. Dabei wird davon ausgegangen, daß der Einsatz eines Modelliersystems geplant wird. Aus der Analyse der geplanten Verwendungszwecke ergeben sich Anforderungen an die zu erstellenden Gesamtmodelle. Es kann somit festgelegt werden, welche semantischen Aspekte mit dem System modellierbar sein müssen. Als nächster Schritt in der Planungsphase muß eine Menge von Modelliertechniken derart zusammengestellt werden, daß es für jeden Aspekt mindestens eine Technik gibt, mit dem dieser Aspekt modelliert werden kann. Zusätzlich können natürlich auch noch weitere Techniken

[5] siehe Definition in Abschnitt 3.2.1

gewählt werden, die für die Nutzer des Modelliersystems von Interesse sind. Anschließend an diese Vorarbeiten wird das eigentliche C-CASE-System realisiert.

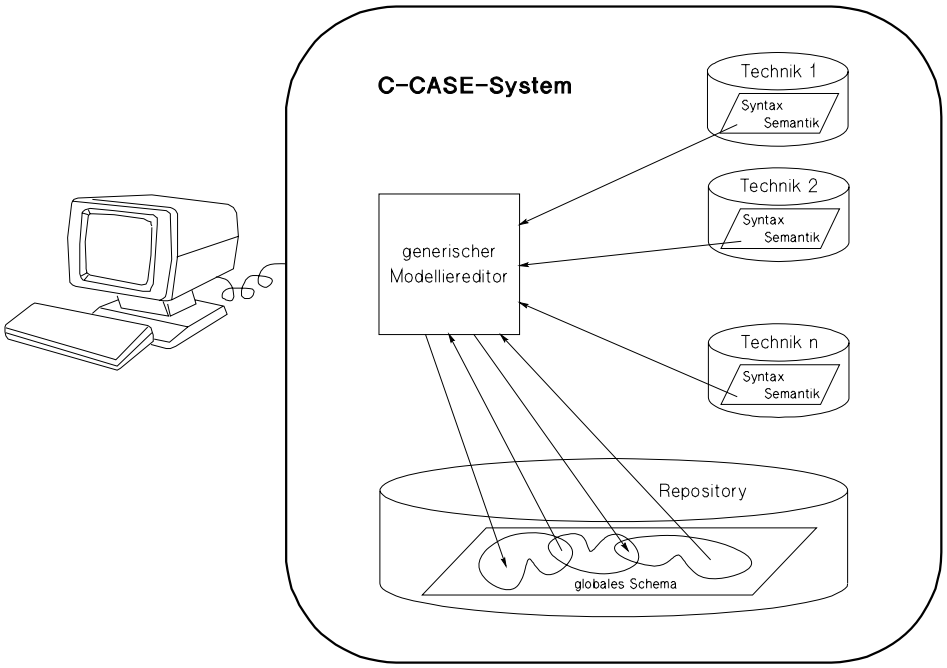

Abbildung 12: C-CASE-System

Dazu wird ein Grundsystem (Abb. 12) bestehend aus einem semantischen Repository und einem generischen Modelliereditor nach den in der Planung erarbeiteten Anforderungen konfiguriert. Dabei wird für das Repository ein globales Schema entworfen und angelegt, in das sich alle geforderten semantischen Aspekte abbilden lassen und für jede, in der Planungsphase gewählte, Modelliertechnik eine Notation, eine Syntax, eine Semantik und die Abbildung der Semantik in das Repositoryschema definiert. Für die Beschreibung dieser Definitionen wird ein Format verwendet, das vom Modelliereditor interpretiert werden kann. Der Modelliereditor (Abb. 12) ist also ein generisches Werkzeug, daß die Modellierung mit unterschiedlichen Techniken unterstützen kann.

Die Abbildung eines Modells in das Repository muß eindeutig umkehrbar sein, damit ein Modell auch nach einer zwischenzeitlichen Speicherung wieder in den Modelliereditor eingelesen und in der Notation bzw. der Syntax dieser Technik weiterbearbeitet werden kann. Wesentlich schwieriger ist das Einlesen von Modelldaten in einer Technik, wenn die Modelldaten selbst nicht in dieser Technik erfaßt und gespeichert wurden. Für diesen Fall muß zu jeder Modelliertechnik eine Integrationsfunktionalität definiert werden. Sie beschreibt die Rückabbildung aller im Repository gespeicherter Modelldaten in die Darstellung der jeweiligen Modelliertechnik. Welche Auswirkungen und Vorteile diese Art der Integrationsfunktionalität hat, wird in den nächsten Abschnitten ausführlicher erklärt.

Zur Konfiguration eines C-CASE-Systems muß also das Schema des semantischen Repositories entworfen und angelegt sowie eine Menge von Modelliertechniken einschließlich ihrer Integrationsfunktionalität definiert werden. Danach ist das C-CASE-System fertig konfiguriert und einsatzbereit.

4.2 Modellieren mit einem C-CASE-System

Der Modellierer startet den Modelliereditor (ME) und wählt eine beliebige Technik, für die der ME konfiguriert wurde. Der ME interpretiert die Notationsdefinition dieser Technik und ermöglicht dem Modellierer die Eingabe eines Modells in dieser Notation. Daraufhin prüft er die syntaktische Richtigkeit des Modells, analysiert die Semantik und speichert diese im globalen Schema des Repositories. Nach Fertigstellung des ersten Modells wechselt der Modellierer innerhalb des ME die Technik. Aus der Semantikdefinition dieser Technik ermittelt der ME die Menge der damit modellierbaren Aspekte. Die im ME eingebaute Integrationsfunktionalität prüft, ob es bereits Daten im Repository gibt, die einen oder mehrere dieser Aspekte beschreiben. Ist dies der Fall, so werden diese Aspekte in der Notation und Syntax der neu gewählten Technik dargestellt. Der Modellierer beginnt also mit der Erstellung des zweiten Teilmodells nicht auf dem 'leeren' Bildschirm. Damit wird eine Mehrfachmodellierung derselben Aspekte und der damit verbundene unnötige Aufwand vermieden. Auch die Gefahr, daß sich zwei Modellierungen derselben Aspekte widersprechen, ist damit beseitigt.

Bei dieser Art der inkrementellen Modellierung kann eigentlich nicht von einer Erstellung von Teilmodellen und deren Integration gesprochen werden. Vielmehr werden durch die Anwendung weiterer Techniken gewisse Aspekte zum bereits existierenden Gesamtmodell hinzugefügt. Eine Reihenfolge, in der die Modelliertechniken angewendet werden müssen, ist bei der Verwendung eines C-CASE-Systems nicht vorgeschrieben. Es ist dem Modellierer völlig freigestellt, welche Reihenfolge, d.h. welche Methode er zur Erstellung seines Modells anwendet. Die inkrementelle Modellierung impliziert allerdings immer, daß die Reihenfolge der Anwendung der Modelliertechniken derart gewählt wird, daß eine nachfolgende Technik in gewissen Aspekten mit den bereits angewendeten Techniken überlappt. Ist dies nicht der Fall, so werden zwei semantisch völlig unabhängige Teilmodelle erstellt. Der Modellierer beginnt in diesem Fall mit der Erstellung des zweiten Teilmodells auf dem 'leeren' Bildschirm.

4.3 Erweiterbarkeit

Ein C-CASE-System ist auch nach einer abgeschlossenen Konfiguration noch flexibel um weitere Modelliertechniken erweiterbar. Häufig werden mit den neuen Techniken keine neuen oder zusätzlichen Aspekte modelliert, sondern lediglich eine andere Kombination von Aspekten. Ein Beispiel hierfür sind die in Abschnitt 2.3.2 bereits vorgestellten objekt-orientierten Ansätze. In ihnen werden verschiedene Basistechniken nacheinander

angewendet, jeweils aus dem objekt-orientierten Blickwinkel. Die Menge der semantischen Aspekte, die mit objekt-orientierten Methoden modelliert werden können, ist meist nicht größer als z.B bei einigen kombinierten Techniken. Existiert also bereits ein C-CASE-System, das für die Modellierung mit einigen ausgewählten Einzeltechniken oder kombinierten Techniken konfiguriert wurde, so kann dieses System um ein Werkzeug zur Unterstützung einer objekt-orientierten Methode erweitert werden, ohne die bereits im Einsatz befindliche Konfiguration des existierenden C-CASE-Systems zu ändern. Dazu muß lediglich die Syntax und die Semantik der neu zu verwendenten Techniken in derselben Art, wie die anderen Modelliertechniken, definiert werden. Der generische Modelliereditor interpretiert die neue Syntaxdefinition, stellt dem Modellierer Funktionen zur Erstellung von Modellen in der neuen Technik zur Verfügung und kann mit Hilfe der Semantikdefinition die so erfaßten Modelldaten im semantischen Repository ablegen. Da davon ausgegangen wurde, daß mit der neuen Modelliertechnik keine neuen (d.h. noch nicht im semantischen Repository repräsentierbaren) semantischen Aspekte modelliert werden, muß das globale Schema des Repositories nicht geändert werden. Der Modelliereditor kann daher auch sofort auf bereits gespeicherte Modelldaten zugreifen und diese in der neuen Technik darstellen.

Soll von einem bereits konfigurierten C-CASE-System eine zusätzliche Modelliertechnik unterstützt werden, mit bisher nicht berücksichtigten semantischen Aspekten, so muß zuerst das Schema des Repositories entsprechend erweitert werden. Auf die Konfiguration der bereits verwendeten Modelliertechniken hat eine reine Erweiterung des Schemas keinen Einfluß.

Es kann aber auch vorkommen, daß im Rahmen einer Erweiterung gewisse Teile des bestehenden Schemas geändert werden müssen, da z.B ein semantischer Aspekt verfeinert, d.h. differentierter betrachtet wird. In diesen Fällen muß nicht nur das Schema geändert werden, sondern auch alle Semantikdefinitionen von Modelliertechniken, mit denen die geänderten Aspekte modellierbar sind. Nicht zu vergessen sind dabei die zu diesen Aspekte gespeicherten Daten.

4.4 Weitere Vorteile

Vorteil : Modellsichten
Die Integrationsfunktionalität des Modelliersystems ermöglicht es, daß ein Modell in der Syntax bzw. Notation verschiedener Techniken angezeigt werden kann, wie dies z.B. in [Bro92] gefordert wird. Bei den Darstellungen eines Modells in den einzelnen Techniken handelt es sich jeweils um vordefinierte, besonders aussagekräftige Sichten des Gesamtmodells. Die Aussagekraft der Sichten beruht auf der Tatsache, daß bei der Entwicklung der Modelliertechniken immer versucht wurde, die Erstellung möglichst aussagekräftiger Teilmodelle zu erreichen. Dabei ist es sogar möglich, die Darstellung z.B. einer kombinierten Technik zu wählen, deren Aspekte nicht in dieser Technik selbst, sondern in mehreren anderen Techniken schrittweise erstellt wurden. Solche Darstellungen sind besonders wichtig, um Inkonsistenzen zwischen Aspekten aufzudecken, wenn die Aspekte

mit unterschiedlichen Techniken modelliert wurden. Dem Argument, auf diese Weise Inkonsistenzen zu entdecken, könnte entgegnet werden, daß es besser gewesen wäre, diese Aspekte gleich in der kombinierten Technik gemeinsam zu erstellen. Dies ist nicht immer richtig, da es sein kann, daß ein Modellierer ein Modell prüft, das von einem oder mehreren Kollegen erstellt wurde, die diese umfassendere Technik nicht beherrschen.

Vorteil : Umweltschnittstellen
Ein weiterer Vorteil eines C-CASE-Systems wird deutlich, wenn nach Abschluß eines Modellierprojektes ein weiteres Modell erstellt werden soll. Gibt es nämlich Überlappungen zwischen dem bereits modellierten Weltausschnitt und dem jetzt zu modellierenden, so können die Modelldaten des existierenden Modells, die den überlappenden Ausschnitt beschreiben, ermittelt und als Basis für die Erstellung des neuen Modells wiederverwendet werden. Werden die Modelle als Analysemodelle für die Software-Entwicklung verwendet, so stellen diese Modelldaten gerade die Schnittstelle zwischen dem bereits existierenden Programm und dem geplanten neuen Programm dar. Die Integration von Programmen kann dadurch bereits in der Analysephase berücksichtigt werden.

Vorteil : Gesamtmodell
Die Modelldaten des neuen Modells sollten im selben Repositoryschema gespeichert werden, in dem sich bereits die des ersten Modells befinden. Auf diese Weise kann im Laufe der Zeit schrittweise ein Gesamtmodell z.B. eines Unternehmens aufgebaut werden. Dies geschieht ohne jeglichen Zusatzaufwand, da im Rahmen der Software-Erstellung sowieso immer wieder neue Modelle erstellt werden müssen. Ein in dieser Form vorliegendes Unternehmensmodell kann als allgemeines Unternehmens-Informations-System für Ablauf-, Organisations- und sonstige Analysen genutzt werden [Bro92]. Dazu muß allerdings das in Abschnitt 5.6 beschriebene Problem der Speichergraphsichten gelöst werden.

5 Zu lösende Probleme und existierende Ansätze

In einem C-CASE-System der vorgestellten Art wird als Basis für die Integration von Modelliertechniken ein standardisiertes Repository verwendet, in dem alle Modelldaten gespeichert werden können. Für dieses Repository muß ein Speicherschema entwickelt werden, das alle mit diesen Techniken modellierbaren Aspekte umfaßt. Welche semantischen Aspekte mit einer Modelliertechnik repräsentiert werden sollen, muß für jede Technik genau analysiert und festgelegt werden. Da der Modellierer dieses Schema aber nicht direkt füllt, sondern weiterhin seine Modelle in der Notation der verschiedenen Techniken eingeben können soll, muß für jede Technik die Abbildung der Modelle auf das globale Schema definiert werden. Hierzu müssen Formalismen entwickelt werden, mit denen sich die Syntax, die Semantik und diese Abbildungen präzise beschreiben und festlegen lassen. Um einmal im Repository gespeicherte Modelle weiterbearbeiten zu können, muß die Abbildung von Modelldaten auf das Speicherschema umkehrbar sein. Ein weiteres Problem, das es zu lösen gilt, ist das Visualisieren der Modelle nach der Extraktion der Modelldaten aus dem Repository.

Im folgenden werden die angesprochenen Probleme näher betrachtet und mit bereits existierenden Ansätzen verglichen bzw. gegen diese Ansätze abgegrenzt.

5.1 Repositories

Mit der Entwicklung und Standardisierung von IRDS [Gol85, MR86, PSM90] ist ein Rahmen für den Aufbau von Repositories definiert worden. Diese Art von Repositories soll helfen, die großen Mengen unterschiedlicher Daten z.B. in einem Unternehmen, die zum Teil in verschiedenen Systemen gespeichert sind, besser handhaben zu können. Dazu ist es notwendig zusätzlich zu den Daten der realen Welt (Namen, Gehälter, Produktnummern, ...) auch noch Metadaten zu speichern. Diese Metadaten [HL93] beschreiben den Aufbau, die Speicherung, die Bedeutung, die Verwendung von Daten sowie deren Beziehungen zueinander. Zur Speicherung von Metadaten und deren Weiterverarbeitung muß ein Schema definiert und beschrieben werden. Bei den Daten, die das Schema der Metadaten beschreiben, handelt es sich bereits um Metametadaten, die wiederum beschrieben werden müssen. Hieraus entstand die Idee eines selbstbeschreibenden Repositories, das im Falle von IRDS aus vier Schichten besteht (siehe Abb. 13).

Je zwei übereinanderliegende Schichten bilden ein Paar, wobei jeweils die obere Schicht eines Paares eine Beschreibung der unteren Schicht enthält. Die unterste der vier Schichten wird Anwendungsschicht genannt und enthält Daten der realen Welt. Dementsprechend beschreibt das Information-Resource-Dictionary (IRD) als darüberliegende Schicht die Daten der Anwendungsschicht. Im IRD werden die Metadaten gespeichert und wiederum durch Daten in der IRD-Schema Schicht beschrieben werden. Die oberste Schicht (IRD-Schema-Definition) beschreibt den Aufbau der IRD-Schemata. Hierzu wird für jedes Repository ein Datenmodell festgelegt, das nicht mehr innerhalb des Repositories selbst beschieben wird, sondern z.B in dessen Dokumentation oder bei standardkonformen Repositories in der entsprechenden Standardreferenz [ANS88, ISO90b, Lef91]. Die oberste Schicht bildet somit den Fixpunkt in der Kette von Paaren, deren obere Schicht die Daten der darunterliegenden beschreibt.

Es gibt zahlreiche Arbeiten zum Thema "Repositories und deren Anwendung". Die offensichlichsten Unterschiede zwischen diesen Arbeiten lassen sich systematisch durch Antworten auf die folgenden vier Fragen gewinnen:

1. Welches Datenmodell wird in der obersten Schicht (IRD-Schema-Definition[6]) verwendet ?

2. Wie sind die IRD-Schemata aufgebaut bzw. welche Art von Metadaten können im IRD gespeichert werden ?

3. Wie werden die Daten in das IRD eingegeben, d.h. wer tut es und ggf. mit welchen Werkzeugen ?

[6] Die oberste Schicht wird im ISO-Standard "Fundamental Level" genannt

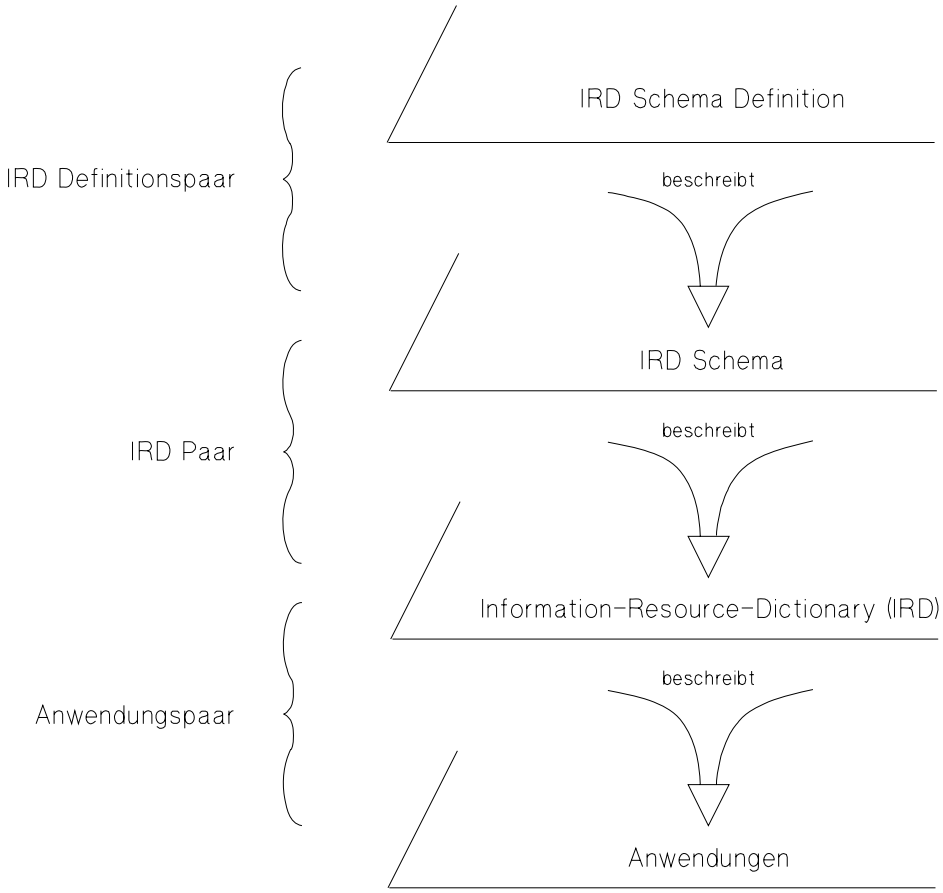

Abbildung 13: IRDS - Vier Schichten Architektur

4. Für welchen Anwendungszweck soll das Repository, insbesondere die Schemata der IRD-Schema Schicht definiert werden ?

ANSI [ANS88] und ISO [ISO90b] haben einen Standard für ein IRDS verabschiedet. In beiden Standards wird für die IRD-Schema-Definition jeweils ein Entity-Relationship-Modell festgelegt. Das von ISO verwendete ERM ist ein wenig mächtiger, da es z.B mehrstellige Beziehungen und attributierte Beziehungen zuläßt [Win88]. Wie in [HL93] erwähnt, ist für die oberste Schicht jedes beliebige Datenmodell verwendbar, sofern es mächtig genug ist, um die, für die Verwendung des Repositories benötigten, IRD-Schemata zu beschreiben. Bei der Erarbeitung des ANSI-Standards wurde versucht, einen für möglichst viele Anwendungsbereiche verwendbaren Rahmen zu erstellen.

Demgegenüber ist der ISO-Standard [ISO90a] explizit darauf ausgelegt, die Verwaltung und Beschreibung von Daten zu unterstützen, die in relationalen Datenbanksystemen (RDBMS) gespeichert sind. Die meisten kommerziellen RDBMS verfügen über ein "Data-Dictionary" (D/D), dessen Daten ebenfalls in relationaler Form gespeichert werden. In diesen D/D werden nicht nur die Schemata der Datenbanken beschrieben, sondern auch die Datenbank-Nutzer, deren Rechte, Statistiken, Integritätsbedingungen und vieles mehr.

Es handelt sich also bei diesen D/D bereits um IRD's im Sinne der IRDS-Standards. Zu einem selbstbeschreibenden Repository fehlt noch eine Beschreibung dieser D/D's bzw. IRD's. Daher wurde im ISO-Standard [ISO90a] ein Schema festgelegt, mit dem D/D von RDBMS beschrieben werden können.

Ein Ziel des ISO-Standard ist, daß die Hersteller von RDBMS eine ISO-konforme Beschreibung ihrer D/D entwickeln und mit ihren Systemen ausliefern. Diese Beschreibung soll dann in das Repository des Kunden eingespielt werden. Damit dies immer und einfach möglich ist, geht der ISO-Standard davon aus, daß das Repository selbst mit Hilfe eines RDBMS realisiert wurde, und alle Datenbanksysteme über eine standardisierte SQL-Schnittstelle [ISO89] verfügen. Ein ISO-konformes Repository stellt für ein Unternehmen ein zentrales System zur Verwaltung aller in RDBMS gespeicherten Daten dar.

Den Ideen des ISO-Standards folgend werden in [DK87] und in [HBRY91] weiterentwickelte Repositories vorgestellt. In beiden Arbeiten werden IRD-Schemata in Form vom relationalen Tabellen definiert. Mit dem Repository in [DK87] wird die Verwaltung von Computeranwendungen unterstützt. Dazu werden Metadaten zu Programmen, Programmteilen und Datenelementen sowie deren Dokumentation und deren Benutzer gespeichert, einschließlich der Beziehungen zwischen den verschiedenen Elementen. Das Repository in [HBRY91] ermöglicht nicht nur statische sondern auch dynamische und konditionale Abhängigkeiten zwischen Metadaten zu beschreiben. Zur Beschreibung der hierfür vorgestellten IRD-Schemata war es allerdings nötig, statt des vom Standard vorgeschlagenen ER-Modells ein mächtigeres Datenmodell (TSER) auf der obersten Schicht des Repositories zu verwenden. In beiden Arbeiten werden SQL-Anfragen vorgestellt, die beispielhaft die Verwendung der Repositories als Informationssystem für Metadaten erläutern. Auf die Art und Weise, wie die IRD's mit Metadaten gefüllt werden, wird in keiner der beiden Arbeiten näher eingegangen.

Bei den bisher vorgestellten Repositories wurde davon ausgegangen, daß zu bereits existierenden Daten Metadaten eingegeben werden und somit die Verwaltung dieser Daten vereinfacht wird. Eine etwas andere Anwendung bei der Metadaten erzeugt und verarbeitet werden, ist die Entwicklung von Applikationen oder allgemeiner die konzeptuelle Modellierung. Auch hierbei werden Dinge der realen Welt beschrieben, unabhängig davon, ob sie schon jetzt oder ggf. erst nach der Realisation der Anwendung existieren. Trotzdem handelt es sich bei Modelldaten von konzeptuellen Modellen um Metadaten, die mit Hilfe von Repositories gespeichert und verarbeitet werden können. Die Anwendungsschicht dieser Repositories bleibt dabei leer und wird in den Entwicklungsphasen einer Anwendung nicht weiter beachtet. Ein weiterer Unterschied ist die Art, wie das IRD mit Metadaten gefüllt wird. Hierfür stehen dem Benutzer Werkzeuge (CASE-Tools) zur Verfügung, mit denen er Modelle erstellt. Diese Werkzeuge setzen die Modelldaten in Metadaten um, die daraufhin im Repository gespeichert werden.

In einem C-CASE-System gibt es nur ein Werkzeug, das Metadaten im Repository speichert bzw. liest - den generischen Modelliereditor. Ein C-CASE-System wird allerdings auch nicht um CASE-Tools erweitert, sondern um Modelliertechnikbeschreibungen, die vom ME interpretiert werden und den ME die Gestalt eines neuen Werkzeugs annehmen

lassen. Daher spielt die Standardisierung des Repositories für die Erweiterbarkeit des Modelliersystems keine Rolle.

Von großem Nutzen ist die Verwendung eines Standardrepositories allerdings nach der Fertigstellung der Applikation. Geht man davon aus, daß sowohl die Modelldaten der Analysephase als auch die der Entwurfsphase im selben Repository gespeichert werden, so handelt es sich bei diesen Modelldaten exakt um jene Metadaten, die auch in den bereits beschriebenen Repositoryansätzen [HBRY91, DK87] zu einer Applikation erfaßt werden. Der große Unterschied liegt in dem Zeitpunkt der Eingabe der Metadaten. Werden zur Entwicklung einer Applikation CASE-Tools verwendet und die Daten in einem offenen, standardisierten Repository gespeichert, so liegen die Metadaten zu dieser Applikation bereits vor Inbetriebnahme der Applikation vor. In den anderen Fällen [HBRY91, DK87] müssen diese Daten nachträglich erfaßt werden. Dies bedeutet nicht nur, daß ein beträchtlicher Zusatzaufwand geleistet werden muß. Die Übereinstimmung der Metadaten mit der Applikation ist meist besser, wenn aus der Beschreibung die Applikation selbst entwickelt wurde, als bei einer Nachdokumentation.

5.2 Schemata für semantische Repositories

Für das Repository eines C-CASE-Systems muß ein globales Schema entworfen werden, in dem alle Modelldaten gespeichert werden können. Um das konzeptuelle Modell unabhängig von der verwendeten Modelliertechnik speichern zu können, muß statt der Syntax des Modells dessen Semantik gespeichert werden. Im folgenden werden einige Arbeiten vorgestellt, die sich mit dieser Problematik beschäftigen. Dabei wird kurz umrissen, mit welcher Intention das jeweilige Schema entwickelt wurde, was in den Schemata gespeichert werden kann und ob Funktionalitäten beschrieben sind, die das jeweilige Schema verwenden.

Entwicklung der Schemata
In [PG89] werden acht der bekanntesten Modelliermethoden daraufhin untersucht, welche semantischen Aspekte mit ihnen, d.h. mit den im Rahmen der jeweiligen Methode verwendeten Modelliertechniken, modellierbar sind. Aus der Vereinigung aller dabei ermittelten Aspekte wurde ein Schema entwickelt, in dem folglich die Modelldaten aller Modelle gespeichert werden können, die mit einer der acht Methoden erstellt werden.

In [JF91] wird ein System vorgestellt, das ebenfalls auf einem semantischen Repository basiert. Die Entwicklung des Schemaaufbaus wird nicht erklärt; stattdessen werden drei Ziele definiert, die das in dieser Arbeit definierte Schema erfüllen soll. Das Schema soll :

1. alle semantischen Aspekte repräsentieren können, die im "Requirements Engineering" und in der "Künstlichen Intelligenz" allgemein als wichtig anerkannt sind.

2. die Übersetzung von Modellen, die mit den allgemein verwendeten Modelliertechniken beschrieben sind, von der Notation dieser Modelltechniken in das Schema und umgekehrt unterstützen.

3. die von den Autoren betriebene Forschung im Bereich der Anforderungsdefinitionen und des Entwurfes von Systemen unterstützen.

Die Verifikation dieser Ziele wird in den vorliegenden Veröffentlichungen [JF91, JFH92] nicht explizit durchgeführt. Sie kann auch implizit nicht durchgeführt werden. Zu 1. müßten die Autoren die Aspekte aufzählen, von denen sie glauben, daß sie allgemein als wichtig anerkannt sind. Zu 2. müßten sie die Modelliertechniken nennen, deren Modelle sich abbilden lassen.

In beiden Arbeiten werden also Schemata vorgestellt, in denen sich Modelldaten verschiedener Modelliertechniken speichern lassen. Im nächsten Abschnitt wird allerdings deutlich, daß es kein Schema gibt und auch nicht geben wird, in dem die Semantik aller (denkbaren) Modelliertechniken gespeichert werden können. Daher wird in der Planungsphase der Konfiguration eines C-CASE-Systems jeweils ein Schema entwickelt, das exakt auf die Anforderungen der geplanten Verwendung abgestimmt ist. Dazu muß eine möglichst minimale und semantisch redundanzfreie Menge von Aspekten gefunden werden. In den beiden im folgenden kurz erwähnten Arbeiten werden einige, von Modelliertechniken unabhängige, Ideen aufgezeigt, welche Aspekte für die konzeptuelle Modellierung interessant sind und wie für diese Aspekte passende Schemata bzw. Metamodelle entwickelt werden können.

Statt der Semantik von Modelliertechniken werden in [CPR91] natürlich-sprachlichen Anforderungsdefinitionen untersucht. Daraus werden die, zur abstrahierten Darstellung der Semantik dieser Texte, wichtigsten semantischen Aspekte und ihre Beziehungen zueinander ermittelt.

Rein argumentativ wird in [VR92] schrittweise ein Metamodell entwickelt, daß, nach Ansicht der Autoren, die wichtigsten semantischen Aspekte berücksichtigt, die zur Modellierung eines Informationssystems modelliert werden sollten. Es umfaßt Aspekte zur Beschreibung statischer Zusammenhänge, der Systemumwelt, der Interaktion zwischen dem modellierten System und seiner Umwelt sowie zur Unterstützung der Modellierung im Rahmen des Software-Entwicklungszykluses.

Mächtigkeit der Schemata

Die in den verschiedenen Arbeiten entwickelten Schemata bzw. Metamodelle unterscheiden sich deutlich in der Anzahl der semantischen Aspekte bzw. der Beziehungen zwischen den Aspekten. Das Schema in [PG89] umfaßt 20 Aspekte und 67 Beziehungen, das in [VR92] 15 Aspekte und 21 Beziehungen, das in [CPR91] 5 Aspekte (entity, domain, action, event, constraint) und 5 Beziehungstypen (part-of, is-a, member-of, triggers, constraints). Diese 5 Aspekte und einige der Beziehungstypen sind auch Bestandteil der größeren Schemata. Zusätzlich können in den größeren aber z.B. noch Daten zum Systemumfeld (environment, organization, ...) und zu Systemresourcen (actor, person, device, datastore, ...) erfaßt werden. Das bedeutet, daß im größeren Schema semantisch mächtigere Modelle gespeichert werden können. Modelle, die mit dem Metamodell aus [VR92] repräsentiert werden können, wären in dem Schema von [CPR91] nicht verlustfrei zu speichern.

Für ein C-CASE-System spielt die 'absolute' Mächtigkeit des Schemas keine Rolle, sondern eher die Minimalität des Schemas bezogen auf die zu unterstützenden Modelliertechniken, damit überlappende Aspekte nicht verschieden im Repository gespeichert werden. Sollen mit einem C-CASE-System nur semantisch kleinere Modelle mit wenigen Modelliertechniken erstellt werden, so genügt auch die Konfiguration eines kleineren, weniger mächtigen Schemas für das semantische Repository.

Verwendung der Schemata
Der Unterschied zwischen den Schemata der semantischen Repositories kann aber auch aus der Unterschiedlichkeit der Modelliersysteme resultieren, die diese Schemata verwenden. In [CPR91] wird ein System vorgestellt, das aus einer natürlichsprachlichen Spezifikation ein vorläufiges, allerdings bereits streng typisiertes, semantisches Netz aufbaut. Als Typen werden die bereits erwähnten 5 Aspekte und 5 Beziehungenstypen verwendet. Eine ebenfalls zu diesem System gehörende Expertensystem-Komponente analysiert das vorläufige Netz und fordert den Benutzer auf, die entdeckten Unvollständigkeiten bzw. Inkonsistenzen zu ändern.

Für das Schema in [VR92] gibt es keine computer-gestützten Funktionen zum Füllen des Schemas oder zur Auswertung gespeicherter Modelldaten. Dem Modellierer wird in dieser Arbeit lediglich eine Reihenfolge von Schritten vorgeschlagen, in der er die verschiedenen Aspekte modellieren soll.

Die Schemata in [PG89] und in [JF91] werden dazu verwendet, Modelldaten abzuspeichern, die in verschiedenen, bekannten Modelliertechniken formuliert werden. Da diese Verwendung der des semantischen Repositories in einem C-CASE-System entspricht, werden die Ansätze dieser Arbeiten im nächsten Abschnitt gesondert beschrieben.

5.3 Abbilden von Modelliertechnik-Beschreibungen

Ein Ziel sowohl der Arbeit von [PG89] als auch von [JF91] ist es, Modelle in einem gemeinsamen Repository zu speichern, die mit ganz unterschiedlichen Modelliertechniken beschrieben sind. Dazu müssen in den Modelliersystemen jeweils Funktionen definiert werden, mit denen die Abbildung eines Modells von der Darstellung der Modelliertechnik in die Speicherschemadarstellung durchgeführt werden kann.

In [PG89] wird auf diese Abbildung nicht explizit eingegangen. Allerdings läßt der Aufbau und die Größe des Schemas darauf schließen, daß die benötigten Abbildungen sehr einfach sind und im Prinzip immer ein Modellkonstrukt einer Modelliertechnik auf genau ein Konstrukt des Schemas abgebildet wird. In einer Folgearbeit der Autoren [PG91] wird ein System zur Integration verschiedenartiger CASE-Tools vorgestellt. Es basiert auf der Idee, daß alle Tools ihre Modelldaten im selben Repository speichern. Als Schema des Repositories wird das in [PG89] vorgestellte Schema verwendet. Zum Anzeigen der Modelle werden zwei Subsysteme zwischengeschaltet. Ein Query-Modul extrahiert die relevanten Daten aus dem Repository und ein Diagramm-Modul stellt diese Daten in der Form der gewählen Modelltechnik dar. Die Funktionsweise des Query-Moduls wird

nicht explizit beschrieben; es wird lediglich auf eine weitere Veröffentlichung der Autoren [HFG88] verwiesen. In dieser Arbeit werden Techniken vorgestellt zum Bau von in Prolog implementierten Übersetzern von einer Datenbanksprache in eine andere. Hierzu wird eine Meta-Datenbanksprache als Menge von Prolog-Klauseln definiert und gezeigt, wie sich existierende DB-Sprachen (dBaseII, CODASYL-DML, ...) in diese Metasprache abbilden lassen. Solche Übersetzer müßten geschrieben werden, wenn die CASE-Tools ihre Daten in unterschiedlichen Repositorysystemen bzw. Datenbanksystemen mit jeweils unterschiedlichen Schnittstellensprachen speichern würden. In dem System in [PG91] ist allerdings ein gemeinsames Repository mit dem in [PG89] vorgestellten Schema vorgesehen. Auf die Schnittstelle zwischen den CASE-Tools und dem Repository wird nicht näher eingegangen. Daher liefert [HFG88] keine explizite Lösung für die Funktionalität des Query-Moduls und für die Frage, wie Modelliertechnikbeschreibungen auf die Struktur des Speicherschemas abgebildet werden können oder umgekehrt.

Die Abbildung von Modellierbeschreibungen in das interne Format des Repositories wird in ARIES [JF91, JFH92] durch einzelne Übersetzer definiert. Für die Übersetzung von einer Sprache (Syntax einer Modelliertechnik) in die andere (in diesem Fall die Syntax des internen Modells) wird die Syntax beider Sprachen als Grammatik in Backus-Naur-Form definiert. Ein Übersetzer besteht dann aus einer Menge von Regeln. Jede einzelne Regel transformiert genau ein Konstrukt der Quellsprache in einen Ausdruck der Zielsprache. Für die Rückübersetzung eines Modells aus dem internen Format in die Darstellung einer Modelliertechnik wird ein zweiter Übersetzer definiert und verwendet.

Es wurden bereits einige solcher Übersetzer entwickelt [JF91]. Als Modelliertechniken wurden mehrere Spezifikationssprachen mit Textnotationen verwendet, die zum Teil von den Autoren selbst entwickelt wurden. Zusätzlich wurde ein Übersetzer entwickelt, der das interne Format in natürlich-sprachliches Englisch übersetzt. Als einzige Modelliertechnik mit größerem Bekanntheitsgrad wurden Übersetzer für ER-Diagramme entwickelt. Diese beiden Übersetzer (Hin- und Rückrichtung) sind vermutlich sehr einfach zu verwirklichen gewesen, da die Autoren darauf verweisen, daß drei ihrer fünf semantischen Aspekte (types, instances, relations) semantisch exakt den Modellkonstrukten (entitytypes, instances, relationsshiptypes) der ER-Modellierung entsprechen. Übersetzer für Kontextdiagramme, Datenfluß-Diagramme und Zustands-Übergangs-Diagramme sollen noch entwickelt werden. Da es für die Modellkonstrukte dieser Modelliertechniken keine korrespondierenden Konstrukte des internen ARIES-Modells gibt, dürfte die Abbildungslogik sehr viel komplizierter sein.

Ein großer Unterschied zwischen dem ARIES-System und einem C-CASE-System liegt in der Benutzung der Systeme durch den Modellierer. Bei einem C-CASE-System wird davon ausgegangen, daß der Modellierer die Modelle in der Darstellung der jeweils verwendeten Modelliertechnik sieht, erstellt und bearbeitet. In ARIES wird ein Modell einmal in der Notation einer beliebigen Modelliertechnik erfaßt werden. Daraufhin wird das Modell in das interne Format übersetzt und im Repository gespeichert. Für die Erweiterung des Modells oder für notwendige Änderungen wird das Modell allerdings nicht zurückübersetzt, sondern in einer systemeigenen, grafischen Darstellung weiterverarbeitet [JFH92].

Ein Modell wird nur dann in die Darstellung einer anderen Modelliertechnik zurückübersetzt, wenn ein, nicht zu ARIES gehöhrendes, Werkzeug die Eingabe des Modells in einer bestimmten Notation benötigt.

5.4 Definition von Modelliertechniken

Konzeptuelle Modelle werden (üblicherweise) von Modellierern in der Notation einer Modelliertechnik beschrieben, um über die Korrektheit und die Vollständigkeit der Abbildung eines Weltausschnittes z.B. mit den Nutzern des geplanten Systems, anderen Modellierern oder Programmierern kommunizieren[7] zu können. Voraussetzung dazu ist, daß die Kommunikationspartner dieselbe Sprache 'sprechen'. D.h einerseits, daß alle die Beschreibungskonstrukte der Modelliertechnik und deren Aufbau bzw. Zusammenhänge kennen (die Syntax der Modelliertechnik). Andererseits müssen alle Beteiligten diese Sprache interpretieren können und wissen, welcher Ausschnitt der realen Welt durch ein Modellkonstrukt repräsentiert wird.

Eine ähnliche 'Kommunikation' findet auch zwischen einem Modellierer und einem C-CASE-System statt. Der Modellierer erfaßt mit einem Werkzeug (z.B. dem generischen Modelliereditor) ein Modell in der Notation einer Modelliertechnik. Das C-CASE-System interpretiert daraufhin die Beschreibung des Modells und speichert dessen Semantik im Repository. Von jeder Modelliertechnik, die von einem C-CASE-System unterstützt werden soll, muß daher die Notation, die Syntax und die Semantik eindeutig[8] definiert sein. Hierzu ist eine rein verbale Definition der Techniken, wie sie in den meisten Lehrbüchern [Par91, Sch90b, You89, BCN91] oder auch im Abschnitt 2.1 verwendet wird, ungeeignet.

Zu einigen Modelliertechniken existieren auch formale Definitionen. Dabei werden sehr unterschiedliche Beschreibungstechniken verwendet, in denen die Definitionen der Modelliertechniken formalisiert und formuliert sind.

- Die wohl am präzisesten und formalsten definierte Basistechnik ist das Petri-Netz. Hierzu werden meist graphentheoretische Definitionen verwendet [Rei90].
- In [Egg89] wird die Syntax von Daten-Fluß-Diagrammen als Zeichenkettengrammatik [HU90] in Backus-Naur-Form angegeben.
- In [VHW91] wird für jede Modelliertechnik (als Beispiel werden DF-Diagramme und ER-Diagramme verwendet) ein NIAM-Modell[9] erstellt. Jedes Modellkonstrukt der Technik wird in diesem NIAM-Modell durch einen Entitytypen repräsentiert. Es handelt sich dabei also um ein Metamodell zur Beschreibung der Syntax einer Modelliertechnik. In diesem Metamodell können allerdings nicht alle Zusammenhänge

[7] [Ros77] – Structured Analysis (SA) : A Language for Communicating Ideas

[8] im Abschnitt 2.1 wurde bereits gezeigt, daß es zu einer Modelliertechnik verschiedene Notationen, Syntaxen und Semantiken geben kann.

[9] NIAM [NT89] ist eine Weiterentwicklung des ER-Modells [EN89] bzw. des Binary-Relationship-Modells [Mar83, EBF+87]

zwischen den verschiedenen Modellkonstrukten (Entitytypen) exakt genug beschrieben werden. Daher werden noch zusätzliche Integritätsbedingungen (sie werden in dieser Arbeit Verifikationsregeln genannt) in Form von Prädikaten der Prädikatenlogik ersten Stufe [HU92] formuliert.

- In [Hai90] wird eine Algebra für Relationen (die nicht in erster Normalform sind [SS84]) vorgestellt und gezeigt, wie sich durch die Definition von Relationen und die Anwendung von Algebraoperatoren nicht nur das klassische ER-Modell [Che76] definieren läßt, sondern auch viele andere Modelliertechniken und Konzepte, die im Rahmen der Weiterentwicklung des ER-Modells vorgestellt wurden.

- In [BM90] wird ein ER-Diagramm als knoten- und kantenmarkierter Graph definiert und die Syntax mittels einer Graphgrammatik [Nag79] festgelegt. Graphgrammatiken sind eine Weiterentwicklung von Zeichenkettengrammatiken. Sie eignen sich auch zur Definition von Operationen [Gö88], mit denen ein Graph schrittweise verändert werden kann und jeweils die syntaktische Korrektheit des Graphen gewährleistet ist.

Welche dieser Techniken sich am besten für die Verwendung in einem C-CASE-System eignet, kann natürlich nicht eindeutig beantwortet werden, da jede der Definitionsformen ihre Vorteile hat (gute Basis für theoretische Untersuchungen, gute Basis für die Implementation eines Interpretierers, leicht zu erstellende Definitionen, etc.). Auf diese Entscheidung wird in Abschnitt 6.2 nochmals eingegangen.

5.5 Visualisierung

Die Semantik von Modellen wird in den verschiedenen Techniken durch ganz unterschiedliche Konstrukte präsentiert. Es ist daher nicht möglich Präsentationsdaten, wie z.B x-y Koordinaten von Knoten, sinnvoll in einem semantischen Repository zu speichern. Daher müssen Generatoren entwickelt werden, die aus der logischen Beschreibung eines Modells eine Darstellung erzeugen, die den Präsentationsvorschriften der jeweiligen Modelliertechnik entspricht. Wie bereits in Abschnitt 3.1.1 ausgeführt, existieren zahlreiche Lösungsvorschläge für diese Problematik.

Es besteht allerdings ein großer Unterschied zwischen der Verwendung von Visualisierungsgeneratoren in einem CASE-Tool bzw. in einem C-CASE-System. In einem CASE-Tool können solche Generatoren verwendet werden, um dem Modellierer das aufwendige Formatieren des Modellgraphen abzunehmen. In einem C-CASE-System müssen sie existieren, da es möglich sein soll, daß ein Modellierer z.B. eine Menge von Prozessen und Datenflüssen mit Hilfe von SADT modelliert und dieselben Prozesse und Datenflüsse später als Daten-Fluß-Diagramm dargestellt und weiterbearbeitet werden.

Zur Definition einer Modelliertechnik im Rahmen eines C-CASE-Systems muß also nicht nur die Syntax und die Semantik beschrieben werden, sondern auch die Notation, d.h. die Darstellung der einzelnen Modellkonstrukte und deren Anordnung auf dem Bildschirm.

Ein Vorschlag, wie eine Notationsbeschreibung formuliert und vorallem gespeichert werden kann, wird in [PG91] gemacht. In dieser Arbeit wird davon ausgegangen, daß jedes grafische Objekt eines Diagramms genau einen semantischen Aspekt repräsentiert. Unter dieser Annahme kann eine direkte Beziehung zwischen einem semantischen Aspekt und seiner Darstellung in den verschiedenen Diagrammtypen hergestellt werden. Diese Beziehung wird in einem ER-Modell mit den Entitytypen "Diagrammtyp", "Grafikobjekt", "Datenobjekt" und einigen anderen beschrieben. Instanzen des Entitytypen "Grafikobjekt" sind grafische Symbole wie Rechteck, Kreis, Raute, etc. die in den verschiedenen Diagrammtypen (ERD, DFD, ...) vorkommen. Instanzen des Entitytypen "Datenobjekt" sind alle semantischen Aspekte des globalen Schemas des Repositories. Somit handelt es sich bei diesem ERM um ein Metamodell des globalen Schemas. In einem mehrschichtigen Repository, wie es der IRDS-Standard [ANS88] vorschlägt, können das globale Schema und sein grafisches Metamodell in zwei Schichten übereinander definiert werden. Damit können sowohl die Modelldaten als auch die Metadaten, die deren grafische Darstellung beschreiben im selben Repository gespeichert werden. Ein CASE-Tool mit einem Visualisierungsgenerator kann alle, zur Darstellung eines Diagramms, benötigten Daten aus einem Repository lesen.

5.6 Weitere Probleme

Problem : Speichergraphsichten
Da alle Analyseergebnisse im selben Speichergraphen repräsentiert werden, wächst dieser sehr schnell. Daher muß dem Benutzer eine Möglichkeit zur Verfügung gestellt werden, mit der er Teilausschnitte des modellierten Weltausschnitts bilden kann. Hierzu gibt es zwei, sich eventuell ergänzende Möglichkeiten. Es können zu den eigenlichen Modellieraspekten jeweils Metadaten wie Ersteller, Projekt, Abteilung, ... gespeichert werden. Diese Möglichkeit ist aus IRDS und verwandten Ansätzen [HBRY91, DK87] bekannt.

Durch den einfachen und vor allem sehr intuitiven Aufbau des globalen Schemas ist es dem Benutzer, ggf. mit entsprechender Computerunterstützung, möglich, semantische Anfragen (alle Daten zum Prozeß "Bilanz erstellen") direkt an den Speichergraphen zu stellen. Hat er einen interessanten Ausschnitt gefunden, so kann er sich diesen wiederum in verschiedenen Analysetechnikdarstellungen betrachten. Diese Darstellungen sind meist übersichtlicher und einfacher, da sie von vielen Aspekten abstrahiert und dafür die betrachteten Aspekte besonders gut darstellt.

Hierzu müßte eine geeignete Sprache [MW89, SS93] entwickelt werden, mit der Sichten des globalen Schemas definiert werden können.

Problem : Konsistenz
Selbstverständlich kann ein Benutzer einen, in einer beliebigen Analysetechnik dargestellten, Ausschnitt des Gesamtmodells nicht nur anschauen, sondern auch ändern und erweitern. Hierbei gibt es keine Update-Problematik, wie sie von Datenbanksichten bekannt ist. Der Benutzer arbeitet zwar syntaktisch, d.h. in der Darstellung seiner Analysetechnik nur

mit einem Ausschnitt, abgespeichert wird die Semantik der Änderungsoperationen aber im vollständigen Speichergraphen des Repositories. Es kann aber der Fall auftauchen, daß eine Benutzeroperation zwar syntaktisch richtig ist, semantisch aber Aspekten widerspricht, die entweder außerhalb seines gewählten Darstellungsausschnittes liegen oder in der verwendeten Analysetechnik nicht modellierbar sind.

Hierzu müßten entsprechende Konsistenzbedingungen definiert werden. In den Editor müßte eine Art Erklärungskomponete eingebaut werden, die den Modellierer in Problemfällen unterstützt [CPR91].

Problem : Unterstützen von Methoden
Eine Anforderung an ein C-CASE-System (siehe Seite 28) ist es, dem Modellierer große Flexibilität bei der Wahl der Modelliertechnik und der Reihenfolge deren Anwendung zu geben. Im Sinne der konsistenten, inkrementellen Modellierung ist allerdings nicht jede Reihenfolge sinnvoll. Als nachfolgende Modelliertechnik sollten (im allgemeinen) nur solche gewählt werden, die Aspekte des bereits existierenen Modells, durch die Modellierung weiterer Aspekte, näher beschreiben.

Hierzu müßten sinnvolle Reihenfolgen [CN86, Bai89] entwickelt werden. Zusätzlich könnte das Modelliersystem um eine Komponente erweitert werden, welche den Modellierer in der Wahl der Techniken unterstützt bzw. überwacht [VHW91, KTL+92].

Problem : Korrektes Definieren von Modelliertechniken
Ein großer Vorteil des vorgestellten Systems liegt in der Tatsache, daß die Verarbeitungslogik nicht fest in ein Programm einprogrammiert ist. Der Modelliereditor ist ein generisches Programm, daß durch die Definition der Modelliertechniken sehr flexibel steuerbar und erweiterbar ist. Dadurch liegt aber die Verantwortung für das einwandfreie Funktionieren des System bei den Erstellern der Definitionen.

Hierzu müßten Techniken entwickelt werden, die eine Überprüfung der Definitionen auf Eindeutigkeit, Widerspruchsfreiheit und Vollständigkeit ermöglichen [HU90].

6 Ausblick auf eigenen Lösungsansätze

In dieser Arbeit ist mit C-CASE eine neue Art von Modelliersystemen vorgestellt worden. Mit einem C-CASE-System lassen sich die Vorteile existierender, geschlossener Modelliersysteme (Datenintegration, inkrementelle Modellierung) mit denen offener Modelliersysteme (Erweiterbarkeit, Flexibilität) vereinen. Zur Realisierung eines C-CASE-Systems müssen einige Probleme gelöst werden, von denen im Abschnitt 5 gezeigt wurde, daß es sich dabei um offene wissenschaftliche Fragen handelt. Diese Fragen werden und wurden im Rahmen einer Dissertation an der Uni Ulm bearbeitet. Als Ausblick werden im folgenden die dabei erarbeiteten Lösungsansätze kurz skizziert.

6.1 Globales Repositoryschema

Durch die intensive Analyse einer Reihe von meist bekannten Modelliertechniken wurde ein globales Speicherschema entwickelt. Es enthält alle semantischen Aspekte, die mit den untersuchten Techniken modellierbar sind. Dabei wurde besonderer Wert auf die Ermittlung gemeinsamer semantischer Aspekte in zwei oder mehreren Modelliertechniken gelegt, da diese Gemeinsamkeiten die Basis zur widerspruchsfreien, inkrementellen Modellierung bilden.

An dieser Stelle stellt sich vielleicht nochmals die Frage, ob es überhaupt sinnvoll ist, mehreren Modelliertechniken zur Erstellung eines semantisch umfangreicheren Modells zu verwenden, wenn doch mit dem globalen Schema eine Art Super-Modelliertechnik zur Verfügung steht. Diese Frage läßt sich ganz klar mit "Ja" beantworten.

Das für diese Arbeit erarbeitete globale Schema wird als knoten- und kantenmarkierter Graph mit ungefähr 15 Knoten- und 30 Kantentypen dargestellt. Die diesem Schemagraphen zugrundeliegende Syntax enthält somit mindestens 45 verschieden syntaktische Konstrukte und eine Vielzahl von Integratätsbedingungen, da die meisten Kantentypen nur zwischen ganz bestimmten Knotentypen vorkommen dürfen. Diese Super-Modelliertechnik wäre noch umfangreicher als die im Abschnitt 2.2.2 vorgestellten, kombinierten Modelliertechniken. Sie würde somit völlig dem Abstraktionsprinzip der konzeptuellen Modellierung widersprechen, wonach versucht werden soll, zunächst die wichtigsten Aspekte eines Systems zu modellieren und von den anderen zu abstrahieren. Das Speicherschema kann sinnvoll nur vom generischen Modelleditor oder ggf. anderen computergestützten Modellierwerkzeugen gefüllt werden.

6.2 Syntaxdefinition von Modelliertechniken

Modelliertechniken sind Beschreibungsformalismen und damit im mathematisch-theoretischen Sinne Modelliersprachen. Wie bei den Programmiersprachen kann auch die Syntax einer Modelliersprache mit Hilfe einer Grammatik definiert werden. Da konzeptuelle Modelle häufig stark vernetzte Strukturen aufweisen, werden sie meist als Graph (statt als

linearer Zeichenstrom) dargestellt. Zur formalen Definition der Syntax von Modelliertechniken können daher Graphgrammatiken verwendet werden. Einige spezielle Anforderungen haben es notwendig gemacht, hierfür eine neue Art von Graphgrammatik – die H–Graphgrammatik – zu entwickeln. Für Modelliertechniken, für die noch keine graphische Notation[10] existiert, kann sehr einfach eine entsprechende definiert werden. Mit Hilfe von H–Graphgrammatiken kann somit die Syntax von Modelliertechniken nicht nur einfach und formal sondern vor allem für alle Techniken einheitlich definiert werden. Dies ist besonders wichtig, da für ein C-CASE-System ein generischer Modelliereditor entwickelt werden muß, der die Modellierung in den verschiedensten Modelliertechniken unterstützt.

6.3 Semantikdefinition von Modelliertechniken

Die Semantik der verschiedenen Modelliertechiken wurde zur Entwicklung des globalen Schemas bereits ausführlich analysiert und informal beschieben. Nachdem die Syntax einer Modelliersprache mit Hilfe einer H–Graphgrammatik formal definiert wurde, ist es auch möglich deren Semantik formal zu spezifizieren. Dazu wird in dieser Arbeit zunächst die denotationale Spezifikationsmethode verwendet. Als Zustandsraum werden die jeweiligen Modellgraphen gewählt, als semantischer Bereich dagegen für alle Modelliersprachen einheitlich der Schemagraph des semantischen Repositories. Die Semantik einer Modelliersprache wird daher festgelegt als Beschreibung des Aufbau eines semantischen Schemagraphen. Zusätzlich wird zu jeder Modelliersprache ein Operator definiert, der im semantischen Repository gespeicherte Modelldaten in der Syntax einer Modelliertechnik darstellen kann, auch wenn die Modelldaten selbst nicht in dieser Technik gespeichert wurden. Diese Operatoren definieren exakt die zur Datenintegration mehrerer Modelliertechniken benötigte Integrationsfunktionalität eines C-CASE-Systems.

6.4 Visualisierung

Für das effektive Arbeiten mit konzeptuellen Modellen ist es von großer Bedeutung, daß die Modelle übersichtlich und leicht lesbar dargestellt werden. Zu jeder der bekannten Modelliertechniken existiert mindestens eine entsprechend gute Notation. Im Schema des semantischen Repositories können allerdings bei der Erfassung der Modelle keinerlei Visualisierungsinformationen zu den Modelldaten gespeichert werden, da die Modelldaten eventuell in ganz anderen Modelliertechniken und deren Notationen wieder dargestellt werden. Daher muß ein C-CASE-System über eine Visualisierungskomponente verfügen, die einen beliebigen Modellgraphen in den verschiedenen Notationen der Modelliertechniken darstellen kann. Die Grundprinzipien einer solchen Visualisierungskomponente wurden bereits in einer Extraarbeit [Edi93] an der Uni Ulm untersucht und erfolgreich realisiert.

[10] nicht zu verwechseln mit einer gra_f_ischen Notation

7 Zusammenfassung

In dieser Arbeit wurden der Aufbau und die Funktionsweise einer neuen Art von Modelliersystem vorgestellt. Hierbei handelt es sich um ein konfigurierbares System, bestehend aus einem generischen Modelliereditor, einem semantischen Repository und einer Integrationsfunktionalität. Alle drei Komponenten eines C-CASE-Systems müssen vor dem Einsatz des Systems konfiguriert werden. Im Rahmen der Konfiguration des generischen Modelliereditors werden die Syntax sowie die Semantik von Modelliertechniken exakt definiert. Der Modelliereditor kann die entsprechend erfaßten Definitionen interpretieren und bietet dem Modellierer Funktionen zum Erstellen syntaktisch richtigen Modelle. Gespeichert wird allerdings nicht die syntaktische Darstellung der Modelle, sondern deren Semantik, in dem daher als semantisches Repository bezeichneten Speichersystem. Bei der Konfiguration des Repositories muß ein Speicherschema entwickelt bzw. verwendet werden, in dem sich die Semantik aller vom Modelliereditor unterstützter Modelliertechniken repräsentieren lassen.

Die Aufgabe der Integrationsfunktionsalität ist es, den Austausch von Modelldaten zwischen Modellen zu ermöglichen, die mit unterschiedlichen Modelliertechniken erstellt bzw. weiterverarbeitet werden. In einem Modelliersystem, das mehrere Modelliertechniken richtig unterstützen soll, ist diese Funktionalität unbedingt erforderlich, da ein und derselbe semantische Aspekt häufig in mehreren Modelliertechniken repräsentiert werden kann. Ein einmal modellierter Aspekt sollte aus Effizienzgründen in einer anderen Modelliertechnik nicht nochmal modelliert werden müssen und darf aus Konsistenzgründen nicht noch einmal modelliert werden können, da dies zu Widersprüchen führen kann.

Die Notwendigkeit für den Einsatz von Modelliersystemen, die mehrere Modelliertechniken unterstützen ergibt sich aus folgender Problematik. Die Untersuchung einiger, bekannter Modelliertechniken am Anfang dieser Arbeit zeigte, daß mit diesen Techniken nur wenige, besonders wichtige Aspekte modellierbar sind. Dies entspricht genau der Grundidee der konzeptuellen Modellierung, wonach bei der Modellerstellung einige Aspekte eines realen Weltausschnittes möglichst klar und präzise wiedergegeben, von sehr vielen anderen Aspekte aber abstrahiert werden sollen. Für viele Verwendungszwecke von Modellen genügt die Modellierung einer Sicht (statische-, dynamische-, funktionale-, etc.) nicht aus. In diesen Fällen muß entweder eine kombinierte Modelliertechnik verwendet werden, mit der mehrere Sichten dargestellt werden können, oder mehrere Basistechniken hintereinander. Bei diesem inkrementellen Modelliervorgehen ist es aber notwendig, in einer Technik modellierte Aspekte in den folgenden Techniken wiederdarzustellen und nur noch nicht modellierte Aspekte zu ergänzen. Auf diese Weise erhält man auch ein konsistentes Gesamtmodell, dessen verschiedene Sichten in jeweils unterschiedlichen Modelliertechniken dargestellt werden können. Allerdings ist der Transfer von Modelldaten zu einer anderen sowie die Darstellung dieser Daten in der anderen Technik eine schwierige und sehr aufwendige Angelegenheit, so daß eine Automatisierung dieses Vorgangs im Rahmen eines Modelliersystem auf jeden Fall wünschenswert ist.

Bei der Auswahl eines Modelliersystems stellt sich die Frage, welche Modelliertechniken von den Modelliersystem unterstützt werden müssen. Dies ist sowohl von den geplanten Verwendungszwecken der Modelle abhängig, als auch vom Kenntnis- bzw. Ausbildungsstand der Modellierer. Da sich beide Faktoren im Laufe der Zeit ändern können, ist es wünschenswert, daß das Modelliersystem, entsprechend den zu späteren Zeitpunkten geltenden Anforderungen, erweitert und modifiziert werden kann.

Als Vorschlag zur Lösung der aufgeworfenen Probleme wurde daher in dieser Arbeit eine neue Art von Modelliersystemen vorgestellt. Mit C-CASE-Systemen ist sowohl eine gute Integration von Teilmodellen möglich; sie bieten aber auch einfache Erweiterungsmöglichkeiten für die Unterstützung weiterer, bekannter oder neu entwickelter Modelliertechniken.

Danke:
Ich möchte Herrn Dadam recht herzlich danken für seinen großen Einsatz bei der Betreuung meiner wissenschaftlichen Arbeit und meines wissenschaftlichen Werdegangs.
Christian Kalus und Michael Nathe danke ich für ihre wertvollen Hinweise zu diesem Bericht.

Literatur

[ABS91] R. Andersen, J.A. Bubenko jr., and A. Solvberg, editors. *Proc. 3th Int'l Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991. Springer, LNCS 498.

[ANS88] ANSI. *Information resource dictionary system (IRDS)* . ANSI X3.138-1988, 1988.

[Bai89] S.C. Bailin. An object-oriented Requirements Specification Method . *Communications of the ACM*, 32(5):608–623, May 1989.

[Bal91] H. Balzert. *CASE - Systeme und Werkzeuge* . BI Wissenschaftsverlag, 1991.

[Bat88] C. Batini, editor. *Proc. 7th Int'l Conf. on Entity-Relationship Approach*, Rome, Italy, November 1988. North-Holland.

[BCN91] C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design* . Benjamin Cummins, 1991.

[BFN85] C. Batini, L. Furlani, and E. Nardelli. What is a Good Diagram ? A Pragmatic Approach . In ER [ER85], pages 312–319.

[BGMT89] G. Boudier, F. Gallo, R. Minot, and I. Thomas. An Overview of PCTE and PCTE+ . *SIGPLAN Notices*, 24(2):248–257, February 1989.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database-Schema Integration . *ACM Computing Surveys*, 18(4):323–364, December 1986.

[BM90] C.J. Breiteneder and T.A. Mück. A Graph Grammar Driven ER CASE Environment . In Kangassalo [Kan90], pages 375–392.

[BNT86] C. Batini, E. Nardelli, and R. Tamassia. A Layout Algorithm for Data Flow Diagrams . *IEEE Transactions on Software Engineering*, SE-12(4):538–546, April 1986.

[Boo86] G. Booch. Object-oriented development . *IEEE Transactions on Software Engineering*, SE-12(2):211–221, February 1986.

[Bro92] M. Brough. Methods for CASE: a generic framework . In Loucopoulos [Lou92], pages 523–545.

[BS82] M.L. Brodie and E. Silva. Active and passive component modelling . In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Informations Systems Design Methodologies* , pages 41–91. North-Holland, 1982.

[BTT84] C. Batini, M. Talamo, and R. Tamassia. Computer Aided Layout of Entity-Relationship Diagrams . *The Journal of Systems and Software*, 4:163–173, 1984.

[Cam86] J.R. Cameron. An overview of JSD . *IEEE Transactions on Software Engineering*, SE-12(2):222–240, February 1986.

[Che76] P.P. Chen. The Entity Relationship Model - Towards a Unified View of Data . *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[CJ90] P. Coad and E. Jourdon. *Object-oriented Analysis* . Yourdon Press, Englewood Cliffs, 1990.

[CJA89] R. Carlson, W. Ji, and A.K. Arora. The nested Entity-Relationship-Model . In F.H. Lochovsky, editor, *Proc. 8th Int'l Conf. on Entity-Relationship Approach*, pages 43–57, Toronto, Canada, October 1989. North-Holland.

[CN86] J.L. Carswell and S.B. Navathe. SA-ER: a method that links SA and ER - modeling for DB design . In Spaccapietra [Spa86], pages 381–397.

[CPR91] C. Cauvet, C. Proix, and C. Rolland. ALECSI : An Expert System for Requirements Engineering . In Andersen et al. [ABS91], pages 31–49.

[DHS+91] S. Dewal, H. Hormann, L. Schöpe, U. Kelter, D. Platz, and M. Roschewski. Bewertung von Objektmanagementsystemen für Software-Entwicklungsumgebungen . In H.-J. Appelrath, editor, *Datenbanken in Büro, Technik und Wissenschaft*, pages 352–370, Kaiserslautern, Germany, March 1991. Springer, Informatik Fachbericht 270.

[DK87] D.R. Dolk and R.A. Kirsch. A relational IRDS . *Communications of the ACM*, 30(1):48–61, January 1987.

[DK88] S. Dewal and U. Kelter. Role-Based Requirements Definition for Software Factories using Reusable Requirements Package . In K.H. Bennett, editor, *Software Engineering Environments : Research and Practice* , pages 404–411. Ellis Horwood, 1988.

[EBF+87] G.C. Everest, C. Batini, J.P. Fry, L. Mark, and P.S. Thompson. E-R Modeling versus Binary Modeling . In March [Mar87], pages 63–78.

[Edi93] Edinger. Optimierte visuelle Darstellung komplexer Graphen . Diplomarbeit, Universität Ulm, September 1993.

[Egg89] S. Eggers. Phasenintegration unter Verwendung einer Entwurfsdatenbank. Technischer Bericht 198, Universität Karlsruhe, June 1989.

[EKTW86] J. Eder, G. Kappel, A.M. Tjoa, and R.R. Wagner. BIER - The Behavior Integrated Entity-Relationship Approach . In Spaccapietra [Spa86], pages 147–166.

[EN89] R. ElMasri and S.B. Navathe. *Fundamentals of Database Systems* . Benjamin Cummins, 1989.

[ER85] *Proc. 4th Int'l Conf. on Entity-Relationship Approach*, Chicago, USA, October 1985. North-Holland.

[FGP91] M.G. Fugini, M. Guggino, and B. Percini. Reusing Requirements through a Modeling and Composition Support Tool . In Andersen et al. [ABS91], pages 50–78.

[FM86] P. Feldman and D. Miller. Entity model clustering: Structuring a data model by abstraction . *Computer Journal*, 29(4):348–360, 1986.

[FP90] M.G. Fugini and B. Percini. RECAST: A Tool for reusing Requirements . In *Proc. 2th Int'l Conf. on Advanced Information Systems Engineering*, pages 339–364, Stockholm, Sweden, 1990.

[Gö88] H. Göttler. *Graphgrammatiken in der Softwaretechnik* . Springer, Informatik Fachbericht 178, 1988.

[Gol85] A. Goldfine. The Information Resource Dictionary System . In ER [ER85], pages 114–122.

[Hai90] J. Hainaut. Entity-Relationship Models: Formal Specification and Comparison . In Kangassalo [Kan90], pages 53–64.

[HBRY91] C. Hsu, M. Bouziane, L. Rattner, and L. Yee. Information Resource Management in heterogeneous, distributed Environment . *IEEE Transactions on Software Engineering*, 17(6):604–625, June 1991.

[HFG88] D.I. Howells, N.J. Fiddian, and W.A. Gray. A source-to-source meta-translation System for Database Query Languages . In P.M.D. Gray and R.J. Lucas, editors, *PROLOG and Databases: Implementation and new Directions*, pages 23–38. Ellis Horwood, Chichester, 1988.

[HL93] H.-J. Habermann and F. Leymann. *Repository* . Oldenburg, 1993.

[HU90] J.E. Hopcroft and J.D. Ullman. *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie* . Addison Wesley, 1990.

[HU92] A.V. Hopcroft and J.D. Ullman. *Foundations of Computer Science* . Computer Science Press, 1992.

[HW93] A.H.M. ter Hofstede and T.P. van der Weide. Expressiveness in Concepual Data Modelling . *Data and Knowledge Engineering*, 10:65–100, 1993.

[Imb91] M. Imber. CASE Data Interchange Format standards . *Information + Software Technology*, 33(9):647–655, November 1991.

[ISO89] ISO. *Database Language SQL Addendum 1* . ISO/TC97/SC21/WG3, 1989.

[ISO90a] ISO. *Information Processing Systems - Information resource dictionary system (IRDS) framework* . ISO/IEC JTC 1/SC 21, 1990.

[ISO90b] ISO. *Information Technology - Information resource dictionary system (IRDS) framework* . ISO/IEC 10027, 1990.

[JF91] W.L. Johnson and M.S. Feather. Using Evolution Transformations to Construct Specifications . *IEEE Transactions on Software Engineering*, 1991.

[JFH92] W.L. Johnson, M.S. Feather, and D.R. Harris. Representation and Presentation of Requirements Knowledge . *IEEE Transactions on Software Engineering*, 18(10), October 1992.

[Kan90] H. Kangassalo, editor. *Proc. 9th Int'l Conf. on Entity-Relationship Approach*, Lausanne, Switzerland, October 1990. North-Holland.

[Kat90] R.L. Katz. Business and enterprise modeling . *IBM-Journal*, 29(4), 1990.

[Kel91a] U. Kelter. CASE . *Informatik Spektrum*, 14:215–220, 1991.

[Kel91b] U. Kelter. PCTE – Ein verteiltes Objektmanagementsystem für vernetzte Workstations . In *Proc. Workshop "Workstations: Architekturen, Anwendungen und Entwicklungstrends"*, pages 95–104. Fernuniversität, Hagen, 1991.

[Kil88] Kilberth. *JSP - Einführung in die Methode des Jackson Structured Programmimg* . Vieweg, 1988.

[KL87] W. Kozacyznski and L. Lilien. An Extended Entity-Relationship (E^2R) Database Specification and its Automatic Verifikation and Transformation into Logical Relational Design . In March [Mar87], pages 533–549.

[KS88] G. Kappel and M. Schrefl. A Behavior Integrated Entity-Relationship Approach for the Design of Object-Oriented Databases . In Batini [Bat88], pages 311–328.

[KTL+92] H. Krasner, J. Terrel, A. Linehan, P. Arnold, and W.H. Ett. Lessons Learned from a Software Process Modeling System . *Communications of the ACM*, 35(9):91–100, September 1992.

[LE89] M. Lesziak and H. Eggert. *Petrinetz-Methoden und -Werkzeuge* . Springer, Informatik Fachbericht 197, 1989.

[Lef91] H.C. Lefkovits. *IBMs Repository Manager / MVS* . QED Technical Publishing Group, 1991.

[Lou92] P. Loucopoulos, editor. *Proc. 4th Int'l Conf. on Advanced Information Systems Engineering*, Manchester, UK, May 1992. Springer, LNCS 593.

[Mar83] L. Mark. What is the Binary Relationship Approach . In C.G. Davis, S. Jajodia, P.A. Ng, and R.T. Yeh, editors, *Proc. 3th Int'l Conf. on Entity-Relationship Approach*, pages 205–220, Amsterdam, Netherlands, November 1983. North-Holland.

[Mar87] S.T. March, editor. *Proc. 6th Int'l Conf. on Entity-Relationship Approach*, New York, USA, November 1987. North-Holland.

[MM87] D.A. Marca and C.L McGowan. *SADT - Structured Analysis and Design Technique* . McGraw-Hill, 1987.

[MP86] M. Mees and F. Put. Extending a dynamic modeling method using data modelling capabilities . In Spaccapietra [Spa86], pages 399–418.

[MR86] L. Mark and N. Roussopoulos. Metadata Management . *IEEE Computer*, pages 26–36, December 1986.

[MW89] A.O. Mendelson and P.T. Wood. Finding Regular Simple Paths in Graph Databases . In P.M.G. Apers and G. Wiederhold, editors, *Proc. 15th Int'l Conf. on Very Large Databases*, pages 185–193, Amsterdam, Netherlands, August 1989. Morgan Kaufmann Publishers.

[Nag79] M. Nagl. *Graph-Grammatiken* . Vieweg, 1979.

[NT89] G.M. Nijssen and T.A.Halpin. *Conceptual Schema and Realtional Database Design : A fact oriented approach* . Prentice Hall, Englewood Cliffs, 1989.

[NT90] Nummenmaa and Tuomi. Constructing layouts for ER-diagrams from visibility-representations . In Kangassalo [Kan90], pages 315–329.

[Par91] H. Partsch. *Requirements Engineering* . Oldenburg Handbuch der Informatik 5.5, 1991.

[Per90] B. Pernici. Objects with Roles . In *Proc. th Int'l Conf. on Office Information Systems*, pages 205–215. ACM, May 1990.

[PG89] S. Papahristos and W.A. Gray. Data Dictionary Support for Integration of Systems Development Methods . In *Proc. Int'l Conf. on Advanced Information Systems Engineering*, pages 40–44, Stockholm, Sweden, 1989.

[PG91] S. Papahristos and W.A. Gray. Federated case environment . In Andersen et al. [ABS91], pages 461–478.

[PS89] C. Parent and S. Spaccapietra. Complex Objects Modeling: An Entity-Relationship Approach . In *Nested Relations and Complex Objects in Databases*. Springer, LNCS 361, 1989.

[PSM90] M. Prabandham, W.J. Selfridge, and D.D. Mann. A view of the IRDS reference model . *DB Programming + Design*, pages 40–53, March 1990.

[PSTS91] L.B. Protsko, P.G. Sorenson, J.P. Tremblay, and D.A. Schaefer. Towards the Automatic Generation of Software Diagrams . *IEEE Transactions on Software Engineering*, 17(1):10–21, January 1991.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented Modeling and Design* . Prentice Hall, Englewood Cliffs, 1991.

[Rei90] W. Reisig. *Petrinetze* . Springer, 1990.

[Rei91] U. Reimer. *Wissensrepräsentation* . Teubner, 1991.

[RG92] K.S. Rubin and A. Goldberg. Object Behavior Analysis . *Communications of the ACM*, 35(9):48–62, September 1992.

[Ros77] D.T. Ross. Structured Analysis (SA): A Language for Communicating Ideas . *IEEE Transactions on Software Engineering*, SE-3(1):16–33, January 1977.

[RS77] D.T. Ross and K.E. Schoman. Structured Analysis for Requirements Definition . *IEEE Transactions on Software Engineering*, SE-3(1):6–15, January 1977.

[Sag89] Sagawa. Repository Manager technology . *IBM system journal*, 26(2):209–227, 1989.

[Sch90a] M. Schrefl. Behavior modeling by stepwise refining behavior diagrams . In Kangassalo [Kan90], pages 113–128.

[Sch90b] A. Schulz. *Softwareentwurf* . Oldenburg Handbuch der Informatik 5.1, 1990.

[Sin88] E.J. Sinz. Das strukturierte Entity-Relationship Modell . *Angewandte Informatik*, 1988.

[SM92] S. Shlaer and S. Mellor. *Object Lifecycles: Modeling the World in States* . Yourdon Press, Englewood Cliffs, 1992.

[Spa86] S. Spaccapietra, editor. *Proc. 5th Int'l Conf. on Entity-Relationship Approach*, Dijon, France, November 1986. North-Holland.

[SPYA91] S. Spaccapietra, C. Parent, K. Yetongnon, and M.S. Abaidi. Generalization: a formal and flexible approach . *interner Bericht*, 1991.

[SS84] H.-J. Schek and M.H. Scholl. The Relational Model with Relation-Valued Attributs . *Informations Systems*, 11(2), 1984.

[SS93] G. Santucci and P.A. Sottile. Query by Diagramm: a Visual Environment for Querying Databases . *Software – Practice and Experience*, 23(3):317–340, March 1993.

[STH90] R. Spencer, T. Teorey, and E. Hevia. ER Standards Proposal . In Kangassalo [Kan90], pages 405–412.

[VHW91] T.F. Verhoef, A.H.M. ter Hofstede, and G.M. Wijers. Structuring Modelling Knowledge for CASE Shells . In Andersen et al. [ABS91], pages 502–524.

[VR92] A.A. Verrijn-Stuart and G.J. Ramackers. Model Integration in Information Planning Tools . In Loucopoulos [Lou92], pages 481–493.

[Was89] A.I. Wasserman. Tool Integration in Software Engineering Environments . In F. Long, editor, *Proc. Int'l Workshop on Environments*, pages 137–149, Chinon, France, September 1989. Springer, LNCS 467.

[Win88] J. Winkler. The ER approach and the IRDS standard . In Batini [Bat88], pages 3–19.

[Win89] P. Winsberg. Dictionary standard: ANSI,ISO and IBM . *InfoDB*, Winter 88/89:12–26, 1989.

[YC79] E. Yourdon and L Constantine. *Structured Design* . Englewood Cliffs, Prentice-Hall, 1979.

[You89] E. Yourdon. *Modern Systems Analysis* . Prentice-Hall, 1989.

[ZNG+90] J. Zhu, R. Nassif, P. Goyal, P. Drew, and B. Askelid. Incorporating a model hierachy into the ER paradigm . In Kangassalo [Kan90], pages 67–80.

Inhaltsverzeichnis