Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Technische Informationssysteme

Diplomarbeit

Dateien in föderierten Datenbanksystemen
Untersuchungen an Systemen zur Fabrikplanung

Verfasser: Andreas Köller

12. September 1996

Betreuer der Arbeit : Prof.Dr.habil. G.Saake, Universität Magdeburg ITI
Dipl.-Inf. M.Höding, Universität Magdeburg, ITI

Köller, Andreas:
Dateien in föderierten Datenbanksystemen–Untersuchungen an Systemen zur Fabrikplanung.
Diplomarbeit.
141 Seiten, 22 Abbildungen, 26 Tabellen
Otto-von-Guericke-Universität Magdeburg, 1996.

Vorwort

Die vorliegende Arbeit wurde im Frühjahr/Sommer 1996 am Institut für Technische Informationssysteme der Otto-von-Guericke-Universität Magdeburg erstellt. Das Ziel der Arbeit war, Möglichkeiten zur Integration bestimmter Softwareprodukte in föderierte Datenbanksysteme zu untersuchen. Dabei handelte es sich um Systeme, die keine Datenbankfunktionalität anbieten, sondern ihre Daten lediglich in teilweise proprietären Formaten in Dateien halten. Am Beispiel des Produktionsplanungssystems FACTOTUM wurden Analyse- und Integrationskonzepte entwickelt und in einer ausgeführten Schemaintegration erprobt. Weiterhin wurden Fragen der praktischen Realisierung und des Nutzens einer solchen Datenbankanbindung untersucht.

Danksagung

Von allen, die mich bei der Erstellung der Diplomarbeit tatkräftig unterstützten, kann ich an dieser Stelle nur einigen wenigen meinen Dank aussprechen. Ich danke insbesondere *Michael Höding*, der die Arbeit betreute und mir viele Anregungen gab, *Ingo Schmitt* und *Can Türker* für zahlreiche fachliche Diskussionen sowie *Carsten Schmidt* und *Steffen Gröpke* für ihre Unterstützung von Seiten der Fakultät für Maschinenbau. Außerdem möchte ich *Prof. Dr. Gunter Saake* für die Betreuung der Diplomarbeit danken.

Verzeichnis der Abkürzungen

DBS	*database system*, Datenbanksystem
DBMS	*database management system*, Datenbank-Management-System
DTD	*document type definition*, Dokumenttyp-Definition
FACTOTUM	*Factory Tool and Tuning Master*
FDBS	*federated database system*, Föderiertes Datenbanksystem
FDBMS	*federated database management system*, Föderiertes Datenbank-Management-System
GDM	*global data model*, Globales Datenmodell
GIM	*Generic Integration Model*, Generisches Integrationsmodell
HTML	*Hypertext Markup Language*
ISO	*International Standardization Organization*
ITI	*Institut für Technische Informationssysteme*
MS-DOS	*Microsoft Disk Operationg System*
ODBC	*Open Database Connectivity*
ODMG	*Object Database Management Group*
OpenDM	*Open Database Middleware*
PC	*personal computer*
SGML	*Standard Generalized Markup Language*
SQL	*structured query language*

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Computerbasierte Systeme spielen eine immer größere Rolle bei der Lösung von Problemen in Wissenschaft, Industrie und Verwaltung, wobei insbesondere die Verwaltung von Daten ein wichtiges Einsatzgebiet solcher Systeme ist. Die Vielzahl der angebotenen Rechnersysteme unterschiedlichster Architektur sowie die unterschiedlichen Anforderungen an eine effektive Datenhaltung führten in der Vergangenheit zur Entwicklung und zum Einsatz einer großen Zahl verschiedener Datenbanksysteme. Oftmals werden so in einem einzigen Unternehmen mehrere verschiedene Softwareprodukte verwendet, die unterschiedliche Datenmodelle oder Datenbankschemata zur Verwaltung von teilweise redundanten Daten benutzen.

Im Zuge der Rationalisierung in Unternehmen, aber auch in der Verwaltung und Forschung, entstand der Wunsch, die in unterschiedlichen Datenbanksystemen gehaltenen Daten zu integrieren, d.h. mit Hilfe eines einzigen Datenbank-*Interface* auf sämtliche Daten eines heterogenen Datenhaltungssystems zugreifen zu können und zusätzlich eine globale Konsistenzsicherung zu ermöglichen. Eine komplette Migration alter Daten in ein neues Datenbanksystem kann dabei aufgrund des Zwanges zum Investitionsschutz häufig nicht durchgeführt werden. Auch die Menge der bereits vorhandenen Daten, die Verwendung spezieller, auf den Anwender zugeschnittener Datenbanksysteme oder die zu aufwendige Umschulung der Benutzer stellen oft entscheidende Hindernisse für eine komplette Umstellung auf ein neues Datenbanksystem dar. Ein mögliche Lösung ist die Entwicklung und Implementation eines globalen Datenbanksystems, das einerseits in der Lage ist, Daten der vorhandenen Datenbanksysteme zu lesen und zu schreiben, aber das gleichzeitig den Erhalt der bereits vorhandenen Systeme einschließlich ihrer *user interfaces* gewährleistet. Eine relativ neue und vielversprechende Entwicklung auf diesem Gebiet sind *Föderierte Datenbanksysteme (FDBS)*.

Bei der Entwicklung von FDBS tauchen vielfältige Probleme auf, wie z.B. die Integration heterogener Datenmodelle oder die Behandlung der Aktualisierung

redundant gehaltener Daten. Außerdem stellt sich heraus, daß in der Praxis oftmals nicht nur Datenbanksysteme im herkömmlichen Sinne zu integrieren sind, sondern auch der Wunsch nach einer Datenbankanbindung einfacher Spezialapplikationen besteht, die keine Datenbankfunktionalität anbieten, sondern ihre Daten in Dateien mehr oder weniger komplexer Formate halten. Hier müssen Konzepte und Lösungen gefunden werden, auch solche Applikationen in ein föderiertes Datenbanksystem einzubinden.

Ein wichtiges Mittel zur Rationalisierung der Fertigung in der Industrie ist die Planung von Fabriken zur Optimierung des Produktionsablaufs. Zur Lösung von Aufgaben der Fabrikplanung existieren mathematische Verfahren, mit deren Hilfe eine solche Planung nach wissenschaftlichen Gesichtspunkten durchgeführt werden kann, die aber häufig nur mit Computerunterstützung sinnvoll einsetzbar sind. Für solche Aufgaben der Fabrikplanung wurde in der Vergangenheit eine Anzahl heterogener Teillösungen entwickelt, die sich allerdings durch die Verwendung inkompatibler Datenhaltungskonzepte auszeichnen. Das vom Institut für Arbeitswissenschaft, Fabrikautomatisierung und Fabrikbetrieb (IAF) an der Fakultät für Maschinenbau der Universität Magdeburg vorgestellte System zur Fabrikplanung FACTOTUM („Factory Tool and Tuning Master“) vereinigt verschiedene solcher Einzellösungen, verfügt jedoch nicht über ein Konzept zur gemeinsamen Datenhaltung. Eine Besonderheit der im FACTOTUM enthaltenen Werkzeuge stellt dabei die Tatsache dar, daß ihre Datenhaltung im wesentlichen dateibasiert aufgebaut ist. Diese Komponenten stellen keine datenbanktypischen Operationen zur Datenverwaltung zur Verfügung, wie etwa durch SQL-Schnittstellen, sondern legen ihre Informationen lediglich in Dateien unter dem Betriebssystems MS-DOS ab. Dies erschwert die angestrebte Anwendung eines FDBS zur Datenbankintegration des FACTOTUM. Hier sind Lösungen zu finden, die in den Spezialfällen der FACTOTUM-Komponenten eine möglichst vollständige Datenbankintegration erlauben.

Diese Arbeit beschäftigt sich mit Überlegungen, ein föderiertes Datenbanksystem zur Integration der Einzelkomponenten des FACTOTUM einzusetzen. Dazu werden zunächst die Grundlagen von FDBS erläutert. Danach werden Konzepte aufgestellt, wie dateibasierte Datenhaltungssysteme als Datenbanksysteme betrachtet und die von ihnen gehaltenen Dateien als Grundlage für eine Implementation von Datenbankschnittstellen zu einem föderierten Datenbanksystem genutzt werden können. Es schließt sich eine Einführung in das FDBS-Integrationskonzept GIM sowie eine nähere Vorstellung des PC-Arbeitsplatzes FACTOTUM an. Nachfolgend wird die vorgenommene konzeptionelle Schemaintegration der FACTOTUM-Komponenten in eine föderierte Datenbank dokumentiert. Abschließend werden Vorschläge für die Realisierung der FDBS-Integration diskutiert und die Potentiale einer FDBS-Nutzung im System FACTOTUM untersucht.

Kapitel 2

Föderierte Datenbanksysteme - Ein Überblick

Im folgenden Kapitel soll ein Einblick in die Entwicklung im Bereich föderierter Datenbanksysteme gegeben werden. Da es in diesem Bereich unterschiedliche Ansätze gibt, soll zunächst eine Klärung wichtiger Begriffe vorgenommen werden. Grundlegende Konzepte der Datenbanktheorie (z.B. [HS95]) werden für die Einführung vorausgesetzt. Einen guten Überblick über die Grundlagen von Datenbanksystemen unter spezieller Berücksichtigung von verteilten Systemen und Multidatenbanken bietet z.B. [Bob96].

2.1 Begriffe

Das in dieser Arbeit verwendete Konzept föderierter Datenbanken wurde u.a. in [SL90] vorgestellt. Da diese Arbeit einen guten Überblick über die verschiedenen Arten von Multidatenbanksystemen und über die Struktur föderierter Datenbanksysteme bietet, soll die Einführung anhand der dort vorgeschlagenen Terminologie vorgenommen werden.

Ein **Datenbanksystem (DBS)** ist die Zusammenfassung einer Software, dem *Datenbank-Management-System (DBMS)*, und einer oder mehrerer Datenbanken.

Ein lokales oder **zentralisiertes Datenbanksystem** ist die Zusammenfassung eines Datenbank-Management-Systems, das auf einem einzigen Computersystem läuft, mit einer einzigen, von ihm verwalteten, Datenbank.

Ein **verteiltes Datenbanksystem** besteht sinngemäß aus genau einem DBMS, das mehrere Datenbanken auf einem oder auf mehreren möglicherweise heterogenen Computersystemen verwaltet.

Ein **Multidatenbanksystem** ist jedes Datenbanksystem, das Operationen auf verschiedenen, gleichzeitig laufenden Komponenten-Datenbanksystemen erlaubt. Eine solche Komponente kann ein zentralisiertes oder ein verteiltes Datenbanksystem sein.

Ein Multidatenbanksystem heißt **homogen**, wenn alle beteiligten Datenbanksysteme das gleiche DBMS verwenden (also homogene Komponenten-DBS vorliegen), sonst heißt es **heterogen**.

Ein **föderiertes Datenbanksystem (FDBS,** *federated database system* **)** ist dann die Zusammenfassung eines Datenbank-Management-Systems mit mehreren Datenbanksystemen, die *zusammenarbeiten*, aber dabei in einem später definierten Sinne *autonom* bleiben. Diese Datenbanksysteme heißen **Komponenten-Datenbanksysteme**. Das DBMS heißt **föderiertes Datenbank-Management-System** und bietet kontrollierten und koordinierten Zugriff auf diese Komponenten-DBS.

FDBS sind damit ein Spezialfall von Multidatenbanksystemen (*multiple DBS*). Der wesentlichste Unterschied ist dabei, daß FDBS, im Gegensatz zu Multidatenbanksystemen, die Autonomie der Komponenten gewährleisten.

Es ist eine weitere Unterteilung föderierter Datenbanksysteme möglich. Sheth und Larson teilen in [SL90] FDBS in eng gekoppelte und lose gekoppelte Systeme. Ein FDBS heißt dabei **eng gekoppelt**, wenn das FDBS und sein Administrator für den Aufbau und den Betrieb der Föderation[1] sorgen und den Zugriff auf die Komponenten aktiv kontrollieren. Hier wird dann ein gemeinsames föderiertes Schema verwendet, das dem Benutzer des FDBS die Existenz mehrerer Komponenten-DBS verbirgt. In **lose gekoppelten** FDBS obliegt der Aufbau und Betrieb der Föderation den Benutzern, also den Komponenten-DBS und ihren Administratoren, und es findet keine Kontrolle durch die Administratoren der Föderation statt. Hier erfolgt keine Schemaintegration, und der Benutzer des FDBS sieht weiterhin verschiedene Komponenten-DBS. In der Literatur werden lose gekoppelte FDBS gelegentlich als Multidatenbanksysteme bezeichnet.

Eng gekoppelte FDBS lassen sich wiederum in einfache und mehrfache Föderationen unterteilen, wobei die Unterscheidung auf der Anzahl der verwendeten föderierten Datenbankschemata beruht. Eine graphische Veranschaulichung des Gesagten zeigt Abbildung 2.1.

[1] Der Begriff **Föderation** soll in dieser Arbeit als Kurzform für *föderiertes Datenbanksystem* verwendet werden, während **Föderierung** den Vorgang der Entwicklung einer föderierten Datenbanklösung bezeichnet.

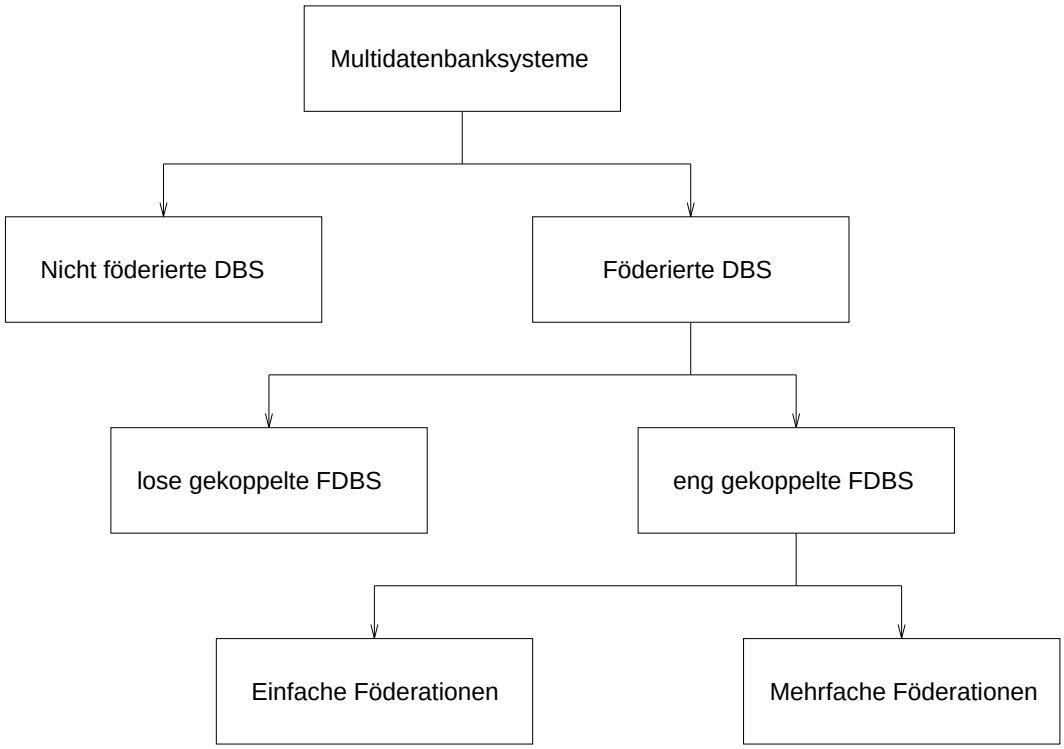

Abbildung 2.1: Begriffsbestimmung für Multidatenbanksysteme

2.2 Aufgaben föderierter Datenbanksysteme

Der Vorteil eines FDBS besteht darin, daß es in der Lage ist, Komponenten-DBS zu integrieren, die sich in solch wesentlichen Eigenschaften wie Datenmodell, Anfragesprache oder Transaktionenmodell unterscheiden. Insbesondere können so z.B. objektorientierte und relationale Datenbanksysteme in einem gemeinsamen FDBS genutzt werden. Komponenten-Datenbanksysteme können sowohl zentralisierte Einplatzsysteme, wie auch verteilte oder wiederum föderierte Datenbanksysteme sein (s. Abbildung 2.2). Ein weiterer Fall eines Komponenten-Datenbanksystems sind einfache dateibasierte Datenhaltungssysteme ohne Transaktionenmodell, wie sie im allgemeinen Programme, die keine Datenbanksysteme sind, für die Abspeicherung ihrer Daten verwenden. Dieser Fall spielt in der vorliegenden Arbeit eine wichtige Rolle und wird darum in einem gesonderten Kapitel behandelt.

Die Komponenten in einem FDBS (bzw. allgemein in einem Multidatenbanksystem) können nach verschiedenen Kriterien bewertet werden. In [SL90] werden dafür drei Konzepte vorgestellt: Verteilung, Heterogenität und Autonomie.

Das Konzept der **Verteilung** beschäftigt sich mit der Aufteilung von Daten auf verschiedene Datenbanken, die sich im allgemeinen auch auf mehreren Computer-

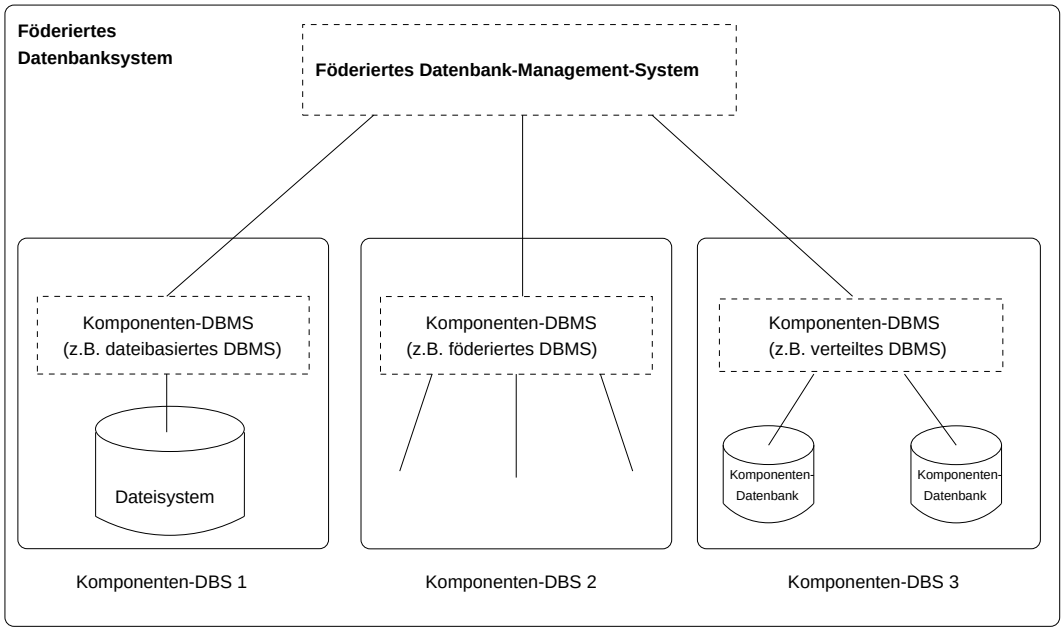

Abbildung 2.2: Möglicher Aufbau eines FDBS

systemen befinden. Dabei kann sowohl intensionale wie auch extensionale Verteilung auftreten. Vorteile einer solchen Verteilung sind z.B. eine mögliche Erhöhung der Verfügbarkeit und Verläßlichkeit von Daten sowie verbesserte Zugriffszeiten auf große Datenmengen. Für FDBS ergibt sich eine Verteilung von Daten oft zwangsläufig aus der Tatsache, daß heterogene Komponenten-DBS zusammengefaßt werden müssen.

Heterogenität entsteht in Multidatenbanksystemen aufgrund unterschiedlicher Hardware, Software, Betriebssysteme usw. Sheth und Larson unterscheiden im wesentlichen Heterogenitäten aufgrund strukturell unterschiedlicher DBMS, die man als **syntaktische Heterogenität** bezeichnen könnte, und **semantische Heterogenität**.

Formen *syntaktischer Heterogenität* sind:

- *Unterschiede in der Datenstruktur.* Verschiedene Datenmodelle bieten unterschiedliche Strukturelemente (z.B. Objekte in OODBS, Relationen in RDBS), die oft aufeinander abgebildet werden können, aber teilweise auch unlösbare Konflikte verursachen.

- *Unterschiede in Integritätsbedingungen.* Verschiedene Datenmodelle bieten unterschiedliche Konzepte zur Integritätssicherung an. In einem Modell implizit vorhandene Bedingungen (wie z.B. das Vorhandensein eines Objekts) müssen u.U. in einem anderen Datenmodell explizit modelliert werden.

- *Unterschiedliche Anfragesprachen.* Während relationale DBS oft mit SQL als Anfragesprache arbeiten, verwenden andere Systeme andere Sprachen. Auch für SQL sind verschiedene Dialekte implementiert.

Weiterhin können Unterschiede in der Fähigkeit zur Transaktionenverarbeitung, in den Anforderungen an Hardware und Betriebssysteme oder in Kommunikationsfähigkeiten auftreten.

Semantische Heterogenität bezeichnet Unterschiede in der *Bedeutung* von Daten. Zwei Datenbankobjekte in verschiedenen Datenbanksystemen könnten z.B. ähnliche Informationen über das gleiche Objekt der realen Welt enthalten, wobei sich die Daten aber in ihrer Bedeutung oder in der Form ihrer Abspeicherung unterscheiden. Eine Gleichbehandlung dieser Attribute führt dann zu Fehlern. Beispiele für solche Konflikte wären unterschiedliche Datumsformate (`12. Juni 1996`, `06-12-96`) oder eine unterschiedliche Speicherung von Umlauten.

Semantische Heterogenität ist ein umfangreiches Problem beim Entwurf föderierter Datenbanksysteme. Es gibt eine Vielzahl möglicher Konfliktarten [BH91, KCGS95], und es können recht komplexe Formen von Heterogenität auftreten. Ein Beispiel dafür sind sogenannte Metakonflikte, die in der Literatur z.T. auch als Schemakonflikte (*schematic discrepancies* [KLK91]) bezeichnet werden. Ein Metakonflikt tritt auf, wenn sich Objekte einer Klasse in *einem* Datenbankschema im Wert eines bestimmten Attributes unterscheiden, während die semantisch gleiche Unterscheidung in einem zweiten Datenbankschema durch die Zugehörigkeit der Objekte zu verschiedenen Klassen dargestellt wird. Weitergehende Untersuchungen dazu finden sich in [CL94] und werden auch am ITI durchgeführt [CHJ+96].

Bei einer Integration bzw. *Föderierung* von Datenbanksystemen entsteht die Frage, welche Rechte das föderierte Datenbanksystem an den Daten der Komponenten-DBS erhalten soll. Hieraus ergeben sich Entscheidungen bezüglich der **Autonomie** der Komponenten-DBS, also der Auswahl der Rechte, die das lokale Datenbanksystem (bzw. sein Datenbankadministrator) an das föderierte System abgeben muß. Hier kann eine Klassifikation in vier Typen vorgenommen werden:

- *Autonomie im Design.* Das Komponenten-DBS behält das exklusive Recht, die Struktur der an der Föderation beteiligten Datenbanken zu bestimmen (also z.B. den modellierten Weltausschnitt, das Datenbankschema oder die Benennung von Attributen). Diese Form der Autonomie ist im wesentlichen verantwortlich für die Heterogenität föderierter Datenbanksysteme.

- *Autonomie in der Kommunikation.* Das Komponenten-DBS behält das Recht zu entscheiden, ob eine Aufforderung zur Kommunikation mit einer anderen Komponente erfüllt wird oder nicht.

- *Autonomie in der Ausführung.* Das lokale System hat das Recht, lokale Datenbankoperationen jederzeit unbeeinflußt von anderen Datenbanksystemen auszuführen und kann bei externen Operationen die Abarbeitungsreihenfolge festlegen. Speziell kann also das FDBMS keine Operationen auf den Daten der Komponente ausführen, die lokale Integritätsbedingungen verletzen würden, und es kann keine Reihenfolge für Ausführungsschritte einer Datenbankanfrage festlegen.

- *Assoziationsautonomie.* Das Komponenten-DBS entscheidet selbst, welche Daten für die Föderation sichtbar sind und welche Operationen vom FDBS ausgeführt werden dürfen.

Es ergeben sich somit verschiedene Aufgaben, die ein FDBS bei der Integration von Komponentendatenbanksystemen zu erfüllen hat (s. auch [BEK+94]).

- Es muß ein hohes Maß an Heterogenität für die Komponenten-DBS möglich sein. Die unterschiedlichen Modelle und Schemata der Komponenten müssen isomorph auf ein globales Datenmodell und ein gemeinsames föderiertes Schema abgebildet werden. Das FDBS muß die Transformation zwischen unterschiedlichen Schemata, Daten und Operationen vornehmen. Um *Update*-Operationen auf Komponenten-DB vornehmen zu können, muß das FDBS eine eng gekoppelte Struktur aufweisen. Eine Motivation dafür findet sich in [SL90, S.204f].

- Das föderierte DBMS muß die Komponentendaten transparent für den Benutzer des FDBS darstellen, d.h. der Benutzer darf die heterogene Struktur des zugrundeliegenden Systems nicht bemerken.

- Die Integrität der Daten muß gewährleistet sein. Das umfaßt Anforderungen wie globale Konsistenz der Daten, globale Zugriffskontrolle, Datensicherheit, *concurrency control* oder *recovery*-Mechanismen.

- Eine hohe Qualität der Föderierungslösung ist die Voraussetzung für die Akzeptanz beim Benutzer.

- Für jede Komponente muß entschieden werden, in welchem Maße die Autonomie gewahrt bleiben muß bzw. kann.

Aus der Notwendigkeit, bei einer Föderierung von Datenbanksystemen einen gewissen Grad an Autonomie zu wahren, ergeben sich Probleme bei der Integration. Die Autonomie eines Komponenten-DBS kann meist nicht vollständig gewahrt bleiben, wenn das FDBS Operationen und Daten einer Komponente benutzen muß. Zumindest behindern die Autonomieforderungen aber eine effiziente Ausführung von Transaktionen des FDBS. Das stellt eines der Hauptprobleme beim Entwurf föderierter Datenbanksysteme dar.

2.3 Aufbau föderierter Datenbanksysteme

Sheth und Larson entwickeln in [SL90] eine **Referenzarchitektur** für föderierte Datenbanksysteme. Diese baut auf der 3-Ebenen-Schemaarchitektur für Datenbanken auf, die von der Arbeitsgruppe ANSI/X3/SPARC (s. Kapitel 2.4) vorgestellt wurde. Diese Schemaarchitektur wird auf 5 Ebenen erweitert, und stellt die Grundlage für die interne Realisierung föderierter Datenbanksysteme dar.

2.3.1 Komponenten der Systemarchitektur

Grundlegende Komponenten der FDBS-Architektur sind *Daten, Datenbanken, Befehle, Prozessoren, Schemata* und *Abbildungen (mappings)*. Prozessoren sind dabei Softwaremodule, die Befehle und Daten bearbeiten. Abbildungen dienen zur Übersetzung von Schemaobjekten in einem Schema in Schemaobjekte in einem anderen Schema. Es werden vier Arten von Prozessoren unterschieden.

Transformationsprozessoren transformieren Befehle zwischen verschiedenen Sprachen und Daten zwischen verschiedenen Formaten. Damit sorgen sie für *Datenmodelltransparenz*. Die zugehörige Abbildung heißt **Transformation** und stellt eine 1:1-Abbildung zwischen zwei Schemata dar.

Filterprozessor. Solche Module kontrollieren den Fluß von Daten und Befehlen, die zu einem anderen Prozessor gelangen und sorgen somit für eine *Zugriffskontrolle*. Ein **Filter** stellt eine Abbildung zwischen einem Schema und einer Untermenge dieses Schemas, dem *Subschema*, dar.

Konstruktionsprozessoren zerlegen Operationen, die von einem Prozessor erzeugt werden, in (Teil-)Operationen, die von zwei oder mehr Prozessoren ausgeführt werden können. Andererseits setzen sie Datenelemente, die von verschiedenen Prozessoren stammen, zu einem gemeinsamen Datenelement für den nachgeordneten Prozessor zusammen (Datenintegration). Konstruktionsprozessoren unterstützen die Transparenz des Ortes, der Verteilung und der Replikation von Daten und dienen somit beispielsweise zur *Schemaintegration*.

Konstruktionen sind demzufolge 1:n-Abbildungen zwischen einem Schema auf der einen Seite und mehreren Schemata auf der anderen Seite des Prozessors.

Zugriffsprozessor. Diese Prozessoren stellen Datenbankfunktionalität bereit. Sie regeln den Zugriff auf Daten einer Datenbank, indem sie Befehle in elementare Datenbankoperationen übersetzen und Daten zurückliefern. Damit sind sie die Grundlage für lokale Zugriffskontrolle, Ausführung, Backup und Recovery lokaler Transaktionen.

Da die beschriebenen Abbildungen jeweils genau einem Prozessor zuzuordnen sind, liegt es nahe, sie im Code des jeweiligen Prozessors fest zu integrieren. Das erhöht jedoch den Änderungsaufwand bei Veränderungen an einem Datenbankschema. Daher ist es besser, Abbildungen getrennt vom Prozessor als separate Datenstrukturen zu halten, um eine gute Änderbarkeit zu gewährleisten.

2.3.2 Schematypen der Referenzarchitektur

Wie bereits erwähnt, besteht die Schemaarchitektur von Sheth und Larson aus 5 Ebenen, die im folgenden beschrieben werden sollen.

Lokales Schema. Ein lokales Schema ist das Schema des Komponenten-Datenbanksystems. Es wird im Datenmodell der Komponente ausgedrückt.

Komponentenschema. Das Komponentenschema stellt die Schemainformationen des lokalen Schemas im kanonischen oder *globalen Datenmodell (GDM)* dar. Damit kann eine einheitliche Darstellung der Komponenten-DBS erreicht werden, und es können *semantische Informationen hinzugefügt* werden, die im GDM benötigt werden, aber im lokalen Schema nicht modelliert sind. (Grund dafür kann ein nicht genügend aussagekräftiges lokales Datenmodell sein. Zusätzliche Schemainformationen sind dann meist im Code der jeweiligen Applikationen enthalten.) Das setzt allerdings ein globales Datenmodell voraus, das semantisch reicher ist als das jeweilige Komponentenmodell. Im umgekehrten Fall gehen Informationen des lokalen Schemas bei der Integration verloren und müssen auf andere Weise bei der Implementation des FDBS berücksichtigt werden. Der Fall semantisch reicher GDM wird z.B. in [SCG91] propagiert, während das am ITI entstandene *Generic Integration Model (GIM)* [Sch95] einen speziellen, semantisch armen Ansatz vorschlägt und motiviert (s. Kapitel 4).

Exportschema. Anhand des Exportschemas filtert ein entsprechender Filterprozessor Daten aus dem lokalen DBS. Damit stellt ein Exportschema diejenige Untermenge eines Komponentenschemas dar, die für das FDBS sichtbar ist. Dabei ist sowohl intensionale als auch extensionale Filterung denkbar. Auch Zugriffskontrolle im Sinne einer Beschränkung der auf der Komponenten zulässigen Operationen ist im Exportschema definiert.

Föderiertes Schema. Das föderierte Schema ist die Zusammenfassung mehrerer Exportschemata. Es ist für die Sammlung von Daten *aus* den Komponenten-DBS sowie umgekehrt für die Verteilung von Daten und Befehlen *auf* die Komponenten zuständig. Je nach der Art der Kopplung im FDBS (eng oder lose) sind ein oder mehrere föderierte Schemata möglich.

Externe Schemata. Ein externes Schema definiert eine Benutzersicht auf die Föderation. Es können außerdem zusätzliche Integritätsbedingungen und Mechanismen zur Zugriffskontrolle implementiert werden. Oft wird nur das GDM als Datenmodell für die externen Schemata verwendet, es sind aber auch FDBS denkbar, die externe Schemata über einen weiteren Transformationsschritt in verschiedenen Datenmodellen modellieren können.

Je nach der speziellen Struktur des FDBS und der Komponenten können diese fünf Ebenen vollständig oder auch nur teilweise durch Transformatoren repräsentiert sein. So ist z.B. im Falle nur eines einzigen Benutzers des FDBS (nur ein externes Schema) die Integration des externen Schemas in das föderierte Schema möglich. Ein Wegfall des Exportschemas liegt vor, wenn die Komponente ihre gesamten Daten dem FDBS zugänglich macht und alle Anfragen des FDBS bearbeitet. Das lokale Schema ist identisch mit dem Komponentenschema, wenn das Komponenten-DBS das globale Datenmodell das FDBS selbst verwendet.

Abbildung 2.3 zeigt ein Beispiel für ein FDBS mit mehreren unterschiedlichen Komponenten, bei denen einzelne Schemaebenen zusammenfallen.

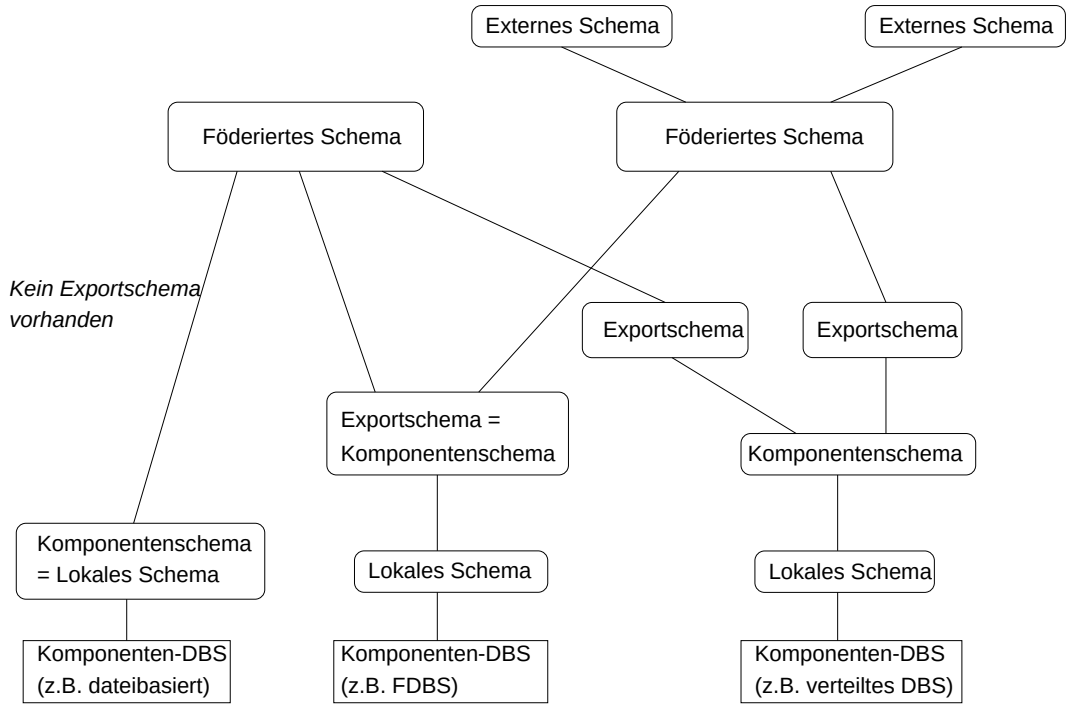

Abbildung 2.3: Ein Beispiel für eine FDBS-Implementation

Ein weiteres Schema, das in einigen FDBS-Ansätzen zu finden ist [SL90], stellt das **Hilfsschema** dar. Es steht in Relation zum Konstruktionsprozessor, steht also in der Hierarchie zwischen den Exportschemata und dem föderiertem Schema. Im Hilfsschema können Informationen modelliert werden, die vom Nutzer des

föderierten Datenbanksystems benötigt werden, aber in keinem der lokalen DBS vorhanden sind. Außerdem können Hilfsinformationen wie z.B. Konvertierungstabellen oder Statistiken für die Anfrageoptimierung gehalten werden. Auch sind weitere Filterebenen z.B. zur Konsistenzsicherung über mehrere Exportschemata (also Komponenten-DBS) denkbar.

Die Verfügbarkeit einsatzfähiger föderierter Datenbanksysteme ist zur Zeit aufgrund der vielfältigen noch zu lösenden Probleme bei der Implementation noch recht gering. Ein vielversprechender Ansatz in Richtung eines voll funktionalen föderierten Datenbanksystems ist das von *C-LAB* in Paderborn entwickelte OpenDM/Efendi, auf das in Kapitel 8.2 näher eingegangen wird. Dort werden auch nähere Hinweise auf den möglichen Aufbau eines föderierten Datenbanksystems gegeben.

Am Institut für Technische Informationssysteme der Universität Magdeburg (ITI) werden im Rahmen des Forschungsprojektes **SIGMA**$_{FDB}$[2] Untersuchungen angestellt, wie mit der Hilfe föderierter Datenbanksysteme eine effiziente Integration heterogener Softwaresysteme erfolgen kann. Dabei findet eine Konzentration auf die Schaffung einer geeigneten Informations- und Kommunikationsinfrastruktur im Bereich Fabrik- und Anlagenplanung statt [HTJ+95]. Anhand dieses Beispiels sollen geeignete Konzepte, Algorithmen und unterstützende Software-Werkzeuge entwickelt werden, die eine FDBS-Integration ermöglichen und vereinfachen.

Die vorliegende Arbeit beschäftigt sich mit der Untersuchung eines praktischen Beispiels aus dem genannten Bereich, an dem die Integration heterogener Komponenten in ein föderiertes Datenbanksystem untersucht und dargestellt werden soll.

2.4 Weitere Ansätze zu heterogenen Systemen

2.4.1 Multidatenbanksysteme

Wie bereits angedeutet, existieren neben dem Ansatz föderierter Datenbanksysteme noch andere Lösungen, zu einer Integration heterogener Komponenten-DBS zu kommen.

Der wesentliche Unterschied zu föderierten Datenbanken besteht dabei im Verlust der Autonomie der Komponenten. In einem nicht-föderierten Datenbanksystem existiert nur eine Datenbank-Management-Ebene, und es wird nicht zwischen lokalen und globalen Benutzern unterschieden.

Gemeinsam ist allen Multi-DBS-Ansätzen die Integration der Komponenten in einer Form, die dem Benutzer der globalen Anwendung die Existenz lokaler An-

[2]Schema Integration and Global Integrity Maintenance Approach for Federated Data Bases

wendungen verbirgt, diese also „transparent“ macht. Das System teilt eine Datenbankanfrage, die durch den globalen Benutzer gestellt wird, in Anfragen für die einzelnen Systeme. Umgekehrt werden die Antwortdaten für den Benutzer wieder „zusammengesetzt“.

Grundlegend sind zwei Typen von Multi-DBS möglich [Bob96]:

- **Globales Schema**
  Multi-DBS-Architekturen mit globalem Schema, zu denen auch die föderierten Datenbanksysteme gehören, integrieren die lokalen Datenbankschemata zu einem globalen Schema, d.h. das globale Schema bildet die Vereinigungsmenge aller lokalen konzeptionellen Schemata.

  Die Grundlage für die meisten dieser Ansätze bildet das ANSI/SPARC-Modell [TK78]. Es besteht aus drei Stufen: dem *Internen Schema*, das physikalische Informationen über die Abspeicherung der Daten in einer Komponente enthält, dem *Konzeptionellen Schema* mit Informationen über die logischen Datenstrukturen (*data dictionaries*), und den *Externen Anwendersichten* (Abbildung 2.4).

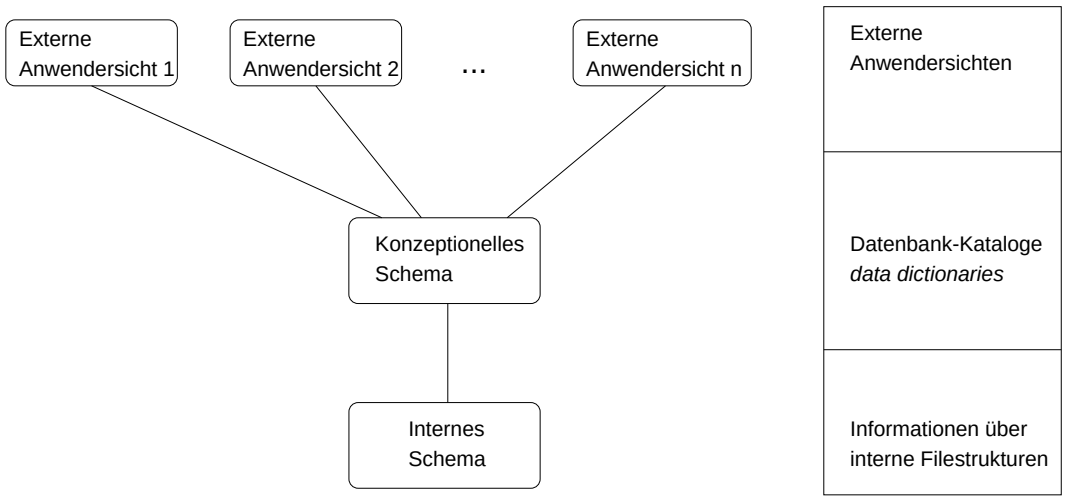

Abbildung 2.4: Das ANSI/SPARC-Modell für Multidatenbanksysteme

  Eine gewisse Autonomie der Komponenten bleibt gewahrt, da lokale externe Schemata definiert werden können, die lokale Applikationen repräsentieren.

- **Multidatenbanksprachen**
  Ein zweiter Ansatz zur Integration von Multi-DBS, der in [BHP92] vorgeschlagen wurde, kommt ohne globales Schema aus. Die Anfragesprache enthält Sprachelemente, die die Verteilung der Anfragen und die Zusammenfassung der zurückgelieferten Daten vornehmen. Mit diesen Sprachelementen muß der Benutzer Anfragen aufbauen.

Diese Form der Multi-DBS garantiert eine hohe Autonomie der Komponenten, erhöht aber auch die Komplexität der Erzeugung verteilter Anfragen. Die Verantwortung für die Auffindung von Daten und deren Anpassung wird dem Programmierer überlassen. Das stellt eine Fehlerquelle dar und verringert wesentlich die Flexibilität eines solchen Systems. Die Anpassung an sich verändernde oder neue Komponenten-DBS wird erheblich erschwert, insbesondere gegenüber dem föderierten Ansatz.

Föderierte Datenbanksysteme sind nach [Bob96] ein Spezialfall von Multi-DBS, die weitgehende Autonomie bieten und nicht auf globale *data dictionaries* angewiesen sind, um verteilte Anfragen zu erzeugen. Die Anfragegestaltung erfolgt mittels der bereits beschriebenen Exportschemata. Außerdem sind Mitglieder einer Datenbankföderation in der Lage, die Föderation zu betreten oder zu verlassen, ohne andere Mitglieder zu beeinflussen.

Weitere Analysen zum Stand der Entwicklung heterogener Datenbanksysteme zeigt E. Rahm auf. In [Rah94] stellt er verschiedene Ansätze zur Unterstützung heterogener Datenbanken gegenüber und ordnet sie in ein Spektrum steigender Kooperation und sinkender Heterogenität ein. Danach stehen föderierte Datenbanksysteme zwischen verteilten Transaktionssystemen, die eine höhere Heterogenität, aber auch geringere Verteilungstransparenz und Kooperation aufweisen, und verteilten DBS, die im hier definierten Sinne nicht mehr heterogen sind.

Nach Ansicht des Autors stellen föderierte Datenbanksysteme im Ansatz eine zu bevorzugende Lösung der Integrationsproblematik heterogener Systeme dar. Sie zeichnen sich durch eine klare Strukturierung aus, so daß schrittweise Systemanalyse und Systemdesign möglich sind. Außerdem sorgen sie für eine hohe Transparenz der Strukturen beteiligter Systeme, die eine Benutzung des fertigen Datenbanksystems sehr vereinfacht. So ist der Benutzer nicht gezwungen, eine komplizierte und mächtige Anfragesprache zu erlernen, um verteilte Datenbankanfragen zu stellen. Außerdem stellen föderierte Datenbanksysteme derzeit den flexibelsten vollständigen Ansatz der Multidatenbank-Integration dar, in dem völlig heterogene Datenbanksysteme, aber auch andere Software-Produkte vorteilhaft zusammengefaßt werden können.

2.4.2 Weitere Integrationsverfahren

Neben dem föderierten Ansatz gibt es noch weitere Möglichkeiten der Datenintegration [Hüs95]. Ein grundlegendes Konzept für die Kooperation verschiedener Datenhaltungssysteme sind **Gateways**. Dabei handelt es sich um eine Einrichtung, die Anfragen in einer Datenmanipulationssprache (DML) in entsprechende Anfragen einer anderen DML übersetzt, und somit den einheitlichen Zugriff auf heterogene Datenbanksysteme regelt. Da mit diesem einfachen Kon-

zept jedoch keine Schemaanpassungen vorgenommen werden können, ist sein Einsatz auf Spezialfälle begrenzt. Es existieren einige *Gateway*-Produkte, die DML-Transformationen zwischen bestimmten Datenbanksystemen vornehmen können (z.B. ein ADABAS-D–DB2–*Gateway* oder INGRES-*Gateways* nach DB2 und IMS). *Gateways* existieren zur Zeit hauptsächlich für Produkte, die auf relationalen Datenmodellen aufbauen. Einige objektorientierte Datenbanksysteme (*ONTOS*) bieten jedoch auch bereits *Gateway*-Funktionalität an.

In [Hüs95] werden weitere Konzepte aus dem Umfeld der Datenbankintegration verglichen. Als wichtige Entwicklungen stellen sich dabei *Data Warehousing*, Datenreplikation und die *ODBC*-Schnittstelle *Open Database Connectivity* dar. Besonders letztere, eine auf SQL basierende Entwicklung aus dem PC-Bereich, hat sich als Quasi-Standard für den Zugriff auf Datenquellen durch Applikationen (allerdings hauptsächlich unter *Microsoft WINDOWS*) durchgesetzt.

Einige kommerzielle Produkte (*INFORMIX-Enterprise Gateway, IBM Data Joiner*) gehen mit ihrer Funktionalität bereits über das Gateway-Konzept hinaus und ermöglichen Verbindungen zwischen mehreren verschiedenen Datenbanksystemen. Damit bieten sie bereits einige Grundfunktionen föderierter Datenbanksysteme an. Es fehlen allerdings Eigenschaften wie ein globales Datenbankschema, ein globaler Zugriffsoptimierer oder die Formulierbarkeit globaler Integritätsbedingungen.

Die genannten Ansätze bieten einige Integrationskonzepte an, sind jedoch nicht vollständig und daher nur für spezielle Aufgabengebiete geeignet. Wenn auch die Verbreitung einiger dieser Integrationslösungen zur Zeit wesentlich stärker ist als die föderierter Datenbanksysteme, so ist doch zu vermerken, daß eine zufriedenstellende Integration beliebiger Komponenten-Softwaresysteme eine globalere und umfassendere Lösung erfordert als sie *gateways*, ODBC, oder *Data-Warehousing*-Produkte bieten können.

Kapitel 3

Dateien in föderierten Datenbanksystemen

Das zentrale Thema dieser Arbeit ist die Untersuchung von Möglichkeiten zur Föderierung dateibasierter Datenhaltungssysteme. Unter einem **dateibasierten Datenhaltungssystem** soll ein Softwaresystem verstanden werden, das keine dokumentierte Schnittstelle für den Zugriff auf seine Daten durch andere Applikationen anbietet. Ein einfaches Offenlegen des verwendeten Dateiformats soll dabei nicht als Schnittstelle betrachtet werden. Solche Systeme stellen den Normalfall bei Applikationen dar, die nicht primär für den Einsatz als Datenbanksystem entwickelt wurden.

Dafür gibt es mehrere Gründe:

- Die Implementation einer Datenbankschnittstelle wie beispielsweise SQL oder ODBC stellt bei der Entwicklung einer Applikation einen erheblichen Aufwand dar, der oft nicht gerechtfertigt ist.
- Die von der Applikation gehaltenen Daten stellen aufgrund ihres geringen Umfanges oder der Datenstruktur kein geeignetes „Datenmaterial“ im Sinne einer Datenbank dar. Allerdings kann der Umfang vorhandenen Datenmaterials während des Betriebs einer Anwendung stark anwachsen, so daß eine anfänglich nicht notwendige Datenbankintegration nachträglich wünschenswert erscheint.
- Der Hersteller einer Applikation ist nicht bestrebt, das Datenformat seiner Applikation offenzulegen, um unerwünschte Einblicke in ihre Funktionsweise zu erschweren.

Trotz dieser Einschränkungen halten dennoch die meisten Applikationen Daten in irgendeiner Form, wobei die Abspeicherung in Form einer oder mehrerer Dateien geschieht, die unter bestimmten Dateinamen an definierten Stellen eines

Dateisystems abgelegt werden. Ist nun der Inhalt einer solchen Datei hinreichend aussagekräftig, um Informationen über den Zustand einer Applikation abzuleiten (also z.B. Daten über ein bestimmtes Projekt zu erhalten, das mit der Anwendung bearbeitet wurde), stellt sich die Frage, ob es sinnvoll ist, die enthaltenen Daten mittels geeigneter Verfahren aus diesen Dateien zu extrahieren, um sie in einem Datenbanksystem einsetzen zu können. Selbstverständlich schließt diese Extraktion auch den umgekehrten Vorgang des Updates lokaler Dateien ein. Ist eine solche Integration erwünscht und sinnvoll, kann die betreffende Applikation als *Datenbanksystem* betrachtet werden. Dabei sind die hier vorgestellten Konzepte nicht nur für eine Datenbankintegration mit föderierten DBS interessant, sondern sie könnten auch hilfreich für die Entwicklung von *Data Warehousing*-Systemen sein.

Für die Lösung von Spezialaufgaben mit Hilfe von Computern stehen zahlreiche Einzellösungen aus allen Gebieten von Wissenschaft, Technik, Wirtschaft und Verwaltung bereit. Oft bearbeiten solche Systeme jedoch nur Teilaspekte eines Problems. Um zu einer kompletten Lösung einer Aufgabe zu kommen, müssen dann mehrere solcher Spezialapplikationen verwendet werden. Hierbei ist problematisch, daß unterschiedliche Software, die von verschiedenen Herstellern stammt, nur in sehr seltenen Fällen zueinander kompatible Schnittstellen anbietet. Oft müssen Daten, die mit Hilfe einer bestimmten Applikation gewonnen wurden, mangels geeigneter Import- oder Exportfunktionen manuell in ein anderes System übertragen werden.

Da jedoch die meisten Applikationen in irgendeiner Form Daten in Dateien speichern, können die in diesen Dateien enthaltenen Informationen für eine effektivere Übertragung von Daten zwischen verschiedenen Systemen genutzt werden. Hier bietet sich eine Einsatzmöglichkeit für föderierte Datenbanksysteme. Unter bestimmten Voraussetzungen können heterogene dateibasierte Systeme in einem FDBS zusammengefaßt werden, wodurch die Vorteile einer Datenbankintegration (einheitlicher Zugriff, Konsistenzsicherung, verteilte Anfragen und Updates) genutzt werden können.

Einige dieser Voraussetzungen sollen näher erläutert werden:

- *Verwendung von Dateien zur Datenhaltung.*
  Die Applikation muß permanent Dateien zur Datenhaltung benutzen. Einige Anwendungen lesen die von ihnen zu bearbeitenden Daten im Dialog mit dem Benutzer ein und geben Ergebnisse nur auf dem Bildschirm oder Drucker aus. Da hier kein Ablegen von Daten in permanenter Form stattfindet, kann keine Integration als Datenbanksystem erfolgen.

- *Aussagekräftiges Dateiformat.*
  Das von der Applikation zur Abspeicherung der Daten verwendete Format muß in geeigneter Form strukturiert sein, um eine Extraktion enthaltener

Informationen zu ermöglichen. Dazu muß das Dateiformat entweder offengelegt bzw. standardisiert sein, oder es muß wenigstens möglich sein, durch Analyse der Dateien zu überprüfbaren Annahmen über die Bedeutung der einzelnen Datenelemente zu kommen. Das ist im Falle nicht offengelegter Formate in der Regel nur dann möglich, wenn die Dateien in einer textlich lesbaren Form vorliegen. Allerdings könnte mit Hilfe geeigneter zu entwickelnder Tools eine Analyse auch solcher Dateien versucht werden. Bestimmte Strukturinformationen wie z.B. eine gewisse Regelmäßigkeit in den Daten einer Datei könnten Anhaltspunkte für eine erfolgreiche Strukturanalyse bieten. Ein Überblick über verschiedene für eine Integration geeignete Dateiformate wird in Kapitel 3.2 gegeben.

- *Zugänglichkeit der Dateien.*
  Die Dateien, die für die Föderation interessante Informationen enthalten, sind reguläre Dateien im jeweils benutzten Filesystem. Um eine lauffähige föderierte Datenbank zu implementieren, ist es nötig, zur Laufzeit der lokalen Applikationen Zugriffe auf diese Dateien durchzuführen. Das führt abhängig vom Betriebssystem (insbesondere *MS-DOS*) zu Problemen, da u.U. Dateien nur von je einem Prozeß geöffnet werden dürfen. Unter *MS-DOS* und *WINDOWS 3.11* kommt es zu weiteren Schwierigkeiten, da kein präemptives Multitasking möglich ist und so ein föderiertes Datenbank-Management-System nicht parallel zu den lokalen Applikationen unkoordinierte Dateizugriffe ausführen kann. Daher scheidet eine reine *MS-DOS*-Lösung für ein FDBS auf dateibasierten Komponenten von vornherein aus.

Sind die genannten Voraussetzungen erfüllt, kann das lokale Datenhaltungssystem als Datenbanksystem betrachtet werden, und es kann eine Integration im herkömmlichen Sinne stattfinden. Dabei sind jedoch einige Einschränkungen und Besonderheiten zu beachten, die im folgenden erläutert werden sollen.

3.1 Besonderheiten dateibasierter Systeme

Dateibasierte Datenhaltungssysteme im vorstehend erläuterten Sinn bieten keine Funktionalität für den Datenzugriff zur Laufzeit an. Insbesondere sind sie nicht in der Lage, Datentransaktionen durchzuführen, d.h. es können vom FDBS keine Anfragen gestellt und keine Ergebnisse zurückerwartet werden. Daher müssen spezielle Transformatoren für die Übersetzung von Datenbankanfragen durch das FDBS und die Rückübersetzung der Ergebnisse eingesetzt werden, die in der Lage sind, zur Laufzeit einer lokalen Applikation auf deren Dateien zuzugreifen und diese zu lesen oder zu verändern.

Da eine lokale Applikation von diesen Vorgängen keine Kenntnis hat, stellt sich insbesondere die Frage der *Autonomie* der Komponenten völlig anders dar. Das

lokale System kann prinzipiell nicht für eine Kontrolle des Zugriffs durch das föderierte DBMS sorgen, so daß die Verantwortung für Datensicherheit und Konsistenz allein beim FDBS liegt. Der Grad der lokalen Autonomie kann lediglich dadurch bestimmt werden, daß nur ein Teil der aus den Dateien der Applikation extrahierten Informationen in das integrierte Schema aufgenommen wird. Außerdem muß das FDBS dafür sorgen, daß beim Rückschreiben von Dateien (bei Datenbankupdates) keine Inkonsistenzen in der Applikation auftreten. Besonders bei längeren Schreiboperationen auf häufig zugegriffenen Dateien ist die Wahrscheinlichkeit sehr hoch, daß während des Schreibens durch das FDBS die Applikation aus der Datei liest oder versucht, in sie zu schreiben. Da die Applikation im allgemeinen keine Kenntnis von der Existenz eines föderierten Systems hat und daher unsynchronisierte Zugriffe des FDBS und der lokalen Anwendung auf die gleichen Dateien stattfinden, können solche Kollisionen auftreten und müssen geeignet behandelt werden. Dies kann zu Problemen führen, da Konzepte zur Behandlung von Zugriffssperren für die lokalen Systeme oft nicht existieren (*MS-DOS*-Anwendungen). Eine Sperrung kann dann nur durch ein übergeordnetes System (File-Server) erfolgen, was zu Problemen führen kann.

Dieses Thema ist z.Z. Forschungsgegenstand und bedarf weitergehender Untersuchungen. Erste Ergebnisse dieser Arbeiten werden in Kapitel 8.3 vorgestellt.

Ein weiteres Problem, das bereits angesprochen wurde, ist die weite Verbreitung nicht-paralleler Betriebssysteme wie *MS-DOS* mit *Windows 3.11* und, in gewisser Hinsicht, auch *Windows 95*. Diese Systeme erlauben keinen echt parallelen Dateizugriff und stellen auch keine Funktionalität für eine unabhängige parallele Abarbeitung mehrerer Prozesse zur Verfügung. Liegen nun einige oder alle der zu integrierenden Komponenten unter einem dieser Betriebssysteme vor, läge der Nutzen einer FDBS-Implementation unter dem gleichen Betriebssystem aus Aufwandsgründen nahe. Das ist jedoch nicht möglich, da aus den bereits erwähnten Gründen parallele Abarbeitung und Dateizugriff für das FDBS unerläßlich sind. Aktuelle Überlegungen gehen dahin, für das FDBS und die Transformatoren das Betriebsystem UNIX zu benutzen und dann alle Dateien der beteiligten Applikationen unter einem UNIX-Filesystem zu verwalten. Der Zugriff auf die Daten durch ihre Applikationen geschieht dann über einen geeigneten Fileserver, der für die lokalen Systeme wie ein gewöhnlicher Massenspeicher erscheint. Der Vorteil dieser Lösung liegt in der Möglichkeit, alle Dateizugriffe der Komponenten abzufangen und auszuwerten. Nachteilig ist, daß im Falle DOS-basierter Komponenten in allgemeinen mindestens zwei physikalische Systeme (also Computer) benutzt werden müßten, da eine zufriedenstellende Emulation eines *MS-DOS*-PC unter UNIX z.Z. nicht möglich ist (s. auch Kapitel 8.3).

3.2 Geeignete Dateiformate

Einige Formen der Datenhaltung durch dateibasierte Systeme eignen sich besonders für eine Integration in ein föderiertes Datenbanksystem. Das sind insbesondere Dateien, deren Struktur sich durch Betrachten und geeignete Experimente erkennen läßt (oder ohnehin offengelegt ist).

Dabei ist festzustellen, daß abhängig vom Format eine Datei in unterschiedlichem Maße Informationen über ihre eigene Struktur, sogenannte *Metainformationen* enthält. So ist der Name einer Datei ein Merkmal für ihren Typ und somit für ihre Struktur. Einige Dateien veränderlicher Struktur enthalten auch Informationen über die jeweilige Instanz am Beginn der Datei in einem Header.

Es sollen die wichtigsten dieser Formate beschrieben werden, wobei sich die Terminologie an der in [Höd96] vorgeschlagenen Bezeichnungsweise ausrichtet. Diese Klassifizierung wird auch im Anhang bei der Untersuchung der im Produktionsplanungssystem FACTOTUM verwendeten Dateien angewandt.

dBase-**ähnliche Tabellen.** Unter *dBase*-ähnlichen Tabellen sollen Dateien verstanden werden, die Informationen über ihre Struktur selbst enthalten, gewöhnlich in Form eines Anfangsblocks vor den eigentlichen Daten. Dadurch kann die Datenstruktur einer Datei relativ flexibel gehalten werden, wobei die Lesbarkeit und Portabilität nicht eingeschränkt wird. Solche Dateiformate werden vorzugsweise von Datenbanksystemen im PC-Bereich (*xBase*) verwendet und sind daher meist standardisiert. Die Ableitung eines relationalen Schemas zur Abbildung einer solchen Datei ist dann relativ einfach.

Stark strukturierte Texte. Als stark strukturierte Texte sollen Dateien bezeichnet werden, die zwar eine komplexe Satzstruktur mit unterschiedlicher Recordlänge aufweisen, bei denen aber *jedes* Datenelement (Attributwert) mit einem entsprechenden Schlüsselwort (*token*) eingeleitet wird. Bei Kenntnis der *tokens* und der Feinstruktur der auf sie folgenden Daten kann eine entsprechende Grammatik konstruiert werden, auf deren Grundlage ein *parser* für die Datei erzeugt werden kann. In der vorliegenden Arbeit soll versucht werden, anstelle der Konstruktion einer vollständigen Grammatik eine Beschreibung solcher Dateien mittels der Dokumentbeschreibungssprache SGML vorzunehmen (s. S. 23). Typische Vertreter solcher Dateiformate sind Grafikformate wie DXF.

Hypertexte. Ein Hypertext ist eine Datenbank miteinander verbundener Dokumente (Texte und Hypertexte) mit einem Benutzerinterface, das es ermöglicht, mit der Hilfe von *links* zwischen den einzelnen Dokumenten zu „springen“.

Die Dokumente, aus denen ein Hypertext besteht, stellen in einem gewissen Sinne einen Spezialfall stark strukturierter Dateien dar, da auch dort alle enthaltenen Datenelemente mit Hilfe von *tokens* eingeleitet und beendet werden. Ein bekanntes Beispiel für ein Hypertextformat ist das bekannte HTML-Format (*hypertext markup language*), das in letzter Zeit durch das *World Wide Web* eine starke Verbreitung erfahren hat. Da z.B. HTML eine Instanz der Sprache SGML darstellt, bietet sich hier eine SGML-Modellierung geradezu an.

Satzorientierte Dateien. Das Unterscheidungsmerkmal dieser Dateien ist ihre Struktur mit Datensätzen gleicher Länge, wie sie meist bei dBase-ähnlichen Tabellen auftreten, aber ohne einleitende Informationen im Sinne einer Strukturbeschreibung. So können Informationen über die Bedeutung einzelner Elemente eines Records nur über eine Analyse des erzeugenden Anwendungsprogramms erfolgen. Satzorientierte Dateien können auch als Spezialfall stark strukturierter Dateien mit *tokens* der Länge 0 betrachtet werden, bei denen die Zuordnung von Bytes in der Datei zu Datenelementen nur aufgrund ihrer Position in der Datei ermittelt werden kann. Satzorientierte Dateien finden sich häufig bei der Abspeicherung einfach strukturierter Informationen in Spezialapplikationen wie den hier analysierten FACTOTUM-Komponenten.

Schwach strukturierte Dateien. Im Gegensatz zu stark strukturierten Dateien wird in schwach strukturierten Formaten nur *ein Teil* der enthaltenen Informationen mittels *token* eingeleitet. Auch liegt keine Struktur mit Datenelementen fester Länge vor. Beispiel für solche Formate sind Texte, in denen an beliebigen Stellen Datenfelder wie z.B. Telefonnummern in einem maschinell erkennbaren Format vorkommen können.

Binärdateien. Bei allen bisher besprochenen Dateiformaten wurde vorausgesetzt, daß zur Abspeicherung nur der 7-Bit-ASCII-Code oder ein ähnlicher Code zur Beschreibung lesbarer Texte verwendet wird. Benutzt eine Applikation ein Datenformat, das ganz oder teilweise binäre Daten verwendet, ist eine Beschreibung und besonders eine Entschlüsselung nicht offengelegter Formate sehr schwierig. Hier könnte evtl. spezialisierte Software eine Unterstützung bieten, die anhand regelmäßiger Strukturen oder bestimmter Bitmuster in einer Datei versucht, eine Beschreibung des Dateiformates zu geben oder Datenelemente zu identifizieren. Ein noch relativ leicht beherrschbarer Spezialfall könnte ein binär abgespeichertes Datenelement in einer stark strukturierten Datei sein (z.B. ein Inline-Bitmap in einem SGML-Dokument). Eine vollständige Beschreibung einer binären Datei dürfte jedoch nur in Spezialfällen gelingen. Benutzt eine zu integrierende Komponente in einem FDBS ein solches Dateiformat, kann oft keine Ableitung eines lokalen Schemas erfolgen.

Allen beschriebenen Dateiformaten ist gemeinsam, daß nicht nur der Dateiinhalt, sondern auch der Dateiname und der Pfad der Datei (also die Dateiposition) semantisch wichtige Informationen darstellen. Oft wird auf bestimmte Datenelemente unter Verwendung des Namens der sie enthaltenden Datei referenziert, oder es werden Dateinamen abhängig von der Semantik der enthaltenen Daten aus flexiblen und festen Elementen (z.B. projektspezifischer Name und feste Endung unter MS-DOS) zusammengesetzt. Die Position einer Datei in einem bestimmten Pfad gibt in vielen Applikationen ihre Zugehörigkeit zu einem bestimmten Projekt oder einer anderen logischen Einheit wieder.

Diese Klassifizierung erhebt keinen Anspruch auf Vollständigkeit. Es sind weitere Datenformate möglich, die sich nicht eindeutig einer der hier beschriebenen Kategorien zuordnen lassen. Alle im System FACTOTUM enthaltenen Dateien lassen sich jedoch mit Hilfe der hier vorgenommenen Klassifizierung beschreiben.

3.3 Beschreibung von Dateien mittels SGML

Im Prozeß der Integration dateibasierter Systeme in ein Datenbanksystem steht am Anfang der Implementation die Frage nach einer Abbildung der Struktur der verwendeten Dateien auf ein Datenbankschema, d.h. nach der Ableitung einer ausreichenden Dateibeschreibung, um Transformatoren für die Anbindung der Dateien an den FDBS-Kern aufbauen zu können. Zunächst stellt sich eine Datei für einen solchen Transformator wie eine unstrukturierte Abfolge binärer Datenelemente (*bytes*) dar, aus der durch Finden geeigneter Strukturinformationen Datenelemente gelesen werden müssen. Natürlich muß auch der umgekehrte Fall des Schreibens solcher Dateien berücksichtigt werden.

Wie im vorhergehenden Kapitel ausgeführt wurde, finden sich in allen für eine Integration geeigneten Dateiformaten genau abgegrenzte Datenelemente, auf deren Bedeutung sich durch geeignete Überlegungen und Experimente schließen läßt. Ist dieser gedankliche Prozeß abgeschlossen, müssen Überlegungen angestellt werden, wie die Struktur dieser Dateien geeignet beschrieben werden kann, um die Erstellung von Parsern und Transformatoren für Schreibzugriffe zu ermöglichen. Gemeinsam ist dabei allen bisher untersuchten Dateien, daß nach dem Auffinden eines bestimmten Datenelements dessen *Ausdehnung* (also sein Anfang und Ende im *byte-stream* der Datei) entweder aufgrund einer festen Länge (satzorientierte Dateien) oder mit Hilfe von Anfangs- und/oder Endtokens (strukturierte Texte) ermittelt werden kann. Das stellt die Aufgabe des *Findens* eines Datenelements (bzw. der Ermittlung der richtigen Position in einer zu schreibenden Datei) an den Anfang des Integrationsprozesses.

Bei der Untersuchung dieses Problems wird deutlich, daß für die Einordnung eines Datenelements in eine Datei zwei grundlegende Konzepte verwendet werden

können, von denen eines oder beide für die Positionierung des Datenelements in der Datei nötig sind:

- *Die Position in der Datei.*
  Die Position einer Bytefolge in einer Datei gibt bei einer Reihe von Dateiformaten hinreichende Auskunft über die Bedeutung des dort gespeicherten Datenelements. Dabei sind drei Fälle zu unterscheiden:

  1. Die *absolute* Position in der Datei ist ausreichend für eine Beschreibung (Dateien mit fester Struktur wie Konfigurationseinstellungen oder Parameterlisten).
  2. Die Bedeutung ergibt sich aus der Position der Bytefolge *modulo* einer Konstanten. Beispiele sind Dateien mit Record-Struktur wie z.B. Tabellen relationaler Datenbanksysteme. Dann heißt die Konstante *Recordlänge*. Es kann hier erforderlich sein, auch die „Zeile“, in der das Datenelement steht, zu identifizieren, was in der Regel über ein spezielles Datenelement in der Zeile (Schlüssel oder Recordnummer) geschieht.
  3. Die Position *relativ* zu einer Anfangsposition ist entscheidend für die Bedeutung. Dieser Fall tritt in komplexeren Dateiformaten mit hierarchischer Struktur auf. Hier werden Bytes zu Gruppen zusammengefaßt, die dann durch geeignete Mittel (z.B. *tokens*) gekennzeichnet werden.

- *Ein einleitendes festes Datenelement (token).*
  Oft können Datenelemente in beliebiger Reihenfolge aufeinander folgen. Typische Vertreter einer solchen Struktur sind Vektorgrafikformate, bei denen die Strukturelemente einer Grafik meist in der Reihenfolge ihrer Erstellung abgespeichert werden. Dann ist eine Ermittlung der Datenbedeutung über die Position des Datenelements allein nicht ausreichend. Vielmehr muß die Bedeutung des Elements durch eine besondere Bytefolge gekennzeichnet werden, die mit einem Parser erkannt werden kann. Dabei ist Wert darauf zu legen, daß die Bytefolge des *token* in den eigentlichen Datenelementen nicht vorkommen kann, da sonst eine eindeutige Abbildung der Datei auf eine Datenbankstruktur nicht möglich ist. Ein Ende eines solchen durch *token* eingeleiteten Datenelements kann durch ein weiteres *token* (auch ein *end-of-line (EOF)* kann ein *token* sein), durch ein neues Anfangs-*token* oder durch die Länge des Datenelements erkannt werden.

Ausgehend von diesen Überlegungen wurde nach einer Möglichkeit gesucht, Dateien allgemein beschreiben zu können, wobei die Verwendung bereits vorhandener und erprobter Beschreibungsmittel als vorteilhaft angesehen wurde. Der Ansatz

kontextfreier bzw. kontextsensitiver Grammatiken erweist sich dabei nur eingeschränkt als geeignet, da eine vollständige Beschreibung recht aufwendig und die Lesbarkeit stark eingeschränkt ist. Daher wird für eine Lösung des beschriebenen Problems der Dateibeschreibung eine Formulierung der Dateistruktur mit Hilfe der Dokumentbeschreibungssprache SGML vorgeschlagen.

3.3.1 Grundlagen von SGML

SGML ist die Abkürzung für *Standard Generalized Markup Language*. Die Sprache wurde 1986 von der ISO definiert [ISO86] und dient zur Beschreibung der Struktur von Dokumenten mit Hilfe einer festgelegten Menge von Sprachelementen. SGML-Dokumente können so als Quelltexte für die automatische Bearbeitung von Texten dienen, sind allerdings auch für Menschen recht gut lesbar, was der Flexibilität von SGML zugute kommt. Die Sprache SGML enstand in dem Bestreben, den Austausch von Texten aus verschiedenen Quellen zu erleichtern und eine Standardisierung früherer Ansätze zur Textformatierung (IBM Script, UNIX `nroff`) vorzunehmen. In den zehn Jahren seines Bestehens hat SGML eine weite Verbreitung erfahren und entwickelt sich zum Standard für die Abfassung von Dokumenten und Texten. So wird es z.B. vom US-Verteidigungsministerium im Rahmen seiner Initiative *CALS (Computer-aided Acquisition and Logistics Support)* zur Erstellung technischer Dokumentationen verwendet. SGML ist als Standard-Markierungssprache für Dokumente in der Lage, Instanzen zur Beschreibung bestimmter Dateitypen festgelegter Struktur zu erzeugen, indem sogenannte Dokumenttypdefinitionen (DTD) aufgestellt werden. Die für das WWW entwickelte Hypertext-Beschreibungssprache HTML stellt eine solche Instanz von SGML dar, kann also mit einem SGML-Parser und der entsprechenden DTD gelesen werden. Eine Einführung in SGML mit Beispielen und Tutorien gibt [vH90].

Aufgrund der starken Standardisierung und hohen Flexibilität von SGML schlägt der Autor die Verwendung von SGML-Dokumenttypdefinitionen zur Strukturbeschreibung von Dateien vor, wobei durch die Angabe zusätzlicher Informationen eine zur Ableitung von Transformatoren ausreichende Beschreibung erfolgen kann.

Diese Vorgehensweise soll im folgenden für die einzelnen Dateitypen aus Kapitel 3.2 beschrieben werden, wobei zunächst eine kurze Vorstellung der für diesen Zweck benötigten Sprachelemente von SGML erfolgen soll.

Der für den angegebenen Zweck benötigte Teil der Sprache SGML sind die **Dokumenttypdefinitionen (DTD)**, die zur Beschreibung der Struktur eines SGML-Dokuments benötigt werden. Ein SGML-Dokument besteht aus einem Text mit *tokens* (ähnlich HTML), wobei die Art und Bedeutung der *tokens* in der DTD festgelegt wird. Ein Beispiel für ein SGML-Dokument mit zugehöriger DTD befindet sich in Anhang C.

Für unsere Zwecke genügt es, sich für die Beschreibung der DTD auf die Syntaxelemente `ELEMENT` für Elementdefinitionen und `ATTLIST` für Attributlisten zu beschränken.

Elementdefinitionen

Definitionen von Elementen haben die Form

```
<!ELEMENT name token-minimierung inhalt >
```

Die *Token-Minimierung* wird hier nicht benötigt, daher ist der Inhalt dieses Feldes leer (`- -`). Der Inhalt eines Elements wird über ein *Inhaltsmodell* beschrieben, das aus weiteren Elementen und dem speziellen Element

`#PCDATA`

bestehen kann, wobei `#PCDATA` **parsed character data** bedeutet. Dies ist die Kennzeichnung für Text bzw. Datenelemente, die auf ein erkanntes *token* folgen, wobei innerhalb eines solchen Textes wiederum andere *tokens* auftreten können. Mit diesen Elementen, den Operatoren aus Tabelle 3.1 und runden Klammern `()` können sogenannte *Modellgruppen* aufgebaut werden, aus denen dann Ausdrücke aufgebaut werden können, die das Inhaltsmodell darstellen.

Binäre Operatoren (für Elemente)		
`,`	Sequenz	genau diese Reihenfolge
`&`	UND	beliebige Reihenfolge
`|`	ODER	exklusives ODER
Unäre Operatoren (für Gruppen)		
`*`	Stern	Auftreten $0 \ldots n$-fach
`?`	Fragezeichen	Auftreten 0 oder 1-fach
`+`	Plus	Auftreten ein- oder mehrmals

Tabelle 3.1: Operatoren in SGML

Ein Beispiel für eine Elementdefinition ist

```
<!ELEMENT Memo      - -  ((To & From), Body, Close?)      >
```

Es wird das Element `Memo` definiert, das aus folgenden Teilen besteht:

1. genau eine Gruppe, die aus den Elementen `To` und `From` besteht, die beide genau einmal vorhanden sind, aber in beliebiger Reihenfolge
2. *danach* genau ein Element des Typs `Body`
3. *danach* 0 oder 1 Elemente des Typs `Close`

Für eine ausführliche Beschreibung des Elementtyps `Memo` s. S. 135.

Mit Hilfe von SGML-Dokumenttypdefinitionen ist lediglich die Beschreibung der groben Struktur von Dateien möglich. Zur Ergänzung um die für Transformatoren nötigen Informationen können **Tabellen** verwendet werden, die die genaue Feinstruktur eines Datenelements je nach Dateityp beschreiben. Für die behandelten Dateitypen werden solche Tabellen in den folgenden Abschnitten vorgeschlagen.

Attributlisten

Attribute bzw. *Attributdeklarationen* werden in SGML verwendet, um spezifische Eigenschaften einzelner Elemente zu definieren. Zur Verdeutlichung soll ein Beispiel angegeben werden:

Mit einem SGML-Element `Text` soll ein fortlaufender Text beschrieben werden. Soll dieser Text gedruckt werden, ist eine linksbündige, zentrierte oder rechtsbündige Anordnung möglich. Diese Information ist für die SGML-DTD irrelevant, spielt aber eine Rolle für einen SGML-Parser, der die SGML-Dokumentinstanz verarbeitet und in einem bestimmten Format ausgibt. Diese Aufgabe führt zur folgenden SGML-DTD:

```
<!ELEMENT Text     - -   (#PCDATA)                          >
<!ATTLIST Text Format    (links, zentriert, rechts)   links  >
```

Dabei ist `Text` der Name des Elements, für das ein Attribut festgelegt wird, `Format` ist der Name des Attributs, dann folgt die Liste der möglichen Werte und darauf der *default*-Wert.

Soll der SGML-Parser einen `Text` nun rechtsbündig formatieren, so ist im SGML-Dokument folgendes anzugeben:

```
<Text Format="rechts">
Hier folgt der Text...
</Text>
```

Der Parser ist für die Auswertung des Attributs `Format` verantwortlich; eine Zuordnung einer Bedeutung kann in SGML nicht erfolgen.

Statt einer festen Wertemenge kann der Typ eines Attributs auch einer von mehreren fest definierten Datentypen sein, von denen die wichtigsten `NUMBER` und `CDATA` sind, wobei letzteres **character data** bedeutet und Daten beschreibt, die keine weiteren SGML-Tags enthalten können (äquivalent zum herkömmlichen Datentyp *string*).

3.3.2 *dBase*-ähnliche Tabellen

Aufgrund der stark vorhandenen Metainformationen in einer solchen Tabelle gestaltet sich eine einheitliche Beschreibung etwas schwierig. Geht man von einer Zeilen- und Spaltenstruktur der Tabelle aus, deren Ausdehnungen und Datentypen im Header der Datei beschrieben werden, so gelangt man zu einer DTD der Form

```
<!ELEMENT Datei        - -  (Header,(Datensatz)+)   >
<!ELEMENT Header       - -  #PCDATA                 >
<!ELEMENT Datensatz    - -  (Attribut)+             >
<!ELEMENT Attribut     - -  #PCDATA                 >
```

Da nicht bekannt ist, wie viele Attribute ein Datensatz enthält, kann eine Unterscheidung der Form

```
<!ELEMENT Datensatz    - - (Att1, Att2, Att3, ...)  >
```

nicht vorgenommen werden. Dies kann nur in einer entsprechenden Tabelle ausgedrückt werden, die ähnlich der SQL-Anweisung `create table` aufgebaut sein könnte. Da in dem hier untersuchten System FACTOTUM solche Dateistrukturen nicht vorkommen, wird dafür keine feste Tabellenstruktur angegeben.

3.3.3 Stark strukturierte Texte

Solche Dateien sind oft recht komplex, lassen sich jedoch mit Hilfe von SGML gut beschreiben. Der Aufbau der DTD orientiert sich an der jeweiligen Dateistruktur und kann daher nicht allgemein wiedergegeben werden. Als Datenelemente (und damit Kandidaten für SGML-Elementdefinitionen) sind dabei Daten zusammen mit den sie einleitenden Schlüsselwörtern anzusehen. Eine Zeile in einer Datei wie

`Startpunkt:␣␣␣␣␣23.4;45.6`

könnte z.B. folgendermaßen modelliert werden:

```
<!ELEMENT Datei      - - (..., Startpkt, ...)          >
<!ELEMENT Startpkt   - - (Stpkt_X-Koor, Stpkt_Y-Koor)  >
```

Das Aussehen der Schlüsselworte kann dann zusammen mit den Datentypen für die Elemente in einer Tabelle definiert werden (Tabelle 3.2 zum obigen Beispiel). Wird eine ganze Gruppe von Datenelementen mit einem Schlüsselwort eingeleitet, so kann dieses ebenfalls in der Tabelle vermerkt werden. Wird nach einem Schlüsselwort in der Datei ein Zwischenraum fester Länge (z.B. eine Anzahl Leerzeichen) gelassen, kann dieser entweder zum Schlüsselwort gerechnet oder in der

Tabelle separat vermerkt werden, um dem Parser ein „Überspringen“ (*skip*) der entsprechenden Stelle zu ermöglichen.

Name	**Anfang**	**Skip**	**Datentyp**	**Ende**
Startpkt	`Startpunkt:`	5		
Stpkt_X-Koor			Real	;
Stpkt_Y-Koor			Real	EOL (*end of line*)

Tabelle 3.2: Beschreibung stark strukturierter Dateien

3.3.4 Hypertexte

Hypertexte können mit den in SGML definierten Sprachelementen beschrieben werden. Dazu ist das SGML-Konstrukt **Attribut** notwendig, das einleitend erläutert wurde (S. 27).

Mit Hilfe von Attributen können alle in Hypertexten vorkommenden Konstrukte erfaßt werden. So können Links z.B. erkannt werden, indem ein `CDATA`-Attribut eingeführt wird. So kann ein SGML-Parser den Zeichenwert dieses Attributs auslesen und anhand dieses Wertes zu einem anderen Dokument verzweigen.

```
<!ELEMENT Link      - -   (#PCDATA)      >
<!ATTLIST Link Ziel   CDATA    #IMPLIED >
```

Der Wert `#IMPLIED` für den *default*-Wert bedeutet, daß bei fehlendem Attributwert im SGML-Dokument ein Standardwert vom Parser bereitgestellt (impliziert) wird.

Insbesondere HTML-Dateien weisen ein Format auf, das sich mit einem herkömmlichen SGML-Parser vollständig lesen läßt. Die zugehörige HTML-Dokumenttyp-Beschreibung (DTD) ist im WWW verfügbar und wird ständig aktualisiert.

Ist die Modellierung von Datentypen notwendig, kann sie wie bei den anderen Dateitypen über entsprechende Tabellen vorgenommen werden. Unter Umständen muß hier auch das genaue Aussehen der Daten für Attributwerte (Ziel von Links, Referenzen usw.) in einer Tabelle angegeben werden. Die Struktur einer solchen Tabelle hängt vom jeweiligen Dateiformat ab.

3.3.5 Satzorientierte Dateien

Satzorientierte Dateien bilden einen recht einfachen Fall für die Modellierung, da sie keine Metainformationen enthalten. Eine DTD für eine zweispaltige Datei hat z.B. die Form

```
<!ELEMENT Datei        - -  (Datensatz)+          >
<!ELEMENT Datensatz    - -  (Elem1, Elem2)        >
<!ELEMENT Elem1        - -  #PCDATA               >
<!ELEMENT Elem2        - -  #PCDATA               >
```

In einer Tabelle werden die genauen Formate der Elemente definiert, wobei die Angabe der Länge und des Datentyps ausreichen (Tabelle 3.3).

Name	Länge	Datentyp
Elem1	8	Integer
Elem2	24	String

Tabelle 3.3: Beschreibung satzorientierter Dateien

Für eine vollständige Beschreibung ist noch die Angabe von Name und Pfad der Datei nötig.

3.3.6 Schwach strukturierte Dateien

Auch für diesen Dateityp gelten die bisher vorgestellten Konzepte. Soll ausgedrückt werden, daß in einem nicht genauer strukturierten Text ein bestimmtes Attribut beliebig oft an beliebiger Stelle auftreten kann, kann ein SGML-Element ähnlich dem folgenden definiert werden:

```
<!ELEMENT Text      - -  (#PCDATA|Attribut)*  >
<!ELEMENT Attribut  - -  #PCDATA              >
```

Auch das genau einmalige Auftreten eines Elements kann definiert werden, z.B. als

```
<!ELEMENT Text      - -  (#PCDATA, Attribut, #PCDATA)  >
```

Dann muß allerdings die *Token-Minimierung* explizit verboten werden und darf beim Parsen des Dokuments nicht aktiviert sein (wenn ein SGML-Parser benutzt wird). Im anderen Fall treten Mehrdeutigkeiten auf (vgl. [vH90]). Da im vorliegenden Fall aber nur die SGML-Modellierung und nicht der Vorgang des *parsing* eine Rolle spielen, kann eine solche Elementdefinition als sinnvoll angesehen werden.

Für die Datentypangabe mit Hilfe von Tabellen ergeben sich keine neuen Erkenntnisse.

3.3.7 Binärdateien

Wie bereits angesprochen, ist der Fall binärer Dateien ein Sonderfall für die Datenmodellierung. Das nicht ohne weiteres lesbare Dateiformat stellt ein wesentliches Hindernis für eine Modellierung solcher Dateien dar. Sollte es allerdings gelingen, die Bedeutung aller oder einiger Datenelemente einer Binärdatei zu entschlüsseln, kann die SGML-Modellierung anhand der bereits vorgestellten Konzepte erfolgen. Meist läßt sich dann eine solche Datei einem der bisher vorgestellten Dateitypen direkt zuordnen.

Als Besonderheit ist zu beachten, daß im SGML-Standard nur bestimmte Zeichen zugelassen sind, so daß ein Dokument, das binäre Daten enthält, zunächst nicht als SGML-Dokument angesehen werden kann. Es existiert jedoch der Datentyp `NDATA`, der Elemente definiert, die vom SGML-Parser nicht gelesen werden. Mit seiner Hilfe können binäre Daten im SGML-Dokument abgelegt werden. Für die Verarbeitung solcher Daten muß ein externer Prozessor aufgerufen werden, der diese Daten liest und nach Ende des Datenstroms die Kontrolle an den SGML-Parser zurückgibt. Dies sind allerdings Implementationsaspekte, die für die Modellierung einer Datei keine große Rolle spielen.

Kapitel 4

Föderierung mit GIM

4.1 Anforderungen an ein Integrationsmodell

Bei der Entwicklung einer föderierten Datenbanklösung steht die Frage nach dem zu verwendenden gemeinsamen Datenmodell (GDM, auch als *canonical data model* bezeichnet). Gewöhnlich wird hierbei davon ausgegangen, daß ein GDM mindestens so ausdrucksstark bzw. semantisch reich sein muß wie das semantisch reichste zu integrierende lokale Datenmodell. Diese Sicht wird u.a. in [SCG91] vertreten. Von diesem Standpunkt aus liegt es nahe, objektorientierte Datenmodelle als GDM zu benutzen, wobei der ODMG-93-Standard zugrundegelegt werden kann [Cat94]. Wie in [Sch95] gezeigt wird, sind objektorientierte Datenmodelle prinzipiell geeignet, mit entsprechenden Erweiterungen als globale Datenmodelle zu dienen. Allerdings bieten solche Datenmodelle keine unabhängige Sicht auf eine Applikation. Es kann zuviel „semantischer Relativismus“[1] im föderierten Schema ausgedrückt werden, so daß ein solches Schema bereits eine anwenderspezifische Sicht auf den modellierten Weltausschnitt enthält. Daher ist die Ableitung anderer Sichten eine sehr schwierige Aufgabe, die nur mit hohem Aufwand zu lösen ist. Außerdem verursacht die Verwendung eines semantisch reichen Datenmodells einen höheren Grad an Heterogenität auf der Schemaebene, was den Integrationsprozeß weiter erschwert.

Es ist jedoch möglich, für das föderierte Schema ein semantisch armes Datenmodell zu benutzen. Die dabei entstehenden Probleme können relativ leicht abgefangen werden, und ein solches Datenmodell kann alle Anforderungen an ein GDM erfüllen.

[1] Der Begriff *semantischer Relativismus* wird benutzt, um eine spezifische Anwendersicht auf einen Weltausschnitt zu beschreiben. Beispiele für solche sichtspezifischen Informationen in objektorientierten Modellen sind z.B. Typkonstruktoren, Vererbungshierarchien, Methoden von Klassen oder abgeleitete Attribute.

Um diese Aussage zu motivieren, soll ein kurzer Überblick über die Anforderungen an ein integriertes Schema gegeben werden (nach [BLN86]):

- *Vollständigkeit und Korrektheit.* Das integrierte Schema muß alle Konzepte enthalten, die in den Komponentenschemata vorkommen. Es muß in der Lage sein, die Vereinigungsmenge der lokalen Schemata abzubilden.

- *Minimalität.* Ein Konzept, das in mehreren Komponentenschemata vorkommt, darf im integrierten Schema nur einmal vorhanden sein.

- *Verständlichkeit.* Das integrierte Schema sollte für Datenbankentwickler und -anwender leicht zu verstehen und anzuwenden sein. Das bedeutet, daß unter mehreren möglichen Repräsentationen eines Integrationsergebnisses in einem bestimmten Datenmodell die qualitativ verständlichste ausgewählt werden muß.

Anforderungen an eine Föderierungsmethodik wurden in [Sch95] abgeleitet und sollen hier kurz wiedergegeben werden.

1. Das FDBS muß die Integration sehr verschiedener lokaler DBS unterstützen. Es müssen totale und umkehrbare Schematransformationen existieren, die für die Integration der lokalen Schemata mit dem integrierten Schema sorgen.

2. *Schemaintegration und -transformation.*

   (a) Die Integration muß adäquat und korrekt ausgeführt werden. Jedes Element eines Komponentenschemas muß eine Entsprechung im föderierten Schema haben. Die Existenz lokaler Schemata muß dem Benutzer der Föderierung verborgen sein.

   (b) Die Integration ist ein dynamischer Prozeß, da Änderungen lokaler Schemata und das Hinzukommen neuer Komponenten möglich sind.

   (c) Der Integrationsprozeß erfordert einen gewissen Anteil an Handarbeit. Eine wenigstens methodische Unterstützung, idealerweise eine teilweise Automatisierung mittels geeigneter Werkzeuge, kann die Fehlerhäufigkeit verringern.

   (d) Eine ausreichende *performance* ist wichtig für die Akzeptanz des Systems in der Praxis. Hier ist darauf zu achten, daß nicht durch eine große Anzahl von Schemaebenen zwischen globaler Applikation und lokaler Datenbank starke Geschwindigkeitsverluste auftreten.

3. *Schematransformationen in externe Schemata.*

   (a) Eine Datenschematransformation vom föderierten in ein externes Schema (das auf einem beliebigen Datenmodell aufbauen kann), muß funktional und unabhängig von den Datenmodellen der lokalen Schemata sein.

   (b) Globale Applikationen benötigen spezifische externe Schemata, die auf ihre speziellen Bedürfnisse zugeschnitten sind. In diesem Zusammenhang steht die Forderung nach logischer Datenunabhängigkeit. Dabei sind zwei Fälle möglich. Zum einen muß es das FDBS ermöglichen, zu einem gegebenen lokalen Schema ein äquivalentes globales Schema zu erzeugen, ein Aspekt, der oft vernachlässigt wird. Andererseits benötigen neue Applikationen spezifische, integrierte und konsistente Sichten auf die föderierten Daten. Auch die Integritätsbedingungen müssen konsistent bleiben, und es müssen wiederum totale und reversible Datenschematransformationen existieren.

Bei der Benutzung eines semantisch armen Datenmodells tritt das Problem auf, daß die Menge der angebotenen Modellierungskonzepte nicht ausreicht, um externe Schemata abzuleiten. Das wird oft als Grund für die Wahl eines semantisch reichen Modells angegeben. Allerdings kann dieses Problem umgangen werden, indem für die externen Schemata wiederum ein semantisch reiches Datenmodell, wie z.B. ein erweitertes OO-Modell, benutzt wird.[2] Dann ist es nicht erforderlich, das Integrationsschema zu *ändern*, es müssen lediglich die in diesem Schema nicht enthaltenen Informationen *hinzugefügt* werden. Das stellt einen wesentlichen Vorteil des semantisch armen Ansatzes dar. Auch kann so eine Anpassung an unterschiedliche und veränderliche Komponentenschemata wesentlich leichter durchgeführt werden, und die Komplexität des Homogenisierungs- und Integrationsprozesses kann entscheidend reduziert werden.

Im nächsten Kapitel soll die Erfüllbarkeit der oben genannten Aufgaben durch das Generische Integrationsmodell (GIM) dargestellt werden.

4.2 Föderierung mit dem Generischen Integrationsmodell

Wie sich aus den vorhergehenden Ausführungen ergibt, muß das Datenmodell des föderierten Schemas ein relativ einfaches Datenmodell sein, das möglichst wenige Konzepte zur Abbildung sichtspezifischer Informationen enthält. Diese Konzepte

[2] Weitere Motivationen für z.B. das objektorientierte Modell als kanonisches Datenmodell finden sich in [Här94].

sollten orthogonal sein, da sich gegenseitige Abhängigkeiten der Konzepte störend auf die Qualität des Integrationsschemas auswirken. Ein Datenmodell, das diese Anforderungen erfüllt, ist das **Generische Integrationsmodell** (*Generic Integration Model, GIM*) [Sch95, CHJ+96, SS96a, SS96b, SS96c].

Wird das föderierte Schema eines FDBS in GIM modelliert, sollte das föderierte Schema nicht gleichzeitig das Interface zu globalen Applikationen sein, sondern es sollten externe Schemata in einem Datenmodell, das mehr semantische Relativismusinformationen ausdrücken kann, definiert werden. Die Föderierung eines Datenbanksystems mit GIM als kanonischem Datenmodell geht in drei Schritten vor sich:

1. Bei der Schematransformation eines lokalen Schemas in ein Komponentenschema findet eine semantische Reduktion statt. Die dabei verlorengehenden Informationen werden in einem speziellen Semantischen Relativismus-Schema (SR-Schema) gerettet.

2. Jetzt kann die eigentliche Integration schneller und leichter geschehen als es z.B. mit einem objekt-orientierten Datenmodell möglich wäre.

3. Nach dem Integrationsprozeß kann das integrierte Schema wieder mit semantischen Relativismus-Informationen angereichert werden, um externe Schemata zu definieren.

Im folgenden soll gezeigt werden, warum die Föderierung mit GIM eine für den föderierten Datenbankentwurf geeignete Verfahrensweise darstellt.

4.2.1 Eignung eines semantisch armen Datenmodells

Anhand der in Kapitel 4.1 aufgestellten Anforderungen an eine Föderierungsmethodik soll hier gezeigt werden, daß GIM für einen Einsatz in FDBS geeignet ist (nach [Sch95]).

1. Die Anpassung an verschiedene lokale Datenmodelle geschieht durch Aufspaltung des lokalen Schemas in Informationen für das Komponentenschema und semantische Relativismusinformationen. Dabei ist darauf zu achten, daß lokale Integritätsbedingungen keinen semantischen Relativismus darstellen und somit in das Komponentenschema überführt werden müssen.
   Es ist hier erforderlich, inverse und totale Schematransformationen für alle Elemente der Komponentenschemata und der zugehörigen SR-Schemata zu finden.

2. *Schemaintegration und -transformation.*

   (a) Durch die Normalisierung[3] der Komponentenschemata treten bei der Integration relativ wenige Konflikte auf. Das macht den Integrationsprozeß einfacher und somit weniger fehleranfällig, da z.B. auch weniger Typkonflikte auftreten.

   (b) Der Integrationsprozeß vereinfacht sich und unterstützt so die flexible Anpassung des integrierten Schemas an sich ändernde Komponenten.

   (c) Aufgrund der geringen Anzahl möglicher Konflikte und der recht gut zu standardisierenden Integrationsmethodik ist die Entwicklung von den Integrationsprozeß unterstützenden Tools möglich. Dazu finden am ITI z.Z. Untersuchungen statt.

   (d) Da die Ableitung gesonderter externer Schemata bei der Benutzung von GIM notwendig ist, sind bis zu 5 Schemaebenen notwendig, was zu *performance*-Verlusten führen kann. Dieser Nachteil wird jedoch durch die erwähnten Vorteile einer Nutzung von GIM aufgewogen, zumal die 5-Ebenen-Architektur einen verbreiteten Ansatz zu föderierten Datenbanken darstellt.

3. *Schematransformationen in externe Schemata.*

   (a) Datenschematransformationen vom integrierten in ein externes Schema sind einfach und aufgrund der wiederverwendbaren semantischen Relativismusinformationen auch vollständig. Zusätzlich können im externen Schema Informationen aufgenommen werden, die nicht im Datenmodell der Komponente, sondern in der zugehörigen Applikation verborgen sind. Diese können in Zusammenarbeit mit einem Experten für die Komponente gewonnen werden.

   (b) Die Vorgehensweise bei der Erzeugung externer Schemata ist auch bei der Verwendung von GIM abhängig vom verwendeten Datenmodell. Der Vorteil von GIM ist aber, daß semantische Relativismusinformationen nur hinzugefügt, nicht geändert werden müssen, um im externen Schema berücksichtigt zu werden. In jedem Fall ist dieser Prozeß nicht aufwendiger als die entsprechenden Schritte bei der Verwendung objektorientierter Datenmodelle als kanonische Datenmodelle.

Zusammenfassend läßt sich sagen, daß die Verwendung eines semantisch armen Datenmodells allgemein den Prozeß der Integration und Datenschematransformation in externe Schemata vereinfacht. Insbesondere erhöht sich die Flexibilität von Schemaintegration und Schematransformationen.

[3] Befreiung vom semantischen Relativismus

Der Autor geht davon aus, daß die Verwendung eines semantisch armen Datenmodells, insbesondere des in diesem Kapitel beschriebenen GIM, für die Integration heterogener Datenbanksysteme vorteilhaft ist. Das gilt natürlich auch für den in dieser Arbeit behandelten Spezialfall dateibasierter Datenhaltungssysteme. Durch die bereits geleistete Arbeit zur Entwicklung eines Algorithmus zur Datenbankintegration für föderierte Datenbanksysteme bestehen gute Voraussetzungen für den Einsatz von GIM in der vorliegenden Arbeit. Gleichzeitig stellt diese Arbeit den Versuch dar, die Methodik der Integration mit GIM an einem realen Beispiel einzusetzen und ihre Eignung zu überprüfen.

4.2.2 Eigenschaften von GIM

Zur Zeit existieren noch keine formalen Syntax- und Semantikbeschreibungen von GIM. Daher soll nur kurz auf die Grundkonzepte von GIM eingegangen werden. Im folgenden soll dann kurz beschrieben werden, wie Schematransformation und -integration in GIM prinzipiell möglich ist. Am Beispiel des Produktionsplanungssystems FACTOTUM (s. Kapitel 5.1) kann dann eine FDBS-Entwicklung mit GIM verfolgt werden.

Folgende Konzepte sind die Grundlagen von GIM:

1. *Einfache Datentypen.* Komplexe Datenstrukturen drücken semantischen Relativismus aus. (So könnten Realweltobjekte, die als komplexe Attribute modelliert wurden, durchaus auch als eigene Objekte gesehen werden, was zwei unterschiedliche Benutzersichten darstellt.) Daher unterstützt GIM nur einfache Datentypen, wie sie in relationalen Datenmodellen als Relationen in erster Normalform vorkommen (keine mengenwertigen Attribute). Abgeleitete Attribute werden ebenfalls nicht unterstützt.

2. *Flexible Integritätsbedingungen.* Da Integritätsbedingungen im integrierten Schema modelliert werden müssen, kann die Behandlung von Konflikten dieser Bedingungen im Integrationsprozeß nicht vermieden werden. GIM unterstützt Integritätsbedingungen ähnlich SQL2.

3. *Objekt-Identifikatoren (OID).* OID unterstützen die Referenzierung von Objekten. Sie können z.B. Schlüsselkonflikte in relationalen Systemen verhindern.

4. *Bidirektionale Referenzen.* Alle Referenzen zwischen Objekten müssen in binäre Beziehungen aufgelöst werden. Dabei werden sowohl 1:1- als auch 1:n- und m:n-Beziehungen unterstützt. Es müssen allerdings immer inverse Referenzen existieren, da die Richtung einer Referenz als semantische Relativismusinformation angesehen wird.

5. *Nicht-überlappende Extensionen.* Zwei Klassen müssen entweder nur identische Instanzen oder keine gemeinsamen Instanzen besitzen. Beziehungen wie Untermenge oder teilweise Überlappung sind verboten, da Untermengenhierarchien semantischen Relativismus ausdrücken.

6. Diese Konzepte müssen so *orthogonal* wie möglich implementiert werden. Die Vermeidung gegenseitiger Abhängigkeiten der Konzepte vermeidet Abhängigkeiten zwischen verschiedenen Konflikten und zwischen Algorithmen zur Konfliktauflösung.

Die ersten beiden dieser Konzepte sind aus der Welt relationaler Datenbanken übernommen. Insgesamt steht das GIM dem relationalen Ansatz näher als dem objekt-orientierten. Es sind allerdings mit OID und Referenzen auch Elemente aus objekt-orientierten Modellen übernommen worden. Die Forderung nach nicht überlappenden Extensionen ist eine spezifische Eigenschaft des GIM. Sie steht in Widerspruch zum Konzept der Spezialisierung im objekt-orientierten Ansatz, so daß z.B. dieser Spezialfall gesondert behandelt werden muß.

Syntaxelemente von GIM

GIM-Schemata werden in Diagrammen dargestellt. Aus Abbildung 4.1 lassen sich die verwendeten syntaktischen Elemente erkennen. Die Grafik stellt ein einfaches Datenbankschema mit zwei Klassen dar, die durch eine bidirektionale Referenz miteinander in Beziehung stehen. Solche bidirektionalen Referenzen (umkehrbar eindeutige Abbildungen) bilden das einzige in GIM enthaltene Konzept zur Darstellung von Beziehungen zwischen Klassen. Andere Beziehungskonzepte müssen entsprechend transformiert werden. In der Klasse E1 ist eine Unterbrechung des gezeichneten Rahmens zu erkennen. Damit wird ausgedrückt, daß das zugehörige Attribut nicht Element der Klasse ist. Diese zeichnerische Besonderheit ist bei der Modellierung komplizierterer Modelle notwendig, wie sich bei der Ableitung der Komponentenschemata zeigen wird.

Ein wichtiger Bestandteil eines GIM-Schemas sind die Integritätsbedingungen. Ein Vorschlag für die Formulierung dieser Bedingungen (*integrity constraints*) wird in [CHJ+96] gemacht.

Danach wird eine **Integritätsbedingung** notiert in der Form:

<schema><nr>$_{<xIC>}$ [<atts>][<exts>]:< bedingung>(<eigenschaften>)

Dabei bezeichnet $<schema>$ den Namen des Datenbankschemas, zu dem die Integritätsbedingung gehört. Innerhalb eines Schemas werden Integritätsbedingungen mittels $<nr>$ numeriert. $<xIC>$ ist eine dreibuchstabige Kennung für die Zugehörigkeit zu einem bestimmten Schritt im FDBS-Design und kann einen der folgenden Werte annehmen:

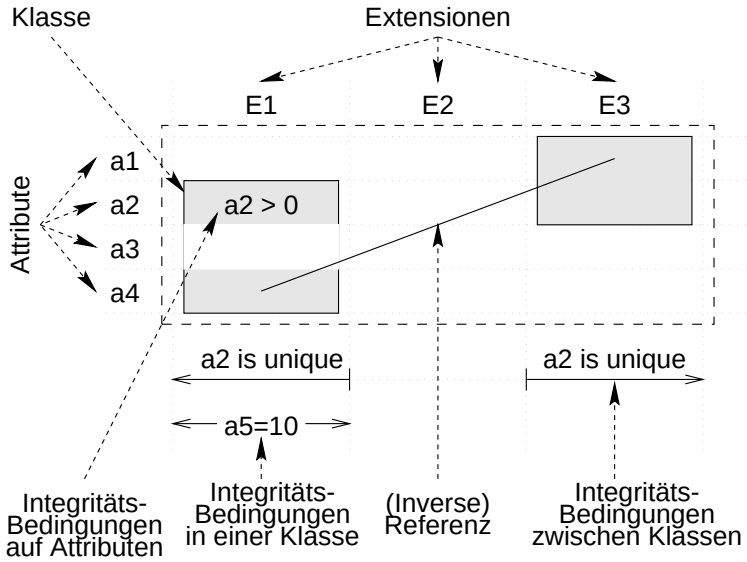

Abbildung 4.1: Syntaktische Elemente eines GIM-Diagramms

- *Lokale Integritätsbedingungen (LIC)* sind die existierenden Bedingungen der lokalen DBS in ihren Datenmodellen.
- *Transformierte Integritätsbedingungen (TIC)* sind die lokalen Integritätsbedingungen, im gemeinsamen Datenmodell ausgedrückt. Dabei können Bedingungen explizit sichtbar werden, die im lokalen Datenmodell noch implizit vorhanden waren.
- *Export-Integritätsbedingungen (EIC)* sind Bedingungen, die auf den Exportschemata definiert werden und den globalen Datenzugriff einschränken.
- *Integrations-Integritätsbedingungen (IIC)* sind auf dem integrierten Schema definiert und leiten sich aus den TIC und EIC ab.
- *Globale Integritätsbedingungen (GIC)* sind zusätzliche Bedingungen, die explizit auf dem integrierten Schema definiert werden.
- *Externe Integritätsbedingungen (XIC)* sind Bedingungen, die explizit auf den externen Schemata definiert werden, um den Zugriff zur föderierten Datenbank zu regeln.

Die Menge von Attributen und die Menge von Extensionen, auf die sich eine Integritätsbedingung bezieht, wird in <*atts*> bzw. <*exts*> definiert. <*bedingung*> enthält die eigentliche Bedingung in mehr oder weniger verbaler Form. Hierfür ist noch keine formale Beschreibung definiert. In <*eigenschaften*> werden spezielle Eigenschaften der Integritätsbedingung aufgeführt, z.B. die Unterteilung in *Intra-Class*- und *Inter-Class*-Integritätsbedingungen, die Unterscheidung von

statischen und *dynamischen* Bedingungen, oder die Klassenzugehörigkeit. Für die Definition der Klassen von Integritätsbedingungen siehe [CHJ+96]. Eine Kurzbeschreibung der verwendeten Klassen wird in Tabelle 4.1 gegeben.

		Extensionale Dimension	
		1 Objekt	m Objekte ($m > 1$)
Intensionale Dimension	0 Attribute	IC_{10}	IC_{m0}
	1 Attribut	IC_{11}	IC_{m1}
	n Attribute ($n > 1$)	IC_{1n}	IC_{mn}

Tabelle 4.1: Klassen von Integritätsbedingungen

4.2.3 Schematransformation in GIM

In einem der wichtigsten Integrationsschritte bei der Implementation eines föderierten Datenbanksystems geschieht die Transformation der lokalen Datenmodelle in das gemeinsame Datenmodell GIM. Um das zu erreichen, müssen getrennt für jedes Schema Umstrukturierungen und Normalisierungen durchgeführt werden. Im Ergebnis entstehen GIM-Schemata als Komponentenschemata für jede Komponente. Bei der Transformation in das Datenmodell GIM werden auch die Integritätsbedingungen transformiert. Dabei entsteht zu jeder im lokalen Schema enthaltenen Integritätsbedingung eine Bedingung im Komponentenschema. Umgekehrt hat jede Bedingung im Komponentenschema eine (evtl. im lokalen Datenmodell implizite) Entsprechung im lokalen Schema.

Aus Platzgründen wird der Transformationsprozeß nur kurz beschrieben. Für die Einführung an Beispielen siehe [Sch95] und [CHJ+96].

Transformation in einfache Datentypen

Analog zum Vorgehen im relationalen Datenbankentwurf muß die erste Normalform für die Attribute hergestellt werden. Mengenwertige Attribute müssen also in eigene Klassen umgewandelt werden. Um diese Klassen referenzieren zu können, werden bidirektionale Referenzen aufgebaut, die einen Zugriff auf die jeweils andere Klasse möglich machen.

In diesem Schritt kann es vorkommen, daß im lokalen Datenmodell implizit enthaltene Integritätsbedingungen explizit formuliert werden müssen. Insbesondere entsteht beim Anlegen einer neuen Klasse die Bedingung, daß ein Objekt dieser Klasse nur dann existieren kann, wenn es eine Referenz zu mindestens einem Objekt der anderen Klasse aufweist. Beispielhaft könnte eine solche Integritätsbedingung folgendermaßen aussehen:

A1$_{TIC}$[ref_att][NeueKlasse]: card(ref_att)>0 (IC_{11}, intra-class, static)

Außerdem werden durch die Transformation in ein anderes Datenmodell implizite Forderungen des Ausgangsmodells sichtbar. Als Beispiel sei hier der Fall des Schlüsselattributs im relationalen Modell angegeben. Ein Wert eines Schlüsselattributs muß eindeutig (*unique*) sein, und darf je nach Semantik einen bestimmten Wert (meist 0) nicht annehmen.

Bei unterschiedlichen Repräsentationen des gleichen Realweltobjekts in verschiedenen lokalen Schemata können außerdem Datentypkonvertierungen nötig werden. Das können z.B. Restriktionen für die Länge eines Strings oder für die Genauigkeit einer `real`-Zahl sein. Dabei treten Probleme auf, wenn keine isomorphe Abbildung zwischen den Werten im lokalen Datentyp und den möglichen Werten im GIM-Datentyp existiert. Diese Probleme müssen mit zusätzlichen Integritätsbedingungen abgefangen werden.

Nicht überlappende Klassenextensionen

Eine sehr strenge Forderung des GIM ist die Forderung nach Disjunktheit der Klassenextensionen. Speziell in OO-Modellen (Spezialisierung), aber auch in relationale Ansätzen wird von extensionaler Überlappung Gebrauch gemacht. Um die Überlappungen zu beseitigen, ist eine extensionale Dekomposition der Klassen nötig. Das geschieht in der Form, daß aus den Klassen A und B die neuen Klassen $A \backslash B$, $B \backslash A$ und $A \cap B$ erzeugt werden. Die Mengenoperationen $\backslash$ und $\cap$ bedeuten dabei „Mengen-Minus“ und Durchschnitt. Diese drei Klassen enthalten keine gemeinsamen Elemente. Die Intensionen der ersten beiden Klassen sind gleich den Intensionen der jeweiligen Ausgangsklasse, während die Intension der Klasse $A \cap B$ die Vereinigung der Attributmengen der Klassen A und B darstellt.

In diesem Transformationsschritt müssen Integritätsbedingungen, die für eine der an der Extensionsüberlappung beteiligten Klassen galten, entsprechend für die neuen Klassen angepaßt werden.

Objekt-Identifikatoren und Referenzen

GIM unterstützt das Konzept der Objekt-Identifikatoren. Einige Datenbanksysteme enthalten dieses Konzept nicht und müssen darum gesondert behandelt werden. Betrachtungen dazu finden sich in [SS95].

Integritätsbedingungen

Explizit formulierte Integritätsbedingungen müssen an das neue Datenmodell angepaßt werden. Beispiele dafür wurden in den vorhergehenden Abschnitten gegeben. Weitergehende Untersuchungen zu einer geeigneten Behandlung von Integritätsbedingungen werden z.Z. am ITI angestellt (vgl. [Jan96]).

4.2.4 Schemaintegration in GIM

Nachdem alle lokalen Schemata in das gemeinsame Datenmodell überführt sind, ist die Datenmodellheterogenität überwunden. Der nächste Schritt ist jetzt die Überwindung der *Schemaheterogenität*. Diese Integration ist eine sehr komplexe Aufgabe, da eine große Zahl von Konflikten auftreten kann. Aus diesem Grunde kann eine Schemaintegration nicht vollständig automatisch durchgeführt werden. Es ist aber möglich, eine Methodik anzugeben, die die Schemaintegration vereinfacht, indem sie häufig auftretende Konflikte berücksichtigt und geeignete Algorithmen für ihre Beseitigung anbietet.

Die vier wichtigsten Konflikte sollen kurz dargestellt werden.

Metakonflikte

Ein Metakonflikt tritt auf, wenn sich Objekte einer Klasse in *einem* Datenbankschema im Wert eines bestimmten Attributes unterscheiden, während die semantisch gleiche Unterscheidung in einem zweiten Datenbankschema durch die Zugehörigkeit der Objekte zu verschiedenen Klassen gekennzeichnet ist (vgl. Kapitel 2.2). Um einen Metakonflikt zu lösen, spaltet man die Klasse im ersten Schema, in der die Unterscheidung durch Attribute dargestellt wird, in mehrere Teilklassen, die entsprechend der Klassenzuordnung des zweiten Schemas gebildet werden. Um die Zugehörigkeit bestimmter Objekte zu einer bestimmten Klasse zu kennzeichnen, werden Integritätsbedingungen eingeführt. Auch muß bei der Klassenaufspaltung die Änderung der entsprechenden Integritätsbedingungen berücksichtigt werden.

Es sind weitere Formen von Metakonflikten denkbar, so z.B. solche, in denen unterschiedliche Werte eines bestimmten Attributs in einem Schema die Zuordnung von Attributen in Schema A zu Attributen in Schema B bestimmen.

Auflösung von Attributkonflikten

Nach der Auflösung der Metakonflikte liegen sowohl die Intensionen als auch die Extensionen der Klassen in den Komponentenschemata fest. Um mehrere Komponentenschemata zu einem integrierten Schema zusammenzufügen, müssen jetzt Attribute in den beteiligten Schemata identifiziert werden, die gleichartige Aussagen über die gleichen Realweltobjekte treffen. Attributkonflikte treten nun auf, wenn zwei semantisch gleiche Attribute inkompatible Datentypen haben. So tritt z.B. häufig der Fall unterschiedlicher Länge eines Strings auf, was eine isomorphe Abbildung zwischen den Attributwerten unmöglich macht.

Um solchen Konflikten zu begegnen, können Attribute geteilt werden (z.B. kann das Attribut, das den längeren String abspeichert, in zwei kürzere geteilt werden,

von denen eines genau die Länge des Strings im anderen Attribut hat), oder es können zusätzliche Integritätsbedingungen eingeführt werden.

Auflösung intensionaler Konflikte

Intensionale Überlappung behindert die Erstellung externer Schemata. Sie kann jedoch recht einfach durch Aufteilung der Attribute (*intensionale Dekomposition*) beseitigt werden. Dabei werden die Attribute so gruppiert, daß alle Attribute einer Gruppe Aussagen über die gleiche Menge von Klassen machen. Das führt zu einer Äquivalenzrelation über der Menge der Attribute. Intensionale Dekomposition wird im GIM-Diagramm mittels horizontaler Linien dargestellt.

Auf Integritätsbedingungen hat die Dekomposition keinen Einfluß.

Auflösung extensionaler Konflikte

Die Auflösung extensionaler Konflikte geschieht wie im Abschnitt „Nicht überlappende Klassenextensionen“ beschrieben unter Bildung der Mengen $A \backslash B$, $B \backslash A$ und $A \cap B$ aus den Objekten der Klassen A und B. Dabei kommt es zu analogen Änderungen von Integritätsbedingungen.

Probleme treten auf, da die Klasse, die die Durchschnittsmenge der Objekte der Klassen A und B enthält, Redundanz verbirgt. Objekte dieser Klasse werden mehrfach in der Föderation abgespeichert und können daher zu Inkonsistenzen führen. Deshalb müssen Festlegungen zur Behandlung inkonsistenter Objekte getroffen werden. Beispielsweise könnten Änderungen solcher Objekte in einer lokalen Datenbank zu Updates in den anderen Datenbanken führen. Das ist allerdings nur möglich, wenn die Autonomiefestlegungen für die beteiligten Komponenten-DBS ein solches Update zulassen.

Globale Integritätsbedingungen

Bei der Integration mehrerer Komponentendatenbanken treten Effekte auf, die sich nicht durch einzelnes Betrachten der Komponenten erkennen lassen. So könnte z.B. für *uniqueness*-Bedingungen, die in mehreren Klassen jeweils als *intra-class*-Bedingung definiert sind, auch *uniqueness* über die gesamte Menge von Objekten gefordert sein. Das macht die Einführung globaler Integritätsbedingungen notwendig, die solch zusätzliche Schemainformationen modellieren können. Globale Integritätsbedingungen führen allerdings zu Updateproblemen, da im Regelfall nicht entschieden werden kann, was bei einem lokalen Update, der im Geltungsbereich der Integritätsbedingung stattfindet, in anderen Teilen der Föderation geschehen soll.

4.2.5 Ableitung externer Schemata

Externe Schemata sind ein wichtiger Schritt im föderierten Datenbankdesign. Sie dienen zur Erzeugung nutzerspezifischer Sichten auf die Föderierung. Da sich GIM, wie bereits ausgeführt, nicht für die Ableitung externer Schemata eignet, müssen andere, semantisch reiche Modelle benutzt werden.

Es sind drei Arten externer Schemata definiert, die unterschiedliche Transformationsschritte zu ihrer Ableitung erfordern:

1. Es müssen externe Schemata definiert werden können, die zu den lokalen Schemata äquivalent sind. Das ist notwendig, um eine Migration lokaler Applikationen an die Spitze des föderierten Datenbanksystems vornehmen zu können. Für diese Schemata ist es notwendig, die Transformations- und Integrationsschritte zur Ableitung des integrierten Schemas in umgekehrter Reihenfolge auszuführen.

2. Es muß ein externes Schema ableitbar sein, das eine komplette Sicht auf die gesamte föderierte Datenbank bietet. Dazu muß ein externes Schema erstellt werden, das alle Klassen des integrierten Schemas enthält. Da GIM nicht als externes Datenmodell eingesetzt werden sollte, sind Transformationen in das jeweilige externe Datenmodell notwendig.

3. Es kann erfolrderlich sein, eingeschränkte Sichten auf das föderierte Schema bereitzustellen. Dazu können bei der Ableitung Schemaelemente weggelassen und neue Integritätsbedingungen eingeführt werden. Dieser Prozeß ähnelt der Erstellung von Sichten *views* in relationalen Datenbanken.

Um aus einem umfangreichen GIM-Schema lesbare und übersichtliche externe Schemata herzustellen, können Klassen des integrierten Schemas zusammengefaßt werden, ohne daß dabei Informationen verlorengehen. Dabei ist intensionale und extensionale Komposition möglich. Während die intensionale Komposition von Klassen mit gleichen Extensionen keine Probleme bereitet, treten bei der extensionalen Komposition Schwierigkeiten bei der Ableitung der neuen Integritätsbedingungen auf (s. [CHJ+96]). Dann ist zu entscheiden, ob eine Zusammenfassung der Klassen wirklich nötig ist.

Mit diesen Ausführungen soll die Einführung in GIM abgeschlossen werden. Weiterführende Informationen finden sich in der in diesem Kapitel referenzierten Literatur. Nach Meinung des Autors stellt GIM ein Integrationsverfahren dar, das für den Entwurf föderierter Datenbanken geeignet ist, wobei bereits hinreichende Konzepte und Algorithmen für eine Nutzung in der Praxis vorliegen. Weitere Verfeinerungen der Algorithmen und die Schaffung von Werkzeugen zur Vereinfachung von Integrationsschritten erscheinen dabei wünschenswert. In den nächsten

Kapiteln soll anhand der GIM-Methodik eine föderierte Datenbank über dem System FACTOTUM entwickelt werden. Dabei kann eine praktische Erprobung der GIM-Konzepte erfolgen.

Kapitel 5

Automatisierung in der Fabrikplanung

5.1 Das System FACTOTUM zur computerunterstützten Produktionssystemplanung

Durch die immer schneller fortschreitende technische Entwicklung und verstärkte internationale Konkurrenz entsteht in der Industrie der Zwang nach ständiger Neu- und Weiterentwicklung von Produkten, also nach einer großen Anzahl von Varianten und dementsprechend einer häufigen Umstellung von Produktionsprozessen. Das führt zu vielfachen Neu- und Änderungsplanungen im Bereich des Fabrikbetriebs. Das erfordert die Entwicklung von Hilfsmitteln zur Unterstützung der Produktionssystems- bzw. Fabrikplanung, insbesondere auf dem Gebiet der elektronischen Rechentechnik.

Zahlreiche Produkte zur computerunterstützten Fabrikplanung werden auf dem Markt angeboten. All diese Systeme lösen aber lediglich Einzelprobleme des Planungsprozesses. So müssen für eine vollständige Planung mehrere verschiedene Werkzeuge verschiedener Hersteller eingesetzt werden, die meist keine kompatiblen Datenschnittstellen anbieten. Das führt zu Problemen bei der Weiterverwendung von Ergebnissen eines Werkzeugs mit einem anderen. Oft sind die mit *einem* System, z.B. zur Layoutplanung, gewonnenen Daten bei der Weiterarbeit mit einem anderen System, z.B. einem Simulationstool, manuell neu einzugeben. Die dabei zu übertragenden Datenmengen sind erheblich, in diesem Beispiel etwa die Positionen und Größen aller Maschinen, die Wege, die Produkte während ihrer Bearbeitung in der Fabrik nehmen, die Kapazitäten und Geschwindigkeiten der Transportmittel, Bearbeitungs- und Rüstzeiten usw.

Das Institut für Arbeitswissenschaft, Fabrikautomatisierung und Fabrikbetrieb der Otto-von-Guericke-Universität Magdeburg hat als Lösungsvorschlag für die-

ses Problem den *Arbeitsplatz „Computerunterstützte Produktionssystemplanung“ FACTOTUM*[1] vorgestellt, der mehrere vorhandene Einzellösungen zur Fabrikplanung zusammenfaßt und einen Weg zur Durchführung einer vollständigen Planung aufzeigt [FS94]. Der Anspruch des FACTOTUM ist es, eine „Konzeption einer sinnvoll abgestimmten Branchenlösung für die Unterstützung eines systematischen ganzheitlichen Prozesses zur Planung logistischer Ketten im Industriebetrieb auf der Basis kommerzieller austauschbarer Softwarebausteine“[2] zu sein. Der Arbeitsplatz FACTOTUM umfaßt in der derzeitigen Version folgende Softwareprodukte verschiedener Hersteller:

- *FAST/pro* (Fertigungsablauf-, Analyse- und Statistik-Softwarepaket).
  Die Bedeutung dieses Systems für FACTOTUM besteht in seiner Funktion als Datenbanksystem für die Bereitstellung von Ausgangsdaten für die Produktionssystemplanung.

- *LAPLAS* (Layoutplanungssystem).
  LAPLAS ist in der Lage, aus Ausgangsdaten über den Produktionsprozeß die optimale Lage der verwendeten Maschinen in einer Fabrik nach verschiedenen Verfahren zu bestimmen.

- *DUMAS* (Dimensionierung der Materialflüsse).
  DUMAS ist eine Ergänzung zu LAPLAS. Nachdem die Position der Maschinen festliegt, berechnet es die Auslastung der verwendeten Transportmittel und kann eine Anzahl statistischer Aussagen über den Produktionsprozeß treffen.

- *TAYLOR*
  TAYLOR ist ein graphisches Simulationssystem, das speziell auf die Simulation von Fabriken ausgerichtet ist. Es simuliert den Transport und die Bearbeitung von Produkten und unterstützt so den Planer, Schwierigkeiten im Produktionsprozeß wie Staus oder lange Wartezeiten aufzudecken.

- *AutoCAD*
  Das CAD-System AutoCAD dient in FACTOTUM zum Erstellen von Maschinen- und Werkhallengrundrissen sowie nach Beendigung der Planung mit LAPLAS, DUMAS und TAYLOR zur Feinplanung und zur anschaulichen Darstellung der gewonnenen Erkenntnisse. Hierzu wird auch die AutoCAD-Erweiterung ARE-24 genutzt.

Eine Übersicht über die Zusammenarbeit und die Einsatzbereiche der Komponenten des Produktionsplanungssystems FACTOTUM gibt Abb. 5.1.

[1] Factory Tool and Tuning Master

[2] aus: *Kurzbeschreibung des Exponates des IAF der Otto-von-Guericke-Universität Magdeburg zur Hannovermesse 1995*

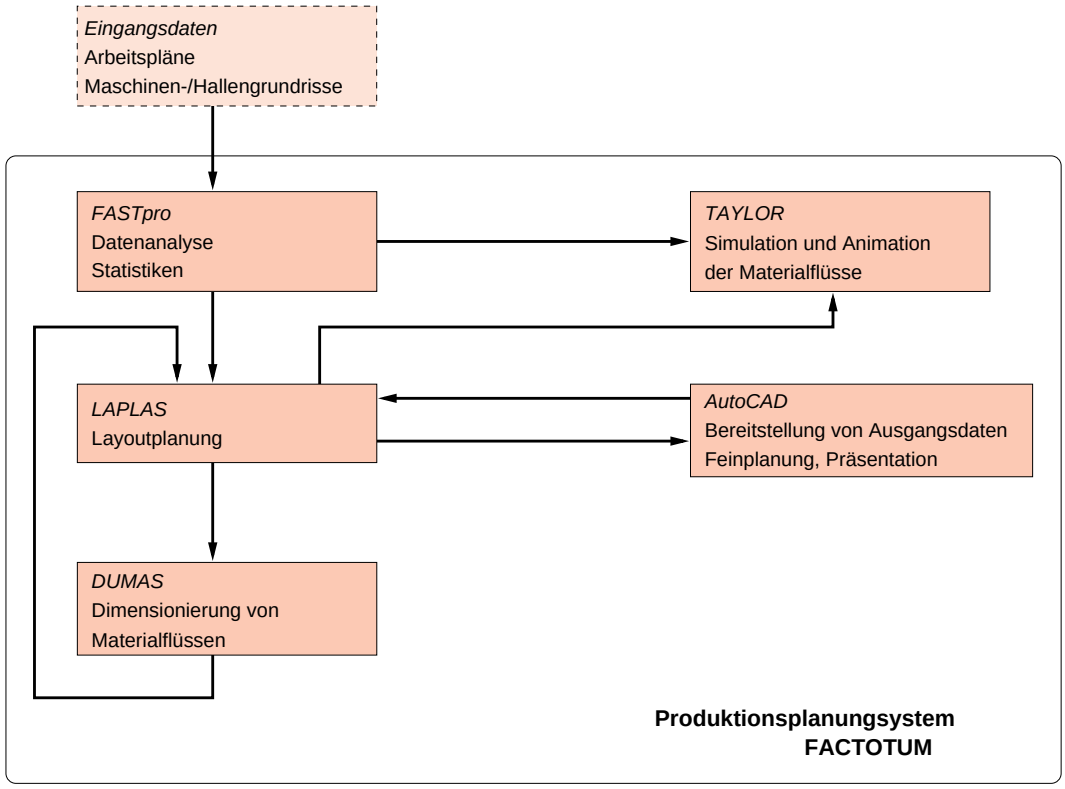

Abbildung 5.1: Das Produktionsplanungssystem FACTOTUM

Alle Systeme außer LAPLAS existieren auch in Versionen für *Microsoft WINDOWS 3.1*. In der aktuellen Version des FACTOTUM wird jedoch nur das System FAST/pro unter *WINDOWS* eingesetzt, die übrigen Tools sind in FACTOTUM in ihrer *MS-DOS*-Version enthalten.

Im folgenden sollen die einzelnen Komponenten und die Bedeutung ihrer Einsatzbereiche für die Produktionssystemplanung näher betrachtet werden. In einem ersten Schritt zur Föderierung dieser Systeme werden dabei auch die von den einzelnen Produkten verwendeten Datenhaltungskonzepte analysiert.

5.1.1 Gewinnung von Eingangsdaten - FAST/pro, AutoCAD

Das Fertigungsablauf-, Analyse- und Statistik-Softwarepaket FAST/pro ist ein recht umfangreiches System zur Planung, Steuerung und Überwachung der Auftragsabwicklung bzw. des Auftragsdurchlaufes im Produktionsbetrieb [Grö96]. Wesentliche Aufgaben, die mit dem System bearbeitet werden können, sind die Einplanung von Aufträgen, die Erkennung von potentiellen Schwachstellen und Problemen bei der Durchführung eines Auftrags sowie der Test und die Bewer-

tung verschiedener Planungsmethoden und Steuerungsmaßnahmen mit Hilfe der in FAST/pro integrierten Simulationsfunktionen. Die Bedeutung von FAST/pro für FACTOTUM liegt in der Erweiterung der Fähigkeiten dieses Systems um Funktionen zur Analyse der Auftragsabwicklung. Bei der Bearbeitung von Projekten mittels FAST/pro entstehen Daten über erforderliche Arbeitsgänge, die in den folgenden Komponenten von FACTOTUM als Ausgangsdaten dienen.

Die Datenhaltung in FAST/pro erfolgt in einer relationalen Datenbank, auf die über eine SQL-Schnittstelle mittels SQL-Anfragen zugegriffen werden kann. Die Struktur dieser relationalen Datenbank ist mit dem System FAST/pro fest vorgegeben und kann vom Benutzer nicht verändert werden. Die Anbindung an die restlichen Komponenten des FACTOTUM erfolgt zur Zeit mit Hilfe von SQL-Anfragen, die Dateien erzeugen, die dann wiederum mit Hilfe von Transformatoren (*C++*-Kommandozeilenprogrammen) an das erforderliche Datenformat angepaßt werden.

Daten aus FAST/pro werden verwendet von den Systemen LAPLAS und TAYLOR. Die genaue Struktur der beteiligten Dateien sowie die Beschreibung der derzeitigen Schnittstellen zu den Systemen LAPLAS und TAYLOR findet sich in Anhang A.1.

Zusammengefaßt stellt sich FAST/pro als ein System dar, das zwar als Datenbanksystem aufgefaßt werden kann, das aber trotzdem Dateien zum Datenaustausch mit FACTOTUM-Komponenten verwendet. Diese Dateien haben einfache satzorientierte Formate.

5.1.2 Layoutplanung - LAPLAS

Das Layoutplanungssystem LAPLAS stellt den Kern des FACTOTUM dar. LAPLAS führt einen der wichtigsten Schritte des Planungsprozesses, die Positionierung der Maschinen, aus und hält die Mehrzahl der im gesamten Prozeß verwendeten Daten. Das System LAPLAS hat jedoch in der vorliegenden Version einige schwerwiegende Nachteile. So liegt es als einziges System nur in einer *MS-DOS*-Version vor, was z.B. einen Mehrbenutzereinsatz des FACTOTUM erheblich erschwert. Außerdem verfügt LAPLAS über keinerlei Verzeichnisverwaltung. Das macht eine saubere Trennung verschiedener Projekte voneinander praktisch unmöglich, da außerdem noch wichtige Dateinamen für den Datenaustausch nicht automatisch erzeugt, sondern durch den Benutzer ausgewählt werden müssen.

Als Ausgangsdaten für eine Layoutplanung in LAPLAS dienen neben den von FAST/pro erzeugten Daten lediglich Bibliotheken mit Maschinenformen, die z.B. mit AutoCAD erzeugt, aber auch in LAPLAS direkt interaktiv eingegeben werden können. Eine Analyse der Dateistrukturen findet sich in Anhang A.2. Dabei ist eine recht komplexe Dateienstruktur zu erkennen, die im wesentlichen stark

strukturierte Formate verwendet, aber auch einige satzorientierte Formate implementiert.

Alle weiteren für die Maschinenpositionierung nötigen Daten müssen entweder interaktiv im System eingegeben werden, oder mittels eines geeigneten Systems in die Dateistruktur des LAPLAS geschrieben werden, wobei auf syntaktische und semantische Korrektheit zu achten ist.

LAPLAS ist in der Lage, einige während der Layoutplanung erhaltene Daten zu exportieren. Insbesondere sind dies aktualisierte Materialflußinformationen und das optimierte Layout der Halle in den Formaten DXF oder HPGL [Bor92]. Die überarbeitete Materialflußmatrix dient als Basis für eine Materialflußdimensionierung durch DUMAS (Kapitel 5.1.3). Das Hallenlayout im DXF-Format wird in FACTOTUM verwendet, um mit Hilfe von AutoCAD eine Feinplanung vorzunehmen sowie eine optisch ansprechende Darstellung der erzielten Ergebnisse zu erreichen.

5.1.3 Dimensionierung der Materialflüsse - DUMAS

DUMAS ist eine Abkürzung für den Begriff „Durchsatz von Materialflußsystemen“. Dieses Werkzeug dient der Auswertung von mit LAPLAS gewonnenen Planungsergebnissen, dabei insbesondere der Dimensionierung des ver- und entsorgenden Materialflußsystems (Anzahl, Auslastung, Blockierung von Transportmitteln), der Produktionspuffer und Lagerbereiche sowie zur Kostenbetrachtung und zur bedientheoretischen Analyse der konzipierten materialflußtechnischen Lösung [Grö96].

Hauptfunktionen sind dabei:

- *Optimierung.* Vogelsche Approximation für die Anzahl der Fahrten, Berechnung der Streckenlänge für die Routen, Rundfahrtberechnung.
- *Dimensionierung.* Berechnung der Arbeitsspielzeiten und der Transportmittelanzahl.
- *Transportmittel-Vergleich.* Auslastung, Zeitfonds und Kosten der einzelnen Transportmittel.
- *Zusatzfunktionen.* Interne/externe Blockierung für jede Maschine und jedes Transportmittel, Berechnung der Wartezeit und Warteschlangenlänge, Test auf Simulationswürdigkeit.

Das System DUMAS wurde zur direkten Erweiterung des Funktionsumfanges von LAPLAS konzipiert. Es baut daher auf dem proprietären Dateiformat von LAPLAS auf, kann also LAPLAS-`*.lyt`-Dateien sowie Materialflußmatrizen direkt

einlesen. Für eine Sitzung in DUMAS ist eine Erstellung eines neuen Projektes nicht notwendig, da DUMAS ein LAPLAS-Projekt einfach erweitern kann. Bei der Bearbeitung eines Projektes mit DUMAS werden die Positionen der Quellen und Senken jeder Planungsfläche (Maschine) aus der LAPLAS importiert sowie eine Materialflußmatrix für jedes Transportmittel verlangt. Geänderte Projektdaten können als neue DUMAS-Projekte abgespeichert werden, dies ist jedoch im FACTOTUM meist nicht erforderlich. Weiterhin ist eine Rückspeicherung optimierter Materialflußmatrizen für die Verwendung in LAPLAS möglich. Um jedoch diese Daten in einem LAPLAS-Projekt zu benutzen, muß dort ein kompletter Neuaufbau der Materialflußinformationen vorgenommen werden, da keine automatische Übernahme der Daten der Materialflußmatrix in die Layout-Datei vorgesehen ist.

Die Anpassung von DUMAS an LAPLAS ist nicht konsequent durchgeführt worden. Die zwei entscheidenden Mängel sind:

- DUMAS benötigt zur Ausführung einer Sitzung Materialflußdaten von LAPLAS. Obwohl sich in einer Layout-Datei (`*.lyt`) alle benötigten Daten finden (siehe Anhang A.2), fordert DUMAS jeweils die Anlage einer separaten Materialflußmatrix (`*.mfm`), die in LAPLAS nur interaktiv mittels entsprechender Menüoption erzeugt werden kann.
- Eine Bibliothek von Transportmitteln steht sowohl in LAPLAS als auch in DUMAS bereit. Das führt zu Inkonsistenzen, da sich Änderungen an Transportmitteln in LAPLAS in DUMAS nicht bemerkbar machen (und umgekehrt).

Intern hält DUMAS eine Datei, die u.a. Kostendaten für Transportmittel und Hebezeuge enthält. Dies sind Daten, die teilweise auch in LAPLAS gehalten werden, was zu Inkonsistenzen und Fehlern beim Einsatz von DUMAS führen kann. Hier bietet sich ein Ansatzpunkt für eine Verbesserung der Funktionalität des FACTOTUM durch eine Datenbankintegration.

Für die Formate der im Austausch von Daten benutzten Dateien ergeben sich keine neuen Erkenntnisse. Das Format der internen Transportmittelbibliothek ist in Anhang A.3 dargelegt. Hier handelt es sich um eine stark strukturierte Datei.

5.1.4 Simulation - TAYLOR

Simulation in der Fabrikplanung

Im Rahmen der Optimierung der Ablauforganisation eines Fertigungsprozesses spielt die Frage nach der Möglichkeit einer vorab erfolgenden Simulation eine große Rolle. Durch die Computersimulation einer Fertigung ist es möglich, mit

relativ geringem Aufwand eine experimentelle Verbesserung z.B. der Maschinenaufstellung, der Transportwege und verwendeten Transportmittel vorzunehmen. Dies führt in Fällen, in denen eine analytische Herangehensweise zu komplex ist, zu zwar ungenaueren, aber in vergleichsweise kurzen Zeiträumen erzielbaren Ergebnissen. Diese für Simulationen allgemein gültige These wird durch in der Fakultät für Maschinenbau vorgenommene Forschungen unterstützt. Mit [Kre95] liegt eine Arbeit vor, in der der Nutzen der Computersimulation in der Planung der Arbeitsorganisation untersucht wird. Krenzel kommt dabei zu folgenden Schlußfolgerungen:

- Durch die erforderliche hohe Flexibilität moderner Industrieunternehmen ist die Um- und Neustrukturierung von Produktionsprozessen notwendig geworden. Dabei hat die Komponente Organisation einen hohen Stellenwert, da durch optimierte Unternehmensorganisation eine Flexibilitätssteigerung und Kostenreduzierung möglich ist. Hier bietet sich die Simulation als ein Mittel der Optimierung an.
- Die computerunterstützte Simulation bietet die Möglichkeit, in der Praxis anfallende große Datenmengen zu verarbeiten und in vertretbarer Zeit eine große Anzahl von Varianten experimentell auszuwerten.
- Die in modernen Simulationssystemen vorhandenen Animationstools bieten eine effiziente Möglichkeit, über die anschauliche Darstellung des Produktionsprozesses von der Umorganisation betroffene Mitarbeiter in den Planungsprozeß einzubeziehen.

In diesem Zusammenhang treten jedoch noch Mängel und Schwächen zur Zeit verfügbarer Simulationstools zutage. In [Kre95] wird hier insbesondere die fehlende Integration von Bewertungsverfahren für die simulierten Tätigkeiten genannt. Ein weiteres Problem, das in diesem Zusammenhang auftritt, ist die schwierige Erfassung der für die Simulation nötigen Daten. Hier wird der offensichtliche Nutzen eines integrierten Fabrikplanungssystems deutlich, das in der Lage ist, die genannten Probleme weitgehend zu lösen. Weiterhin wird festgestellt, daß die vorhandenen Simulationssysteme entweder nicht flexibel genug (grafische Simulationstools) oder zu kompliziert zu erlernen und zu benutzen sind (Simulationssprachen wie GPSS). Das von Krenzel nicht untersuchte System TAYLOR bietet in dieser Hinsicht mit seiner grafisch orientierten Oberfläche und der zusätzlichen, relativ mächtigen Simulationssprache nach Ansicht des Autors einen akzeptablen Kompromiß für den FACTOTUM.

Das Simulationssystem TAYLOR

TAYLOR ist ein für die Fabrikplanung spezialisiertes grafisches Simulationssystem, das z.Z. nur in einer DOS-Version verfügbar ist, aber in Zukunft auch unter

Windows zur Verfügung stehen soll. Es ist in der Lage, Simulationen im Bereich der Fabriksimulation durchzuführen und perspektivische Abbildungen einer Fabrikhalle zu erzeugen. TAYLOR verwendet für alle Zwecke des Ex- und Imports eine Programmierschnittstelle (TLI - *TAYLOR Language Interface*), die eine prozedurale Sprache implementiert.

Mit dem TLI lassen sich eine Anzahl von Funktionen definieren, die interaktiv zur Laufzeit von TAYLOR aufgerufen werden können. Dazu steht eine große Anzahl von Befehlen zur Verfügung. Allerdings fehlen Sprachelemente zur Erzeugung von Simulationsobjekten wie Maschinen oder Materialflüssen. Es lassen sich lediglich bereits vorhandene Datenelemente (Variablen) verändern. Damit ist die Sprache nicht in der Lage, den Aufbau eines neuen Projektes aus vorhandenen FACTOTUM-Daten zu unterstützen, so daß neue Projekte prinzipiell manuell neu erzeugt werden müssen. Das bedeutet einen erheblichen redundanten Arbeitsaufwand, der allerdings aufgrund fehlender Mächtigkeit des TLI unumgänglich ist. Der Nutzen des TLI beschränkt sich damit auf die Ausführung von Simulationen sowie auf ein mögliches „Verschieben“ von Maschinen, z.B. im Rahmen einer Änderungsplanung. Weiterhin existiert eine Funktion zum Export von Simulationsergebnissen auf eine Datei zur Laufzeit einer Simulation.

Im Rahmen des FACTOTUM existieren zur Zeit TAYLOR-Funktionen, die die im Anhang beschriebenen Dateien als Ausgangsdaten für eine Simulation importieren. Dazu muß allerdings, wie oben ausgeführt, zunächst per Hand ein Simulationsmodell erzeugt werden. Dazu wird ein Grundriß der Fabrikhalle aus LAPLAS im HPGL-Format importiert (die einzige Import-Funktion von TAYLOR neben den Funktionen). Dann werden vordefinierte Maschinensymbole anhand der Hintergrund-Zeichnung manuell plaziert. (Hier könnte eine Unterstützung durch eine TLI-Funktion erfolgen, wobei dann zunächst jede Maschine manuell grob plaziert werden müßte, um dann per Funktion an die richtige Stelle gesetzt zu werden.) Ist dieser Vorgang beendet, kann mit Hilfe der importierten Materialflußinformationen die Simulation durchgeführt werden. Das aus LAPLAS importierte Hintergrundbild wird dann nicht mehr benötigt. Neben den erwähnten Importfunktionen existieren z.Z. noch eine Reihe von Funktionen, die bestimmte Simulationsinformationen in eine ASCII-Datei exportieren.

Die Verzeichnisverwaltung von TAYLOR beschränkt sich auf die Auswahl eines Arbeitsverzeichnisses, in dem dann alle Filezugriffe erfolgen. Ein Wechsel des Verzeichnisses während des Imports oder Exports von Daten ist nicht möglich.

Das System TAYLOR enthält keine im Sinne von Kapitel 3 aussagekräftigen Dateien. Es kann allerdings ein Datenimport und -export mit Hilfe von spezifischen TLI-Funktionen und TAYLOR-Prozeduren erfolgen. Diese „Schnittstelle“ ist in Anhang A.4 beschrieben.

TAYLOR verwendet zum Datenaustausch Dateien bereits beschriebener Formate. Die interne Datenhaltung erfolgt mit Hilfe einer Binärdatei, deren Struktur nicht zu entschlüsseln war.

5.1.5 Graphische Weiterverarbeitung - AutoCAD

Nach der Optimierung eines Fabrikhallenlayouts mittels LAPLAS, DUMAS und TAYLOR liegt im allgemeinen eine Vektorgrafikdatei im `*.dxf`-Format vor, die die fertige Fabrikhalle mit den Grundrissen der positionierten Maschinen darstellt. (Außerdem sind im Prozeß eine Reihe statistischer Daten entstanden, für die aber im Rahmen des FACTOTUM keine standardisierte Auswertung vorgenommen werden kann.)

AutoCAD dient hier zur Darstellung und Weiterverarbeitung der gewonnenen Layoutdatei. In den Planungssystemen nicht berücksichtigte Randbedingungen (wie z.B. nötige Mindestabstände von Transportmitteln zu bestimmten Maschinen) müssen jetzt manuell nachgearbeitet werden (Feinplanung). Nach Abschluß dieser Arbeiten kann das Aussehen des Hallengrundrisses optisch verfeinert werden.

Dazu werden im FACTOTUM farbige grafische Darstellungen von Maschinen verwendet, die sowohl in Riß- als auch in perspektivischer Darstellung vorliegen, so daß zwei- und dreidimensionale (perspektivische) Ansichten der Fabrikhalle erzeugt werden können. Dazu ist in AutoCAD wiederum eine manuelle Plazierung der Maschinenabbildungen auf dem importierten Grundriß der Fabrikhalle nötig. Mit dem zu AutoCAD optional erhältlichen Rendering- und Animationswerkzeug ARE-24 (AutoCAD Rendering Extension) können dann Animationen („Videos“) einer Fahrt durch die Fabrikhalle erzeugt werden, wobei hier der Aufwand erheblich ist.

Ergebnisse dieses Prozesses können auf dem WWW-Server des IAF betrachtet werden.

Eine genaue Untersuchung der von AutoCAD benutzten Datenhaltungskonzepte war nicht notwendig.

5.1.6 Ansätze zur Integration - CADLOG

Bei CADLOG handelt es sich um ein Konvertierungsprogramm, das einige Aufgaben der Integration der verschiedenen FACTOTUM-Komponenten übernimmt. Zusammen mit einigen *MS-DOS*–Batch-Dateien dient es als Hauptmenü für den FACTOTUM, wird allerdings z.Z. im IAF nicht dazu eingesetzt. Neben der Funktion als Starthilfe für die FACTOTUM-Komponenten kann CADLOG einige Konvertierungen ausführen. Das sind im einzelnen:

1. *DXF->LAPLAS*
   Es können Planungsflächen (also Maschinen) im DXF-Format an LAPLAS angepaßt werden. Die LAPLAS-Importfunktion liest nur eine eingeschränkte Version des DXF-Formats, in der nur rechtwinklige, geschlossene Linienzüge, keine Multilinien, verwendet werden dürfen. Die Positionen von Quelle und Senke werden durch Kreise innerhalb der Planungsfläche markiert, und der Name der Fläche wird in die Planungsfläche eingetragen.[3]

   So können Maschinen in AutoCAD gezeichnet und dann (als DXF-Dateien) in eine LAPLAS-Bibliothek importiert werden. Dazu werden die entsprechenden Daten von CADLOG direkt in eine vorhandene oder neue Flächenbibliothek eingefügt.

2. *Bibliotheks-Konvertierung*
   In LAPLAS werden absolute Größen für die Abmessungen von Planungsflächen verwendet. Bei Wiederverwendung von Maschinen in einem anderen Zeichenmaßstab ist eine Konvertierung notwendig, die durch CADLOG vorgenommen werden kann.

3. *Hallenlayout*
   Hier können einfache, regelmäßige Hallenlayouts direkt als neue Versionen eines bestehenden LAPLAS-Projekts angelegt werden (auch das Anlegen eines neuen Projekts ist möglich). Dabei wird eine neue `*.lyt`-Datei und mithin eine neue Variante direkt erzeugt, die dann in LAPLAS verwendet werden kann.

4. *Fabriklayout*
   Analog der Erzeugung eines Hallenlayouts können hier bestehende Vektorgrafiken im DXF-Format in neue Versionen eines LAPLAS-Projekts konvertiert werden.

CADLOG speichert keine Informationen ab. Die zu verwendenden Verzeichnisse werden dem System über entsprechende Umgebungsvariablen des DOS mitgeteilt.

5.2 Andere Ansätze zur integrierten Fabrikplanung

5.2.1 Integriertes System zur Fertigungsplanung

In [Sch90] beschreibt M. Schulze Dieckhoff ein integriertes System zur Fertigungsplanung, das auf einer HP9000/300-basierten Anlage implementiert wurde. Auch

[3] aus: Online-Hilfe zu CADLOG

hier werden von der Datenerfassung bis zur Simulation alle Phasen der Fabrikplanung computerunterstützt durchgeführt. Die einzelnen Module führen chronologisch aufeinanderfolgende Aufgaben aus und benutzen dabei ein gemeinsames Datenhaltungssystem. Über den Grad der Homogenität der Datenhaltungskonzepte der verwendeten Module läßt sich anhand der vorliegenden Quelle allerdings nicht genügend aussagen, so daß keine Einschätzung hinsichtlich einer Föderierung der Daten vorgenommen werden kann. Für die Bereitstellung von Daten für eine Simulation aus den Daten der Arbeitspläne und der Layoutplanung wird ein Transformator verwendet, der aus vorhandenen Daten Quelltext in der Sprache des Simulationssystems SIMAN erzeugt. Im direkten Vergleich zu FACTOTUM zeigt sich hier zunächst der erhebliche Nachteil, daß aufgrund der Realisierung auf einem Großrechner ein Transport dieses Systems zu einem Kunden und damit eine Datenerfassung vor Ort unmöglich ist.

Die Arbeit von Schulze Dieckhoff bietet jedoch im theoretischen Teil interessante Hinweise zum Gebiet der Fabrikplanung. Es wird festgestellt, daß zwar durch eine Optimierung des Arbeitsablaufs einer Fertigung erhebliche Einsparungen bzw. Rentabilitätsverbesserungen erzielt werden können, daß das jedoch hauptsächlich durch Optimierung der Lagerhaltung (Bestände) und der Materialkosten geschehen kann. Eine Layoutplanung kann bei der Optimierung hilfreich sein, wirft jedoch einige Probleme auf. Zunächst ist die Aufgabe der Layoutplanung von hoher Komplexität und daher für große Aufgabenstellungen erst mit moderner Rechentechnik zu bewältigen. Weiterhin ist für ein Layoutplanungssystem eine sehr hohe Flexibilität erforderlich, da eine große Anzahl von Randbedingungen und Planungszielen berücksichtigt werden müssen. Drittens führt eine bisher meist vorgenommene Optimierung mit der Zielfunktion „minimale Transportleistung“ nicht immer zu einer entscheidenden Einsparung an Kosten. Grund dafür ist der allgemein geringe Anteil der Transportzeit an der gesamten Wartezeit eines Auftrags. Daher müssen weitere Teilziele gewichtet in der Zielfunktion der Optimierung berücksichtigt werden können (z.B. eine bestimmte Lage von Maschinen zueinander). Auch eine intensive Einbeziehung des Planers in den Optimierungsprozeß, also ein Verzicht auf vollständig automatisierte Optimierung, vermag hier die Ergebnisse zu verbessern. Insgesamt ist festzustellen, daß eine Verkürzung der *Durchlaufzeit* gegenüber einer reinen Verkürzung der Transportzeit bedeutende Vorteile bringt.

Das in dieser Diplomarbeit untersuchte Produktionsplanungssystem FACTOTUM unterstützt in mehrfacher Hinsicht bereits Lösungsansätze zu diesen Problemen. Insbesondere können hier sogenannte Sympathien von Maschinen zueinander festgelegt werden, die zu Gruppierungen bestimmter Maschinen führen, es können Flächen mit besonderen Eigenschaften (Attributflächen) definiert werden, und es ist möglich, solche Teilziele der Optimierung mit globalen Faktoren zu wichten.

5.2.2 FASTDESIGN

Auch in [Ric92] wird ein dem FACTOTUM ähnliches System beschrieben. Es handelt sich um das Fertigungsstruktur-Planungssystem FASTDESIGN, das aus den Teilen FASTPLAN (Strukturanalyse), FASTSIM (Simulation) und FASTGRAF (CAD-Funktionen, Darstellung) sowie einer Datenbasis mit Ausgangsdaten, Arbeitsplänen und Datenbankfunktionalität besteht. Die Integration dieser Teilsysteme wurde offenbar bereits im Entwurfsstadium der Software berücksichtigt, so daß das System FASTDESIGN eine wesentlich weniger heterogene Struktur aufweist als z.B. FACTOTUM. Allerdings handelt es sich hierbei um ein kompaktes System, dessen Komponenten nicht austauschbar sind. Vielmehr muß bei geplanten Erweiterungen auf die Unterstützung des Herstellers des Systems zurückgegriffen werden, was eine Investition in dieses System zumindest unsicherer macht. Für die Bereitstellung der CAD- und Datenbankfunktionalitäten des Systems wird auf etablierte Applikationen zurückgegriffen (AutoCAD, SQL-Server).

Zum System FASTDESIGN existiert noch ein Facility-Management-System mit der Bezeichnung FASTMAN, das für das Datenmanagement in der fertig geplanten Fabrik eingesetzt werden kann. Dies stellt einen sehr interessanten Ansatz dar, da ein effizientes Datenmanagement die Phase der Ist-Analyse bei nötigen Umplanungen einer Fabrik drastisch verkürzt und somit wesentlich zur Rationalisierung beiträgt.

5.2.3 CAD-FAIF

Am Institut für Betriebswissenschaften und Fabriksysteme der Technischen Universität Chemnitz-Zwickau wurde das Fabrikplanungssystem CAD-FAIF entwickelt [Aut91]. Hierbei handelt es sich um ein modulares System zur Erfüllung verschiedener Aufgaben im Bereich der Fabrikplanung wie Erfassung und Aufbereitung der Planungsdatenbasis, Strukturierung des Teilesortiments, statische Ressourcendimensionierung, Strukturierung der Fertigung und Optimierung und Verifizierung der Planungsergebnisse. Die einzelnen Module sind homogen aufgebaut und verwenden eine einheitliche Datenbasis. Eine Integration von Systemen fremder Hersteller ist nicht vorgesehen, so daß die Verantwortung für eine Weiterentwicklung des Systems allein bei der TU Chemnitz liegt. Zur Zeit befindet sich ein Planungstool zur „Effizienten Werkstättenplanung“ in der Entwicklung, das eine teilweise Unterstützung des auch im FACTOTUM verwendeten Simulationssystems TAYLOR bietet. In diesem Tool wird eine Grundarchitektur für eine Werkstatt festgelegt, die für eine Werkstättenplanung parametrisiert werden kann. Diese Vorgehensweise umgeht das Problem der Unmöglichkeit eines automatischen Aufbaus eines Taylor-Modells (s. Kapitel 5.1.4). Allerdings läßt sich so nur eine sehr begrenzte Menge von Werkstätten planen (Einschränkungen sind

z.B. 20 Fertigungsplätze, 1 Lager, 4 Transporteinrichtungen, 60 Personen).

CAD-FAIF wird im Rahmen eines Projektes der TU Chemnitz zur rechnergestützten Fertigungssegmentierung eingesetzt und dient dort zur Grobplanung und zur Bereitstellung von Daten für das System CAD-SIPLAN.

5.3 Einsatz von Datenbanksystemen in der Fabrikplanung

Die in Kapitel 5.1 beschriebenen Komponenten des Systems FACTOTUM weisen nur geringe Merkmale einer Integration und Kooperation auf. Da aber ein enger logischer Zusammenhang zwischen den Komponenten besteht, und ein Datenaustausch zwischen einzelnen Applikationen in größerem Umfang nötig ist, liegt der Einsatz eines Datenbanksystems zur Integration nahe. Aus den bereits erläuterten Gründen (Investitionsschutz, weitgehender Erhalt der Selbständigkeit der Komponenten, gute Voraussetzungen für eine Vereinheitlichung des Datenzugriffs) bietet sich hier eine Integration in ein föderiertes Datenbanksystem an. Im Rahmen des Projektes **SIGMA**$_{FDB}$ [HTJ+95] werden Untersuchungen zu diesem Thema angestellt. Daher soll in dieser Arbeit eine Integration der FACTOTUM-Einzelsysteme in ein FDBS konzeptionell erfolgen, wobei die Besonderheiten solcher Ingenieuranwendungen (s. Kapitel 3) berücksichtigt werden müssen.

Es wurden bereits von anderer Seite Versuche zur Integration von Ingenieuranwendungen vorgenommen. Hübel und Sutter beschreiben in [HS93] Arbeiten zur Datenbankanbindung von Ingenieursystemen. Dabei wird ein „Workstation-Server-Datenbanksystem zur Handhabung komplex-strukturierter Verarbeitungsgegenstände“ entwickelt und beschrieben. Die Autoren konzentrieren sich auf eine mögliche Gestaltung von Server- und Client-DBS und schlagen eine Grob-Architektur für ein Workstation-DBS vor. Dabei wird besonderer Wert auf Zugriffseffizienz und Ablaufkontrolle gelegt und Aspekte wie Verteilungstransparenz und Autonomie der Ingenieuranwendungen weniger berücksichtigt. Auch werden keine Konzepte für eine konsistente Schemaintegration heterogener Komponenten vorgestellt, sondern es wird auf die Entwicklung homogener Datenbankschnittstellen für jede Komponente orientiert. Als Beispiel für eine Implementation der vorgestellten Konzepte wird das Non-Standard-DBS PRIMA vorgestellt, das auf einem Molekül-Atom-Datenmodell basiert.

Kapitel 6

Ableitung konzeptioneller Schemata

Der erste Schritt im Design eines föderierten Datenbanksystems nach der Schemaarchitektur von Sheth und Larson ist die Ableitung konzeptioneller Schemata für die Komponentensysteme. Dabei besteht dieser Schritt aus zwei Phasen: der Aufstellung lokaler Schemata im vom Komponentensystem benutzten Datenmodell und der Transformation dieses Schemas in das Datenmodell des föderierten Datenbanksystems (hier GIM). Während der erste Schritt im Falle herkömmlicher Datenbanksysteme trivial ist, da bereits Datenbankschemata in einem definierten Datenmodell vorliegen, ist im Falle dateibasierter Komponenten zunächst zu entscheiden, welches Datenmodell für eine Modellierung genutzt werden soll.

Zur Auswahl stehen verschiedene Modelle unterschiedlicher Komplexität und semantischer Reichhaltigkeit (z.B. ER-Modelle, erweiterte ER-Modelle, objektorientierte Modelle). Das Datenmodell GIM zeichnet sich durch einen semantisch armen Ansatz aus und steht damit relationalen Modellen näher als objektorientierten. Der Autor vertritt die Auffassung, daß bei freier Wahl des lokalen Datenmodells eine relationale Modellierung günstiger ist als die Aufstellung eines objektorientierten Schemas. So kann in einem relationalen Schema nicht mehr Information ausgedrückt werden als in GIM modelliert werden kann, wodurch sich die Aufstellung von Schemata zur Modellierung semantischer Relativismusinformationen (s. Kapitel 4) erübrigt. Informationen über die Komponentensysteme, die im ER-Modell nicht ausgedrückt werden können, können vielmehr über entsprechende Integritätsbedingungen modelliert werden.

Eine weitere Möglichkeit, die sich aufgrund fehlender Datenmodelle für lokale Schemata bietet, wäre, auf die Erstellung solcher Schemata ganz zu verzichten und eine Modellierung sofort im Datenmodell des föderierten Systems vorzunehmen. Allerdings erhöht eine vorherige Modellierung in einem herkömmlichen Datenmodell die Anschaulichkeit und somit die Überprüfbarkeit der Datenbanksche-

mata. Außerdem muß nach Erstellung des föderierten Schemas ohnehin ein externes Schema in einem anderen Datenmodell als GIM erstellt werden (vgl. S.35), so daß sich hier die Möglichkeit eines Vergleichs zwischen externem Schema und lokalen Schemata bietet.

Für die Schematransformation in GIM werden die in Kapitel 4 vorgestellten Konzepte verwendet. Da die dort gegebene Zusammenfassung aus Platzgründen nicht alle im folgenden verwendeten Integrationsschritte und Algorithmen vorstellt, wird auf ausführlichere Beschreibungen der verwendeten Vorgehensweise in [CHJ+96, Sch95, SS96a, SS96b, SS96c] verwiesen.

Die in diesem Kapitel vorgestellten Modellierungsschritte für die einzelnen Systeme sind im Zusammenhang mit der verbalen und SGML-Strukturbeschreibung in Kapitel 5.1 zu sehen.

6.1 Modellierung von FAST/pro

6.1.1 FAST/pro - Lokales Schema

Bei FAST/pro handelt es sich um ein relationales Datenbanksystem mit SQL-Schnittstelle. Aus diesem Grund läge es nahe, FAST/pro nicht als dateibasiertes Datenhaltungssystem, sondern als relationales Datenbanksystem zu behandeln und demzufolge das FAST/pro-eigene relationale Schema als lokales Schema zu benutzen. Allerdings zeigt eine Betrachtung der den anderen FACTOTUM-Applikationen von FAST/pro zur Verfügung gestellten Daten, daß die bisher verwendeten Transformatoren eine semantische Anreicherung der FAST/pro-Daten vornehmen. Insbesondere wird erst beim Übergang von SQL in das Dateienmodell dem Attribut Transportmittel ein Wert zugewiesen (s. S.109). Daher wurde eine Modellierung der für den Datenaustausch benutzten *Dateien* einer Modellierung des Gesamtsystems als relationales Datenbanksystem vorgezogen. Diese Vorgehensweise vereinfacht gleichzeitig auch das lokale Schema und damit alle weiteren Schemata, da nur diejenigen Teile von FAST/pro modelliert werden müssen, die eine Überlappung mit den restlichen FACTOTUM-Komponenten bilden.

Das entstehende ER-Schema für FAST/pro zeigt Abb. 6.1.

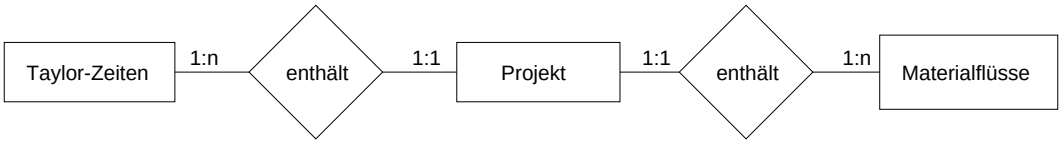

Abbildung 6.1: ER-Diagramm FAST/pro

6.1.2 FAST/pro - Schematransformation in GIM

Die Transformation dieses Schema in GIM stößt nicht auf besondere Schwierigkeiten. Die beiden Klassen MFM und T_Z erfüllen alle Voraussetzungen für eine GIM-Transformation (einfache Datentypen, nicht überlappende Klassenextensionen) und können so recht einfach abgebildet werden (Abb. 6.2).

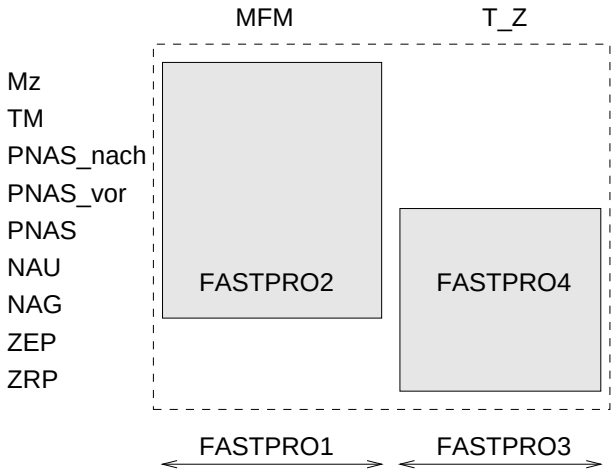

Abbildung 6.2: GIM-Schema FAST/pro

Eine Beschreibung der Attribute dieser Klassen findet sich im Anhang in der Tabelle B.1.

Folgende Integritätsbedingungen müssen aufgestellt werden:

FASTPRO1$_{TIC}$[NAU,NAG][MFM]: (NAU,NAG) is unique(IC$_{mn}$, intra-class)

FASTPRO2$_{TIC}$[NAU,NAG][MFM]: (NAU+NAG) is not empty(IC$_{1n}$, intra-class)

FASTPRO3$_{TIC}$[NAU,NAG][T_Z]: (NAU,NAG) is unique(IC$_{mn}$, intra-class)

FASTPRO4$_{TIC}$[NAU,NAG][T_Z]: (NAU+NAG) is not empty(IC$_{1n}$, intra-class)

Diese Bedingungen sind die im relationalen Modell implizit enthaltenen Schlüsselbedingungen, die bei der Transformation in GIM explizit zutage treten. In den weiteren Schemata für die anderen FACTOTUM-Systeme sollen diese Bedingungen aus Aufwandsgründen nicht ausführlich angegeben werden, da ihre Ableitung anhand des hier angegebenen Beispiels möglich ist. Sie werden jedoch der Vollständigkeit halber erwähnt und in den GIM-Schemata angeführt.

6.2 Modellierung von LAPLAS

LAPLAS bildet sowohl vom Funktionsumfang als auch vom Umfang und der Anzahl der verwalteten Dateien den Kern des FACTOTUM. Daher sind hier

auch die bei weitem umfangreichsten Modellierungsergebnisse zu erwarten. Wie im Anhang A.2 dargelegt, verwendet LAPLAS für die Mehrzahl der Einstellungen und Parameter eines Projekts globale *default*-Werte, die lokal überschrieben werden können. Da diese globalen Werte keine direkte Beziehung zu den restlichen Daten haben und nur als Kopiervorlage (*template*) dienen, werden sie in die Modellierung nicht mit einbezogen. Allerdings sind sie im ER-Diagramm der Vollständigkeit halber angegeben. Des weiteren bietet LAPLAS als einziges System des FACTOTUM eine Projektverwaltung an, die als Grundlage für eine systemweite Projektverwaltung verwendet werden könnte. Dies kommt im ER-Diagramm sowie im GIM-Schema zum Ausdruck.

6.2.1 LAPLAS - Lokales Schema

Das ER-Diagramm (Abb. 6.3, S. 65) zeigt keine Besonderheiten. Die nicht durch *relationships* mit dem restlichen Schema verbundenen Klassen *Globale Einstellungen, Globale Attribute* usw. dienen als Vorlage bei der Erstellung eines neuen Projekts. Die dort vorgeschlagenen Werte können in den entsprechenden variantenspezifischen Objekten beliebig geändert werden.

LAPLAS verwaltet eine Maschinenbibliothek, in der Maschinengrundrisse im LAPLAS-eigenen Format abgelegt werden können. Hierbei handelt es sich um eine interne Unterstützung der Arbeit mit LAPLAS, wobei dort gespeicherte Maschinengrundrisse jedoch wiederum nur Kopiervorlagen für neu zu erstellende Maschinen sind und daher ähnlich wie globale Einstellungen behandelt werden. Für eine Integration in ein föderiertes Datenbanksystem eignen sie sich nicht.

6.2.2 LAPLAS - Schematransformation in GIM

Bei der Schematransformation fällt auf, daß mehrere Entities des relationalen Modells Attribute enthalten, die komplexe und mengenwertige Datentypen aufweisen (s. Anhang). Im Sinne der GIM-Schematransformation müssen diese mengenwertigen Attribute in eigene Klassen mit entsprechenden Referenzen umgewandelt werden. Dies wird hier aus Aufwandsgründen nur soweit verfolgt, wie die entstehenden Klassen bei der Ableitung des integrierten Schemas eine Rolle spielen (Überlappungen). Für nicht überlappende Bereiche des LAPLAS-Schema wird auf eine Klassenbildung verzichtet. Das betrifft die Attribute attr, symp, koor und rec (s. Abb.6.4), die mengenwertig und von komplexen Datentypen sein können Diese Attribute werden jeweils als Einheit betrachtet und entsprechend modelliert. Ein föderiertes Datenbanksystem hätte dann lediglich die Aufgabe, einen binären Datentyp bereitzustellen, der Werte einer festzulegenden Maximalgröße aufnehmen kann, um diese mengenwertigen Attribute abzuspeichern. Eine weitere Auswertung der Semantik dieser Werte ist nicht notwendig. Eine Besonderheit

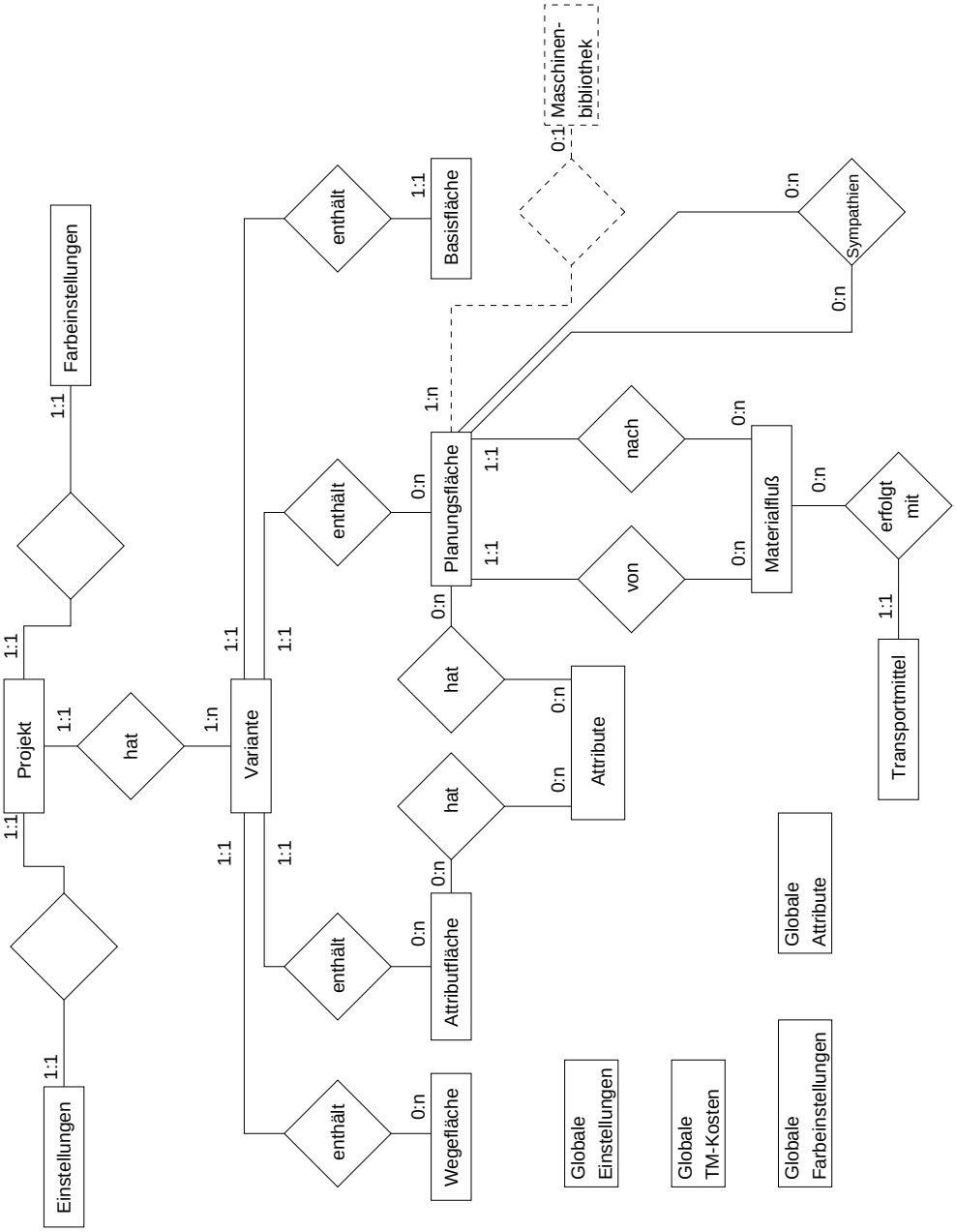

Abbildung 6.3: ER-Diagramm LAPLAS

stellt die Klasse TM dar, da sie nicht aufgrund einer Normalisierung mengenwertiger Attribute entsteht, sondern die Existenz einer echten Klasse (entsprechend dem ER-Entity TM-Kosten) widerspiegelt.

Eine Übersicht über die entstehende Klassenhierarchie in der Darstellungsweise des GIM zeigt Abb. 6.4.

Wie auch in den Untersuchungen von TAYLOR und AutoCAD deutlich werden wird, wurde auf eine GIM-Modellierung von Hallengrundrissen und fertigen Hallenlayouts als DXF- oder HPGL-Dateien verzichtet, obwohl diese Dateien bei der Arbeit mit FACTOTUM ein Rolle spielen können. Ihre Modellierung ist trivial und führt auch bei der Betrachtung des föderierten Schemas in Kapitel 7 zu trivialen Datenbankschemata. Zur Verdeutlichung der Vorgehensweise wird beispielhaft nur der Datenaustausch von DXF-Dateien zwischen AutoCAD und LAPLAS dargestellt.

Die Attribute der entstehenden Klassen sind in den Tabellen B.2 bis B.5 angegeben.

Die in LAPLAS auftretenden Bedingungen zur Sicherstellung der Integrität der Schlüsselattribute sind LAPLAS1 bis LAPLAS16. Dabei treten Besonderheiten auf, da bei allen abgeleiteten Klassen die Einzigartigkeit des Schlüssels nur in einem bestimmten Umfeld gilt. Es gilt dabei folgende Zuordnung (Tabelle 6.1):

Klasse	**IB**	**Geltungsbereich**
PRJ	LAPLAS1	global
VAR	LAPLAS3	global
BF	LAPLAS5	1 Instanz von VAR
RF	LAPLAS7	1 Instanz von VAR
WF	LAPLAS9	1 Instanz von VAR
PF	LAPLAS11	1 Instanz von VAR
MF	LAPLAS13	1 Instanz von PF
TM	LAPLAS15	global

Tabelle 6.1: Geltungsbereiche für Schlüsselbedingungen

Zur Wahrung der Konsistenz mit DUMAS muß die Erstellung lokaler Transportmittel-Bibliotheken explizit verboten werden. Die einzige erlaubte Operation ist eine direkte Kopie der globalen Transportmittelbibliothek, um die für ein neues LAPLAS-Projekt benötigte lokale Bibliothek zu erstellen. Änderungen an Transportmitteldaten dürfen nur in der globalen Bibliothek vorgenommen werden (und müssen dann vom FDBMS an die anderen Teile des FACTOTUM weitergegeben werden). Aus diesem Grund müssen Instanzen der GIM-Klasse TM auch global *unique* sein (Integritätsbedingung LAPLAS15).

In diesem GIM-Schema treten erstmals Integritätsbedingungen auf, die Aussagen über GIM-Referenzen treffen. Dabei kann ein Objekt einer neu entstandenen

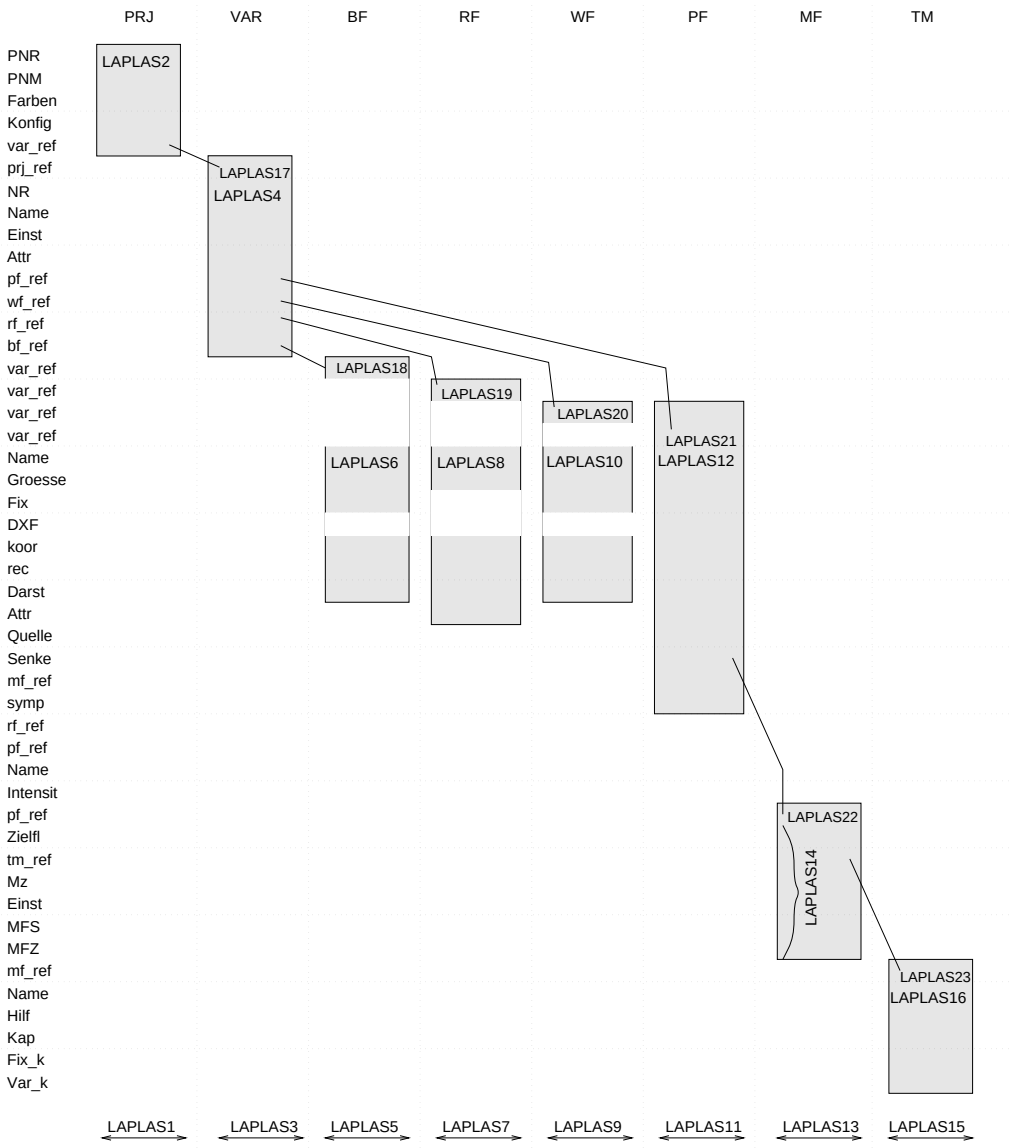

Abbildung 6.4: GIM-Schema LAPLAS

Klasse nur dann existieren, wenn mindestens *eine* Referenz auf dieses Objekt existiert. Eine solche Integritätsbedingung wird beispielhaft für die Klasse VAR angegeben:

LAPLAS17$_{TIC}$[prj_ref][VAR]: card(prj_ref)>0(IC$_{11}$, intra-class)

Die anderen Referenz-Bedingungen lassen sich sinngemäß ableiten.

6.3 Modellierung von DUMAS

6.3.1 DUMAS - Lokales Schema

DUMAS wurde entwickelt, um das System LAPLAS um Funktionen zur Materialflußanalyse zu erweitern. Da es sich bei DUMAS um ein speziell für LAPLAS entwickeltes Zusatzsystem handelt, erscheint eine Ähnlichkeit mit LAPLAS naheliegend. Allerdings wurden bei der Entwicklung von DUMAS wesentliche Eigenschaften von LAPLAS außer acht gelassen. So übernimmt DUMAS nicht die Projekt- und Variantenverwaltung aus der LAPLAS-Umgebung, sondern ist lediglich in der Lage, einzelne LAPLAS-Varianten einzulesen und auf den gelesenen Daten Berechnungen anzustellen. Das führt zu Schwierigkeiten bei der Konsistenzsicherung. Aus der fehlenden Projektverwaltung und daraus resultierenden Unmöglichkeit der projektgebundenen Abspeicherung von Berechnungsergebnissen resultiert das lokale Datenbankschema in Abb. 6.5. Die Relation TM-Bibliothek stellt Daten über Transportmittel in Form einer Bibliothek bereit. Wie später dargelegt werden wird, entstehen hier Inkonsistenzen bei der Zusammenarbeit mit LAPLAS. Ein Transport findet immer zwischen genau zwei Maschinen statt und wird von genau einem Transportmittel durchgeführt. Weitere Besonderheiten treten nicht auf.

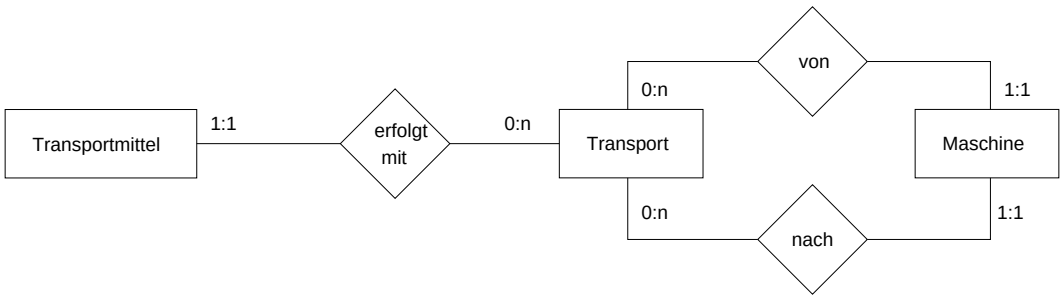

Abbildung 6.5: ER-Diagramm DUMAS

6.3.2 DUMAS - Schematransformation in GIM

Bei der Schematransformation in GIM treten keine bisher nicht angesprochenen Probleme auf. Die abgeleiteten Integritätsbedingungen auf den Schlüsseln sind DUMAS1 bis DUMAS6.

Eine Ableitung neuer Klassen war nicht notwendig. Bei den Attributen Geschw und Zeiten handelt es sich wiederum um komplexe Attribute, für die eine genauere Modellierung nicht notwendig war. Eine Beschreibung der Attribute der GIM-Klassen des Komponentenschemas findet sich im Anhang in der Tabelle B.6.

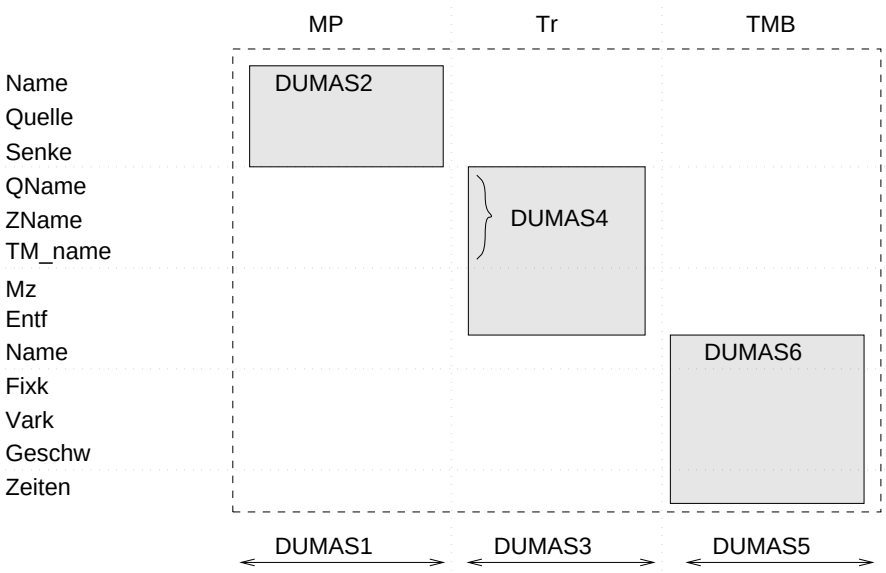

Abbildung 6.6: GIM-Schema DUMAS

6.4 Modellierung von TAYLOR

6.4.1 TAYLOR - Lokales Schema

Das System TAYLOR bereitet bei der Untersuchung Schwierigkeiten, da eine Analyse des TAYLOR-spezifischen Dateiformates nicht möglich war. Es handelt sich um Binärdateien, deren Format ohne nähere Informationen nicht erkennbar ist. Daher konnte nur der Datenaustausch mit TAYLOR über Import und Export von Dateien modelliert werden. Dies geschieht zum einen im HPGL-Format, in dem Hallengrundrisse importiert werden (*Entity* Grundriß) und zum anderen über den Import von Materialflußinformationen direkt aus FAST/pro. Auch kann TAYLOR berechnete statistische Werte auf Dateien ausgeben (Optimierter Materialfluß).

Weitere Import- und Exportfunktionen sind möglich (z.B. das Einlesen von Maschinenpositionen), sind aber aufgrund von Konsistenzproblemen nicht beherrschbar. (Mit Hilfe der TAYLOR-internen Sprache TLI lassen sich zwar Maschinenpositionen verändern, nicht aber Maschinenobjekte erzeugen. Daher müßte ein Benutzer, um Maschinenpositionen einlesen zu können, vorher die exakte Anzahl und Art aller Maschinen manuell erzeugen. Anderenfalls wäre das Einlesen der Maschinenpositionen nicht sinnvoll.)

Das so erhaltene ER-Diagramm zeigt Abbildung 6.7.

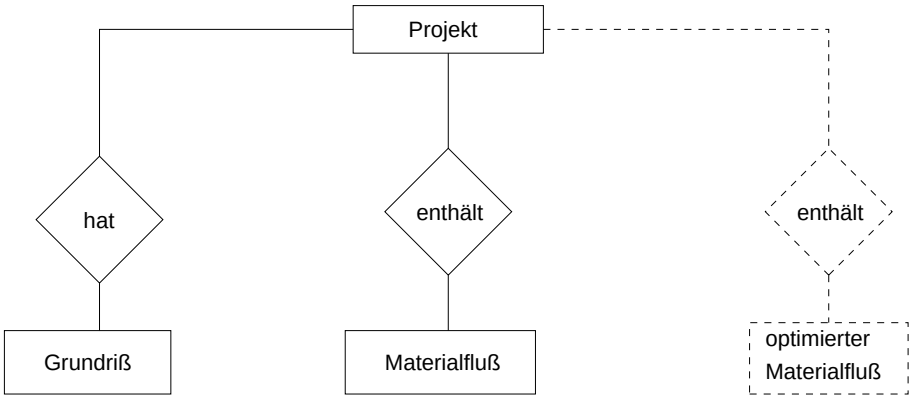

Abbildung 6.7: ER-Diagramm TAYLOR

6.4.2 TAYLOR - Schematransformation in GIM

Die Übernahme von Grundrissen im HPGL-Format von LAPLAS soll in GIM nicht modelliert werden (s. S.66).

Der Datenexport wird im Komponentenschema ebenfalls nicht aufgenommen. Das IAF hat für den Export von Statistikdaten keine Formate oder Standards festgelegt, anhand derer eine Modellierung möglich wäre. Die einzige verbleibende Klasse ist T_Z, die Aussagen über die Rüst- und Bearbeitungszeiten für einzelne Arbeitsgänge trifft (Abb. 6.8).

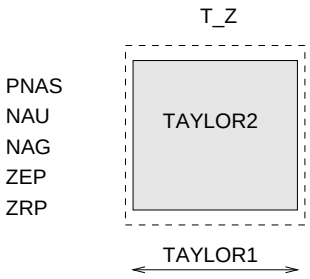

Abbildung 6.8: GIM-Schema TAYLOR

Eine Beschreibung der Attribute dieser Klasse findet sich im Anhang in der Tabelle B.7.

6.5 Modellierung von AutoCAD

6.5.1 AutoCAD - Lokales Schema

Die Aufgabe von AutoCAD besteht in der Bereitstellung von Zeichnungen für LAPLAS und in der Übernahme von Grafiken aus LAPLAS. Dies erfolgt auf der Basis frei gewählter Dateinamen und nicht im Rahmen einer Projektverwaltung. Daher wird das lokale Schema recht einfach (Abb. 6.9).

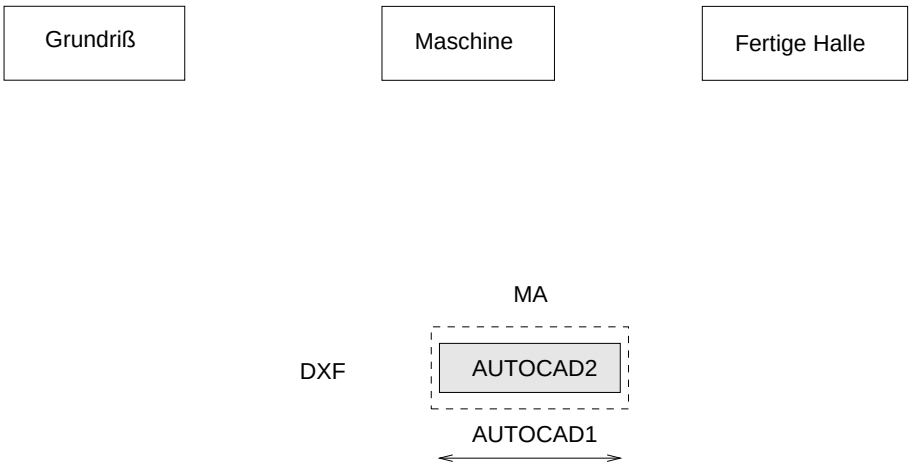

Abbildung 6.9: ER- und GIM-Diagramm AutoCAD

6.5.2 AutoCAD - Schematransformation in GIM

Aufgrund der nur beispielhaften Modellierung der Integration von AutoCAD in den FACTOTUM bleibt lediglich die Klasse MA im Komponentenschema erhalten, die für LAPLAS erzeugte Maschinengrundrisse abbildet. Das einzige dort gehaltene Attribut ist vom Typ `DXF` und bezeichnet eine komplette Vektorgrafik im DXF-Format. Eine Analyse der in dieser Datei enthaltenen Informationen ist nicht nötig, da die Datei über Importfunktionen vollständig und unverändert in LAPLAS übertragen wird. Die Attributtabelle befindet sich im Anhang (Tabelle B.8).

Kapitel 7

Integration in das Föderierte Schema

Nachdem die Ableitung der lokalen Schemata abgeschlossen ist und die Transformation in das Datenmodell der föderierten Datenbank (GIM) erfolgte, kann jetzt die Integration der einzelnen Komponentenschemata in das gemeinsam föderierte bzw. integrierte Schema vorgenommen werden. Die transformierten Integrationsbedingungen wurden aufgestellt und die Einhaltung der GIM-Bedingungen (z.B. einfache Datentypen) mit den beschriebenen Einschränkungen sichergestellt.

Im folgenden soll die Integration der Komponenten des FACTOTUM beschrieben werden, wobei auftretende Probleme aufgezeigt und Lösungen vorgeschlagen werden.

7.1 Ableitung des Föderierten Schemas

Die Ableitung des föderierten Schemas erfolgt in mehreren Schritten. Zunächst sind diejenigen Klassen der einzelnen Komponentensysteme zu identifizieren, die Aussagen über die gleichen Realweltobjekte treffen, sich also überlappen. Dabei ist zu beachten, daß ein paarweiser Vergleich der Klassen zwar hilfreich sein kann, aber nicht für eine vollständige Bestimmung extensionaler und intensionaler Überlappung ausreicht. Daher wurde in [SS96a] ein Algorithmus vorgestellt, mit dessen Hilfe Überlappungen von mehr als zwei Klassen erkannt und richtig behandelt werden können. Sind die betreffenden Klassen identifiziert, sind Überlegungen bezüglich der Art der Überlappung anzustellen. Dazu sind Metakonflikte und Attributkonflikte zu behandeln. Sind diese Konflikte gelöst, kann eine Bestimmung der Überlappungsbereiche erfolgen. Danach können noch globale Integritätsbedingungen hinzugefügt werden, woraufhin eine Aufstellung des föderierten Schemas möglich wird.

Um die Untersuchung von Überlappungen durchführen zu können, wurden zunächst Paare von Klassen identifiziert, die Aussagen über gleiche Objekte treffen. Dies stellt noch keine extensionale Komposition dar, es sollen lediglich Ansatzpunkte für eine Integration gefunden werden. Die sich überlappenden Klassen im FACTOTUM sind:

FAST/pro.MFM_m_l	$\Longleftrightarrow$	LAPLAS.MF
FAST/pro.MFM_l_m	$\Longleftrightarrow$	LAPLAS.MF
FAST/pro.MFM_m_m	$\Longleftrightarrow$	LAPLAS.MF
LAPLAS.MF	$\Longleftrightarrow$	DUMAS.TR
LAPLAS.PF	$\Longleftrightarrow$	DUMAS.MP
LAPLAS.PF	$\Longleftrightarrow$	AutoCAD.MA
FAST/pro.T_Z	$\Longleftrightarrow$	TAYLOR.T_Z
LAPLAS.TM	$\Longleftrightarrow$	DUMAS.TMB

Auf die Untersuchung von Überlappungen weiterer AutoCAD-Klassen (*Entities* Maschine, Fertige Halle) wird aus den oben genannten Gründen verzichtet.

7.1.1 Auflösung von Metakonflikten

Bei der Überlappung von Klassen können Konflikte auftreten, die in der unterschiedlichen Struktur dieser Klassen begründet sind. Ein häufiger Fall von Metakonflikten wird in Kapitel 4.2.4 vorgestellt.

Bei der Durchsicht der sich extensional überlappenden Klassen tritt ein Metakonflikt anderer Art zutage. Betrachtet man die Klasse FAST/pro.MFM, so ist zu erkennen, daß die Attribute PNAS, PNAS_vor und PNAS_nach Aussagen über sehr ähnliche Eigenschaften eines Objekts treffen. Es stellt sich heraus, daß abhängig vom numerischen Wert dieser Attribute ihre Zuordnung zu Attributen der Klasse LAPLAS.MF unterschiedlich ist. In Pseudocode könnte dieser Konflikt folgendermaßen beschrieben werden (vgl. Anhang A.1):

```
if (PNAS_vor==Null)
        { LAPLAS.PF.Name="Ein";
          related(LAPLAS.MF.Zielfl , FASTpro.MFM.PNAS    );  }
else if (PNAS_nach==Null)
        { related(LAPLAS.PF.Name   , FASTpro.MFM.PNAS    );
          LAPLAS.MF.Zielfl="Aus";                              }
     else
        { related(LAPLAS.PF.Name   , FASTpro.MFM.PNAS_vor);
          related(LAPLAS.MF.Zielfl , FASTpro.MFM.PNAS);      }
```

Die Bezeichnung der Klassen und Attribute folgt dem Komponentenschema. (*Bemerkung*: Die Überlappung der Klasse LAPLAS.PF mit FASTpro.MFM, die im Pseudocode zu erkennen ist, wird nicht modelliert, da das einzige überlappende Attribut LAPLAS.PF.Name über eine GIM-Referenz eindeutig aus dem jeweiligen Objekt der Klasse LAPLAS.MF bestimmt werden kann.)

Dieser Konflikt wird gemäß der für GIM notwendigen Verfahrensweise aufgelöst, indem die Klasse FASTpro.MFM in drei Unterklassen mit disjunkten Objektmengen aufgeteilt wird.

Es entstehen die Klassen MFM_l_m (Transporte vom Lager zu einer Maschine), MFM_m_l (Transporte vom einer Maschine zum Lager) und MFM_m_m (Transporte zwischen zwei Maschinen), Abb. 7.1. Da in den beiden erstgenannten Klassen je ein Attribut einen festen Wert erhält, kann es als Klassenattribut definiert werden und aus der Intension der jeweiligen Klasse herausfallen. Die Zugehörigkeit der Objekte zu diesen Klassen wird durch neu entstehende Integritätsbedingungen sichergestellt. Dies sind im einzelnen:

FASTPRO5$_{IIC}$ [PNAS_vor][MFM_l_m]: PNAS_vor is Null(IC$_{10}$, intra-class)

FASTPRO6$_{IIC}$ [PNAS][MFM_l_m]: PNAS is not Null(IC$_{11}$, intra-class)

FASTPRO7$_{IIC}$ [PNAS_nach][MFM_m_l]: PNAS_nach is Null(IC$_{10}$, intra-class)

FASTPRO8$_{IIC}$ [PNAS_vor][MFM_m_l]: PNAS_vor is not Null(IC$_{11}$, intra-class)

FASTPRO9$_{IIC}$ [PNAS_vor, PNAS][MFM_m_m]: (PNAS_vor$*$PNAS) is not Null (IC$_{1n}$, intra-class)

Die Integritätsbedingungen FASTPRO1 und FASTPRO2 ändern sich ebenfalls, da sie jetzt auf drei verschiedenen Klassen definiert sind:

FASTPRO1$_{IIC}$[NAU,NAG][MFM_m_m,MFM_l_m,MFM_m_l]:
((MFM_m_m$\cup$MFM_l_m$\cup$MFM_m_l).NAU+
(MFM_m_m$\cup$MFM_l_m$\cup$MFM_m_l).NAG) is unique(IC$_{mn}$, inter-class)

FASTPRO2$_{IIC}$[NAU,NAG][MFM_m_m]: (NAU+NAG) is not empty(IC$_{1n}$, intra-class)

FASTPRO2$_{IIC}$[NAU,NAG][MFM_l_m]: (NAU+NAG) is not empty(IC$_{1n}$, intra-class)

FASTPRO2$_{IIC}$[NAU,NAG][MFM_m_l]: (NAU+NAG) is not empty(IC$_{1n}$, intra-class)

Die Auflösung dieses Metakonfliktes durch die Aufteilung der Klasse MFM in drei Subklassen spiegelt sich bereits in der aktuellen Version des FACTOTUM wider, indem drei verschiedene SQL-Abfragen drei Dateien erzeugen, die unterschiedliche Formate aufweisen. Bei der Implementation der hier vorgestellten Konzepte in ein föderiertes Datenbanksystem ist zu beachten, daß der Datenbankadapter für FAST/pro für die korrekte Zuordnung von Objekten aus FAST/pro zu diesen drei Klassen zu sorgen hat.

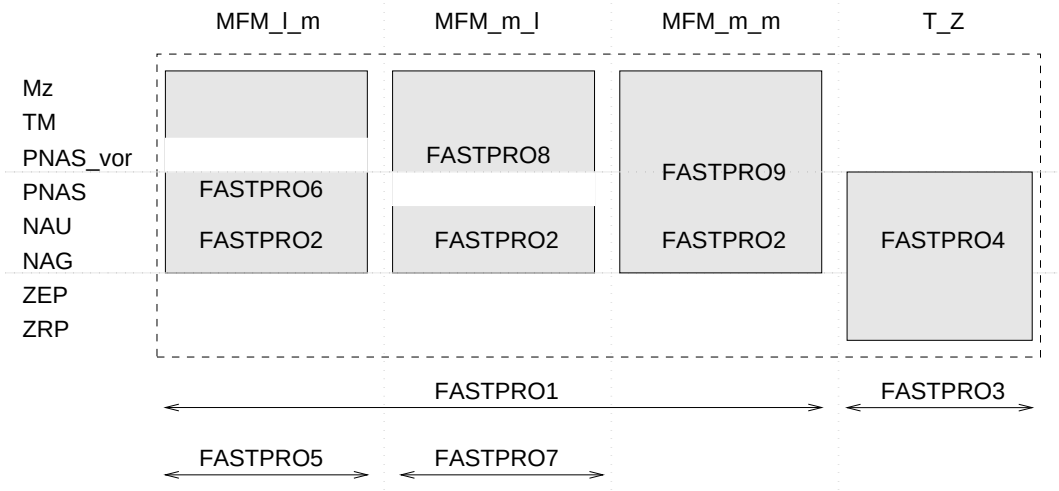

Abbildung 7.1: GIM-Schema FAST/pro nach Auflösung von Metakonflikten

7.1.2 Auflösung von Attributkonflikten

Attributkonflikte treten auf, wenn zwei semantisch gleiche Attribute Informationen in verschiedenen Formaten speichern. Einfache Formen der Attributkonflikte liegen vor, wenn eine umkehrbar eindeutige Abbildung der Werte beider Datentypen vorliegt. In komplizierteren Fällen existiert eine solche Abbildung nicht, wobei der häufigste Fall die Abspeicherung von Daten unterschiedlicher Genauigkeit (bei `real`-Zahlen Länge der Mantisse, bei Strings die Stringlänge) ist. Hier muß bei der Komposition der lokalen Integritätsbedingungen (IB) entschieden werden, ob der „ungenauere“ Wert (die stärkere IB, konjuktive Komposition) oder der „genauere“ Wert (die schwächere IB, disjunktive Komposition) in das föderierte Schema aufgenommen werden soll. Eine mögliche Lösung ist es, im föderierten Schema *zwei* Attribute einzuführen, von denen eines Daten des ungenaueren Datentyps hält, während das zweite Attribut die *zusätzlichen* Informationen (Stellen, Zeichen usw.) abspeichert. So kann sowohl beim Lesen als auch beim Schreiben Konsistenz erreicht werden.

Ein Beispiel im FACTOTUM für den ersten Fall (umkehrbar eindeutige Abbildung) ist die Attributgleichheit

$$\mathsf{FASTpro.MFM.Mz} \Longleftrightarrow \mathsf{LAPLAS.MF.Mz}$$

Das linke Attribut ist ein Real-Wert, der allerdings nur ganzzahlige Werte (Stückzahlen) abspeichert. Das rechte Attribut ist vom Typ Integer. Hier kann durch eine einfache Typwandlung `real`$\Longleftrightarrow$`integer` Abhilfe geschaffen werden. Eine zusätzliche Integritätsbedingung entsteht dadurch nicht.

Der zweite Fall (keine eineindeutige Abbildung) tritt bei der Attributüberlappung

$$(\text{LAPLAS.MF.MFS} \ , \ \text{LAPLAS.MF.MFZ}) \Longleftrightarrow \text{DUMAS.Tr.Entf}$$

auf. In den linken Attributen werden Anfangs- und Endpunkt eines Transports gehalten, im rechten Attribut nur die Entfernung zwischen diesen Punkten. Hier wird eine Abspeicherung der Attribute MFS und MFZ vorgeschlagen, da sich die Entfernung Entf aus diesen beiden Werten berechnen läßt. Zur Erhaltung der Konsistenz der Daten muß dann im föderierten Datenbanksystem ein *update* des Attributs Entf durch die Komponente DUMAS verboten werden.

Ein weiterer Attributkonflikt ergibt sich zwischen Attributen der Klassen aus FAST/pro, LAPLAS und DUMAS, die Materialflüsse beschreiben. Hier werden Maschinennamen in FAST/pro als Zahlenwerte abgespeichert (Attribute PNAS und PNAS_vor), während die semantisch gleichen Attribute in LAPLAS und DUMAS vom Typ `string` sind. Hier ist eine Umwandlung `integer`→`string` über eine `STR`-Funktion unproblematisch, während für die Rückrichtung sichergestellt werden muß, daß für Objekte, die sowohl in LAPLAS und DUMAS wie auch in FAST/pro existieren, nur Attributwerte verwendet werden, die sich verlustfrei in numerische Werte umwandeln lassen.

7.1.3 Intensionale und Extensionale Konflikte

Die Auflösung intensionaler und extensionaler Konflikte findet durch entsprechende Dekomposition der Klassen statt. Für die Bestimmung der neu entstehenden Klassen wird der in [SS96a] beschriebene Algorithmus verwendet, der ein graphisch vereinfachtes GIM-Schema benutzt, das für die Beschreibung der Überlappungsbereiche ausreicht.

Aufgrund der Größe der Komponentenschemata wird keine vollständige graphische Repräsentation der Klassenstruktur angestrebt. Vielmehr erfolgt eine Beschränkung auf die tatsächlichen Überlappungsbereiche. Mehrere getrennte Diagramme können verwendet werden, wenn sich Mengen von Klassen finden lassen, deren *Intensionen und Extensionen disjunkt* sind. Eine Möglichkeit, solche Mengen zu finden, bietet sich bei Aufstellung der überlappenden Klassen (s. oben) durch paarweisen Vergleich aller Überlappungen auf gleichlautende Klassennamen und Aufstellung entsprechender Äquivalenzklassen. Die für FACTOTUM relevante Aufteilung der Überlappungsbereiche in einzelne GIM-Diagramme ergibt sich daraus wie folgt:

1.Diagramm: FASTpro.MFM_m_m, FASTpro.MFM_m_l, FASTpro.MFM_l_m, LAPLAS.MF, DUMAS.Tr

2.Diagramm: LAPLAS.PF, DUMAS.MP, AutoCAD.MA
3.Diagramm: LAPLAS.TM, DUMAS.TMB

Außerdem überlappen sich die Klassen FASTpro.T_Z und TAYLOR.T_Z. Dies stellt jedoch keine Besonderheit dar, da diese Klassen identisch sind (TAYLOR liest die von FAST/pro erzeugte Datei ohne Änderungen über einen Importfilter ein). Auch der Austausch von Hallengrundrissen zwischen AutoCAD und LAPLAS sowie der Export von Dateien im HPGL-Format durch LAPLAS für TAYLOR könnte modelliert werden. Da hier jedoch keinerlei Besonderheiten auftreten und keine Daten aus Dateien gelesen werden müssen, sondern lediglich Dateien kopiert werden, wurde auf eine Darstellung verzichtet.

Die jeweiligen Überlappungszonen, die entstehenden GIM-Diagramme und eine anschauliche Darstellung der jeweiligen extensionalen Überlappungen werden in Abb. 7.2, S. 79, dargestellt.

Aufgrund der Größe der einzelnen Komponentenschemata und der Größe und Komplexität des integrierten Schemas wird eine graphische Darstellung des gesamten Schemas nicht als zweckmäßig betrachtet. Eine gedankliche Zusammenfassung aller Überlappungsbereiche und der nicht überlappenden Teile der Komponentenschemata stößt jedoch nicht auf besondere Schwierigkeiten.

Für die Integritätsbedingungen gilt folgendes:
Jene Integritätsbedingungen der Komponentenschemata, die Schlüsselattribute definieren, werden entsprechend der neuen Klassenaufteilung u.U. zu *inter-class*-Bedingungen und ändern sich sonst nicht.

Bedingungen, die Referenzattribute betreffen, ändern sich ebenfalls nicht. Integritätsbedingungen, die auf zwei in Typkonflikt stehenden semantisch gleichen Attributen definiert sind, müssen geeignet zusammengefaßt werden. Letzteres Problem ist bisher nicht allgemein gelöst. Es muß von Fall zu Fall unterschieden werden, auf welche Art und Weise aus den lokalen Integritätsbedingungen eine neue Bedingung abgeleitet werden kann. Dabei können Probleme bei der Behandlung von *updates* auftreten.

Die in den Diagrammen erkennbaren Namen für die überlappenden Intensionen der Komponentenschemata (Attribute im föderierten Schema) sind frei gewählt und geben die Bedeutung des jeweiligen Attributs bzw. der Gruppe von Attributen wieder. Die Zuordnung von Attributen der Komponentenschema-Klassen zu diesen Attributen des föderierten Schemas zeigen die Tabellen 7.1 bis 7.3.

Bemerkungen zu Tabelle 7.1, S. 80:
Die Attribute PF.Name und TM.Name beziehen sich auf Klassen, die durch GIM-Referenzen mit der Klasse MF verbunden sind. Daher ist eine eindeutige Bestimmung der Attributwerte möglich.
Das föderierte Attribut Entfernung wird im Schema MF durch *zwei* Attribute abgebildet. Da das Attribut Tr.Entf weniger Informationen enthält als diese beiden

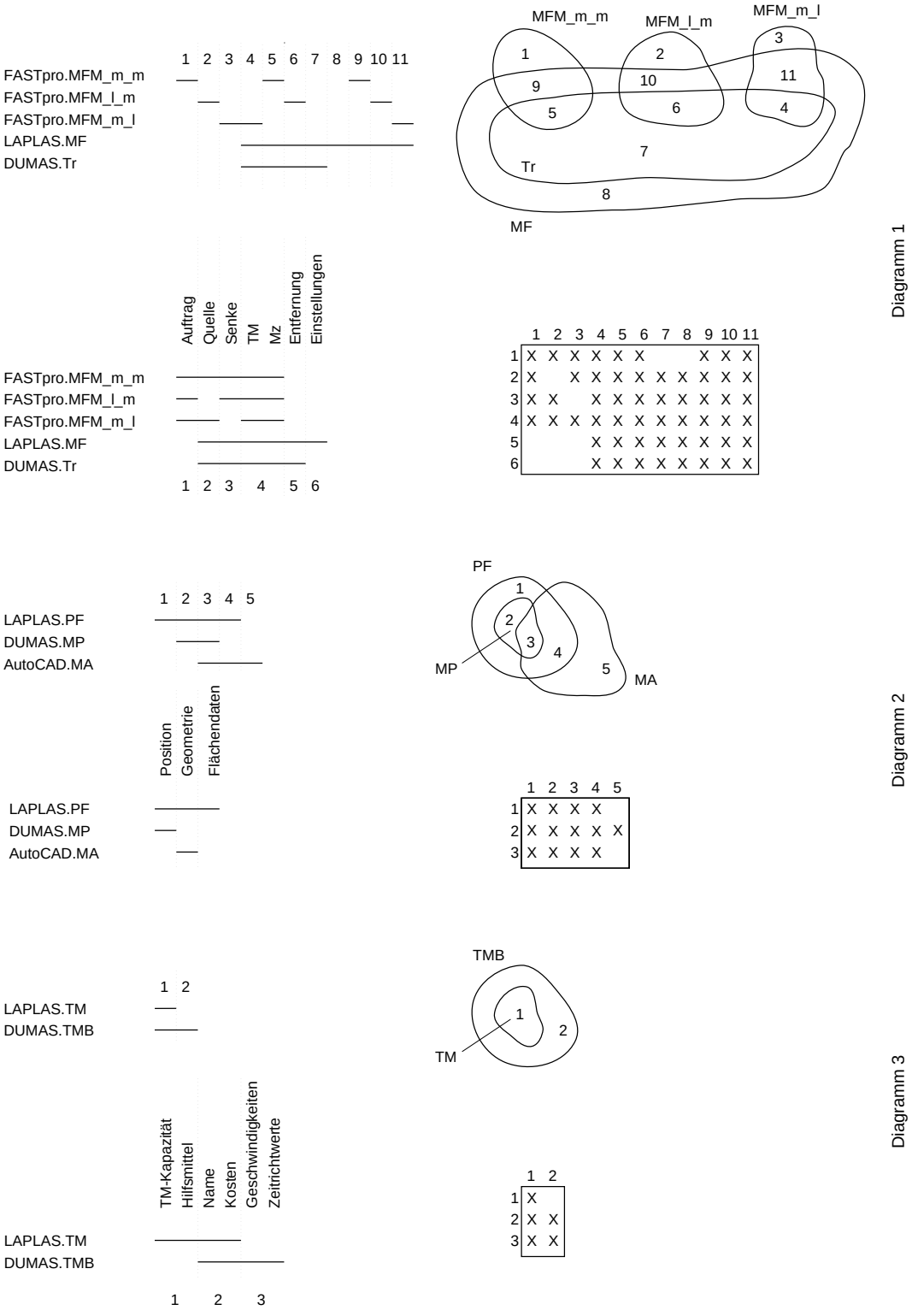

Abbildung 7.2: Ableitung des integrierten Schemas

	MFM_m_m	MFM_l_m	MFM_m_l	MF	Tr
Auftrag	NAU,NAG	NAU,NAG	NAU,NAG		
Quelle	PNAS_vor	`"Ein"`	PNAS_vor	PF.Name	QName
Senke	PNAS	PNAS	`"Aus"`	Zielfl	ZName
TM	TM	TM	TM	TM.Name	TM_name
Mz	Mz	Mz	Mz	Mz	Mz
Entfernung				MFS,MFZ	Entf
Einstellg.				Einst	

Tabelle 7.1: Attributzuordnung *Materialflüsse*

Attribute, müssen Vorkehrungen für die Konsistenz getroffen werden (Verbot eines *update* von Tr.Entf durch das System DUMAS).
Bei den „Attributen“ `"Ein"` und `"Aus"` handelt es sich um Konstanten des Typs `string`, die für Objekte der jeweiligen Klassen in die zugehörigen föderierten Attribute geschrieben werden müssen. Das muß im Datenbankadapter für FAST/pro berücksichtigt werden.

	PF	MP	AM
Position	Name, Quelle, Senke	Name, Quelle, Senke	
Geometrie	DXF		DXF
Flächendaten	Groesse, koor, rec, Fix, Darst		

Tabelle 7.2: Attributzuordnung *Maschinen*

Bemerkungen zu Tabelle 7.2:
Geometriedaten einer Fläche sind als Attribute des Typs DXF modelliert (Abschnitte 6.2, 6.5). Die Klasse LAPLAS.PF enthält zwar semantisch gleiche Daten auch in anderer Form (Attribute koor, rec). Allerdings existiert in LAPLAS eine Importfunktion, die DXF-Daten in das interne Format umwandelt. Eine Analyse und die Ableitung von Informationen zum *parsing* dieser DXF-Dateien zur Nachbildung dieses Imports im FDBS erschien daher nicht sinnvoll.

	TM	TMB
TM-Kapazität	Kap	
Transport-Hilfsmittel	Hilf	
Name	Name	TM_name
Kosten	Fix_k, Var_k	Fixk, Vark
Geschwindigkeiten		Geschw
Zeitrichtwerte		Zeiten

Tabelle 7.3: Attributzuordnung *Transportmittel*

Die Behandlung von Attributkonflikten wurde bereits in Abschnitt 7.1.2 angesprochen. Es treten keine weiteren Besonderheiten auf. Daher wurde auf die Aufstellung sämtlicher Integritätsbedingungen verzichtet.

7.1.4 Globale Integritätsbedingungen

Globale Integritätsbedingungen lassen ein Hinzufügen neuer Informationen zur föderierten Datenbank zu (semantische Anreicherung). So können Bedingungen formuliert werden, die die Zusammenarbeit verschiedener Komponentensysteme regeln. Ein wichtiger Einsatzbereich solcher Bedingungen ist die Beseitigung von Inkonsistenzen zwischen Komponenten.

Im System FACTOTUM tritt eine Inkonsistenz auf, da in den Klassen LAPLAS.TM und DUMAS.TMB (Transportmittel) eine doppelte Abspeicherung semantisch gleicher Daten erfolgt. Das ermöglicht es, in LAPLAS Transportmittel zu erstellen, die in DUMAS nicht definiert sind. Dies führt zu Fehlern bei der Übertragung eines Projektes von LAPLAS nach DUMAS und muß deshalb ausgeschlossen werden. Das geschieht folgendermaßen über Integritätsbedingungen:

$FS1_{GIC}$ [TM.Name, TMB.TM_name][TM,TMB]:
TM.Name is not Null $\Rightarrow \exists$ Objekt$\in$TMB(TM.Name=TMB.TM_name)
(IC_{mn}, inter-class)

Für jedes Transportmittel in TM muß also ein Objekt in TMB existieren, das den gleichen Namen trägt.

Weitere globale Integritätsbedingungen könnten für die Existenz verschiedener Grafiken (DXF, HPGL) aufgestellt werden. Dies ist aber trivial und soll deshalb hier nicht ausgeführt werden.

In den Fällen, in denen Integritätsbedingungen für gleiche Intensionen auf teilweise überlappenden Extensionen gelten, muß für den überlappenden Teil eine gemeinsame Integritätsbedingung gefunden werden. Dies kann jedoch zu Problemen bei der Implementation führen (vgl. [CHJ+96]).

7.2 Ableitung Externer Schemata

Nach der Aufstellung eines föderierten Schemas soll nun ein aussagekräftiges externes Schema abgeleitet werden, das die nicht geänderten Teile der Komponentenschemata sowie die integrierten Überlappungsbereiche darstellt.

Dazu muß zunächst die dekomponierte Klassenstruktur der GIM-Schemata wieder geeignet komponiert werden, um zu einer Klassenhierarchie zu kommen, die eine konsistente und redundanzfreie Abspeicherung aller Datenobjekte gestattet. Dazu wird ein Algorithmus ähnlich dem in [SS96a] beschriebenen benutzt.

Als Zielmodell für den Algorithmus kommt ein objektorientiertes Modell mit Rollenkonzept zum Einsatz. Die Vorgehensweise für diese Komposition soll aus Platzgründen nicht erläutert werden. Die entstehenden Schemata für die drei nicht-trivialen Überlappungsbereiche zeigt Abb. 7.3, S. 83. Die Rollen, die einzelne Objekte einnehmen können, ergeben sich aus der extensionalen Ausdehnung der einzelnen Klassen. Aus den angegebenen Tabellen der Extensionen lassen sich für ein gegebenes Objekt die möglichen Rollen ablesen.

7.2.1 Das komplette Externe Schema

Da die lokalen Schemata im Entity-Relationship-Modell modelliert wurden, liegt es nahe, auch das externe Schema in einem ähnlichen Modell darzustellen, um Vergleichbarkeit zu ermöglichen. Im erweiterten ER-Modell (EER) der TU Braunschweig [HS95, EGH+92] existiert ein Konzept zur Spezialisierung und Generalisierung, mit dem die aus dem integrierten Schema abgeleiteten Klassenhierarchien abgebildet werden können. Eine Darstellung des externen Schemas unter Verwendung dieses Spezialisierungskonzeptes zeigt Abb. 7.4, S. 84. In dieser Darstellung ist die Kopplung der Komponenten-Datenbanksysteme mit Hilfe der entwickelten neuen Klassen gut zu erkennen, was den wesentlichen Vorteil dieses Datenmodells darstellt. Für ein real existierendes föderiertes Datenbanksystem ist jedoch die Implementation eines objektorientierten Modells eher zu erwarten. Anhand des hier abgeleiteten GIM-Schemas und der hier sowie in der Literatur zu GIM beschriebenen Algorithmen läßt sich allerdings ein externes Schema in einem beliebigen Datenmodell mit Spezialisierungskonzept aufstellen. Diese Schemaableitung ist hier nicht ausgeführt, da sich ein solches Schema nur schlecht mit den lokalen Schemata der Komponenten vergleichen läßt. Auch ist eine solche Ableitung recht trivial, und es fehlt ein echtes föderiertes Datenbanksystem, durch das ein konkretes Datenmodell als Entwicklungsziel vorgegeben würde.

	4	5	6	9	10	11	1	3	2	7	8
4	X	X	X	X	X	X	X	X	X	X	X
1	X	X	X	X	X	X	X	X	X		
2	X	X	X	X	X	X	X	X		X	X
3	X	X	X	X	X	X	X		X	X	X
5	X	X	X	X	X	X				X	X
6	X	X	X	X	X	X				X	X

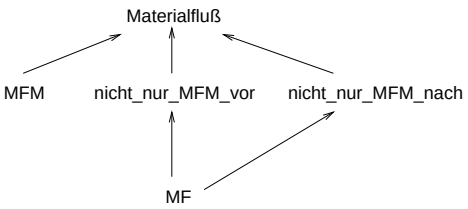

Diagramm 1

	Extensionen	Intensionen	Superklasse	Bezeichnung
A	1-11	4	-	Materialfluß
B	1-6, 9-11	1, 4	A	MFM
C	1, 3-11	2, 4	A	nicht_nur_MFM_vor
D	1-2, 4-11	3, 4	A	nicht_nur_MFM_nach
E	4-11	2-6	A, C, D	MF

	1	2	3	4	5
2	X	X	X	X	X
1	X	X	X	X	
3	X	X	X	X	

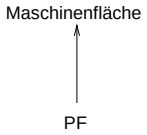

Diagramm 2

	Extensionen	Intensionen	Superklasse	Bezeichnung
A	1-5	2	-	Maschinenfläche
B	1-4	1-3	A	PF

	1	2
2	X	X
3	X	X
1	X	

Diagramm 3

	Extensionen	Intensionen	Superklasse	Bezeichnung
A	1-2	2-3	-	TMB
B	1	1-3	A	TM

Abbildung 7.3: Externe Teilschemata für die Überlappungsbereiche

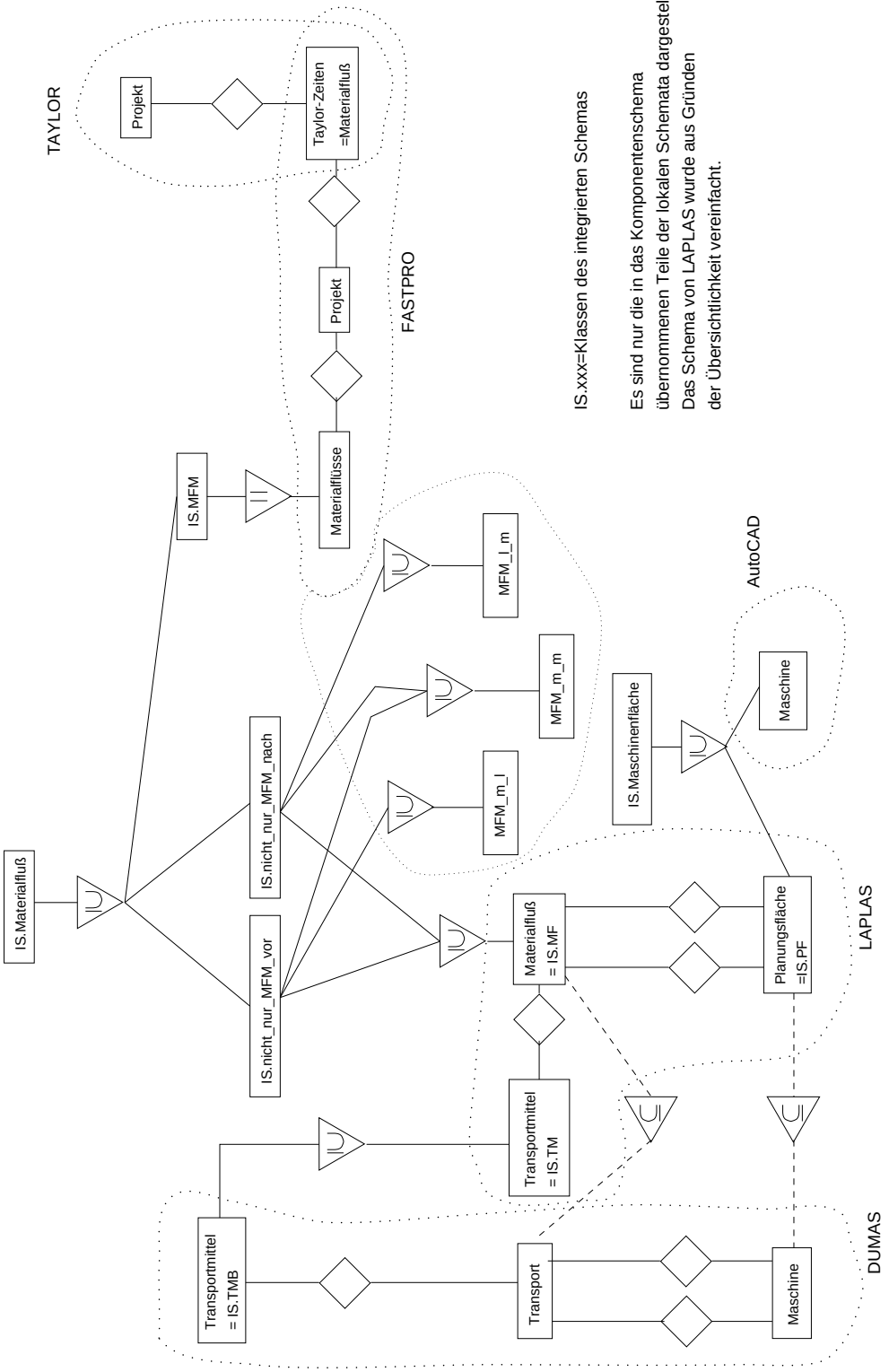

Abbildung 7.4: Gesamtes externes Schema im EER-Modell

7.3 Schlußfolgerungen

Die in den vorhergehenden Kapiteln abgeleiteten Informationen (Strukturinformationen der Dateien, Datenmodelle und Datentypen, Integritätsbedingungen, externes Schema) bilden zusammen mit den Dateistrukturanalysen im Anhang eine komplette Beschreibung des Systems FACTOTUM und der Zusammenarbeit der einzelnen Komponenten. Die bei der Analyse gefundenen Inkonsistenzen und Probleme bei der Zusammenarbeit der einzelnen Komponenten wurden aufgeführt und Möglichkeiten zur Lösung vorgeschlagen. Aufgrund der gegebenen Informationen ist es bei Vorhandensein eines föderierten Datenbanksystems möglich, Datenbankadapter für die Anbindung der Komponenten an ein FDBS zu erstellen, dessen Datenbankschema wiederum aus dem hier vorgestellten integrierten Schema abgeleitet werden kann. Das angegebene externe Schema im EER-Modell dient zur anschaulichen Darstellung der erreichten Ergebnisse und der Unterschiede zwischen dem föderierten Schema und einer einfachen Zusammenfassung der lokalen Schemata. Bei Verwendung eines föderierten Datenbanksystems können Probleme des FACTOTUM gelöst, bestimmte Arbeitsabläufe vereinfacht und eine einheitliche Sicht auf alle in den Komponenten enthaltenen Daten angeboten werden.

Allerdings sind alle Integrationsbemühungen im Rahmen der Genauigkeit der verwendeten Modelle zu sehen. Die Entwicklung des vorliegenden Modells erfolgte über eine Vielzahl von Schritten. Die wichtigsten Etappen waren:

1. Analyse der zugrundeliegenden Datenhaltungskonzepte
2. Untersuchung der verwendeten Dateien, Import- und Exportfunktionen
3. Ableitung lokaler Schemata und Identifikation verwendeter Attribute und ihrer Datentypen
4. Ableitung transformierter Komponentenschemata und Integritätsbedingungen
5. Aufstellung des föderierten Schemas und Transformation der Integritätsbedingungen
6. Hinzufügen zusätzlicher Informationen über das System
7. Ableitung eines externen Schemas

Der Umfang des bearbeiteten Datenmaterials in jedem dieser Schritte, die Komplexität gegenseitiger Abhängigkeiten oder die Vielzahl von Nebenbedingungen stellen Quellen für Fehler dar, die sich teilweise aus frühen Stadien der Systemanalyse in alle weiteren Etappen fortpflanzen und im ungünstigsten Fall den Wert

des entstandenen Datenbankschemas erheblich mindern können. Das Fehlen eines einsatzfähigen föderierten Datenbanksystem erschwert die Angabe vollständiger Informationen für die Integration weiter. Es bleibt festzustellen, daß der Versuch einer Föderierung selbst eines recht kleinen Systems wie dem FACTOTUM eine Aufgabe darstellt, die sehr umfangreich und fehleranfällig ist. Sorgfältig ausgearbeitete, vollständige und stabile Konzepte und Algorithmen zur Datenbankintegration sind daher unabdingbare Voraussetzung für eine praxisorientierte Entwicklung föderierter Datenbanksysteme.

Im vorliegenden Fall bedeutet insbesondere das Fehlen fester lokaler Datenbankschemata in den dateibasierten Datenhaltungssystemen eine erhebliche Komplizierung des Integrationsprozesses. Hier könnte eine teilweise Automatisierung der Erkennung von Datenelementen in Dateien oder der Schemaableitung von großem Nutzen sein. Daher erscheint eine Untersuchung der Möglichkeiten zur Entwicklung von Werkzeugen für den Datenbankentwurf sehr sinnvoll und wünschenswert. Auch die Entwicklung von Algorithmen und Regeln zur Aufstellung und Transformation von Integritätsbedingungen wird als wesentliche Voraussetzung für einen erfolgreichen Integrationsprozeß erachtet. Die Vorteile einer Nutzung föderierter Datenbanksysteme für die Integration heterogener Systeme, deren Weiternutzung erwünscht ist, liegen jedoch auf der Hand und sollten als Motivation für weitere Forschung auf diesem Gebiet dienen.

Kapitel 8

Möglichkeiten zur Realisierung

In Kapitel 7 wurde mit Hilfe der Integrationsmethodik des GIM versucht, ein möglichst vollständiges integriertes Datenbankschema für ein föderiertes Datenbanksystem über den Komponenten des Produktionsplanungssystems FACTOTUM zu entwickeln. In diesem Kapitel soll diskutiert werden, welche Möglichkeiten zur praktischen Durchführung eines solchen Vorhabens bestehen, und es soll der Stand der Entwicklung zum jetzigen Zeitpunkt dargestellt werden.

8.1 Architektur eines Föderierten Datenbanksystems

Nach den allgemeinen Ausführungen zu Anforderungen an ein föderiertes Datenbanksystem in Kapitel 2 soll nun eine Architektur für ein solches System vorgestellt werden, wie sie als Grundlage für das in Entwicklung befindliche Produkt OpenDM/Efendi verwendet wird.

In OpenDM/Efendi besteht das föderierte Datenbank-Management-System aus einem Kern, der die Datenbankfunktionalität bereitstellt, sowie einer Anzahl von Adaptern. Diese Adapter sind Datenbankadapter zur Anpassung an verschiedene Komponenten-DBS und externe Adapter zur Anbindung des FDBS an Applikationen (*front ends*). Der Datenbankkern greift auf eine globale Datenbank zu, in der die mit Hilfe des föderierten Schemas modellierten Daten gehalten werden.

Diese Architektur (Abb. 8.1) zeichnet sich durch eine gute Erweiterbarkeit aus, ist also in der Lage, verschiedenste Datenmodelle für die Komponenten-DBS zu unterstützen. Dazu ist jeweils die Entwicklung geeigneter Datenbank-Adapter nötig.

Für die Zwecke einer Integration des FACTOTUM scheint diese Architektur aus den genannten Gründen gut geeignet zu sein.

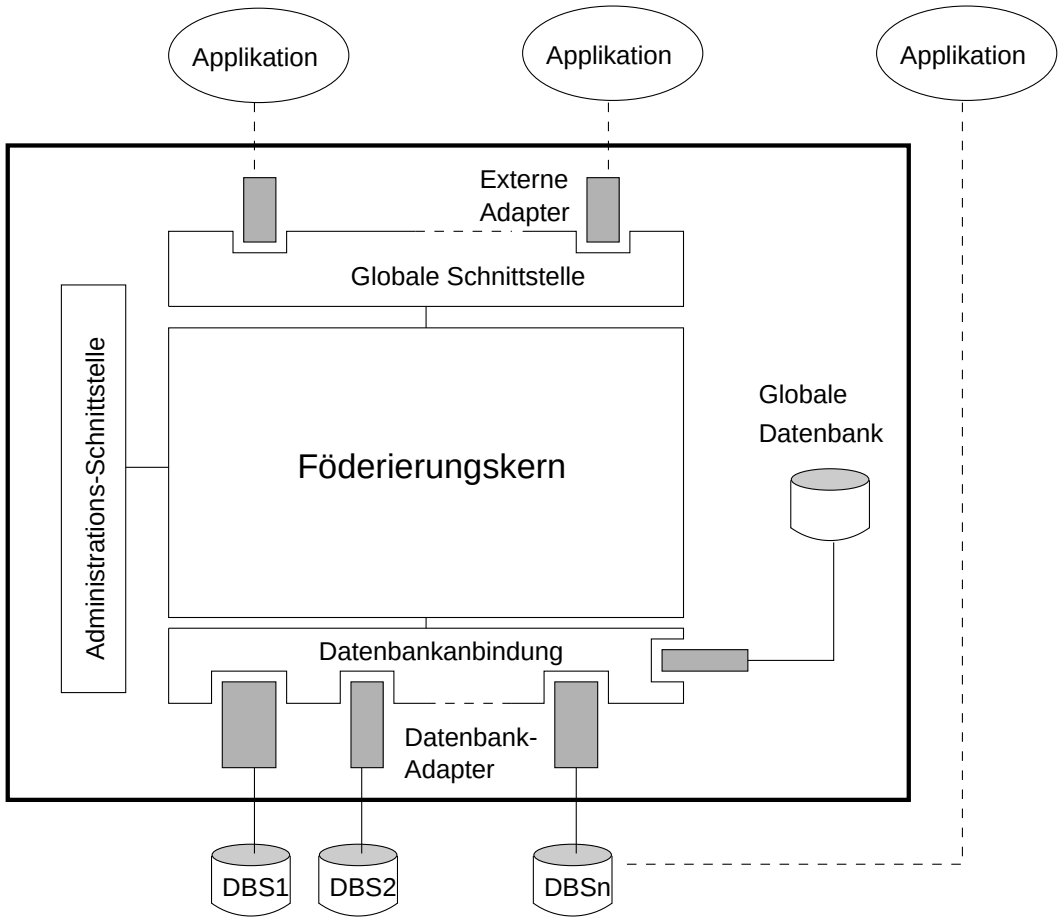

Abbildung 8.1: Architektur eines Föderierten Datenbanksystems, nach [Rad94]

8.2 Das föderierte Datenbanksystem OpenDM/Efendi

Das föderierte Datenbanksystem *OpenDM/Efendi* ist eine Entwicklung des C-LAB Paderborn[1], das auf dem vom gleichen Institut entwickelten Datenbank-Werkzeug *OpenDM* aufbaut. Ein Prototyp des Systems mit einer Anbindung an relationale und objektorientierte Komponentendatenbanken wurde bereits vorgestellt, und eine lauffähige Version mit der Möglichkeit eigener Adapterprogrammierung ist in Vorbereitung.

[1] C-LAB, vormals **cadlab**, ist ein gemeinsames Forschungs- und Entwicklungsunternehmen der Universität-GH Paderborn und der Siemens Nixdorf Informationssysteme AG

8.2.1 Das Datenbank-Werkzeug OpenDM

Das von C-LAB entwickelte System OpenDM (*Open Database Middleware*) stellt einen objektorientierten Standard für die Datenbankintegration dar, der sowohl auf relationale Datenbanken als auch auf objektorientierte Datenbanken oder Filesysteme (und die auf ihnen aufbauenden dateibasierten Datenhaltungssysteme) aufsetzen kann. OpenDM bildet den Kern einer Produktfamilie von Datenbanksystemen, die verschiedene Stufen der Datenintegration anbieten. Zur Zeit existiert eine Konfiguration des OpenDM mit dem Namen *OpenDM/ODMG* [Rad95], die in der Lage ist, für die genannten Komponentensysteme eine einheitliche objektorientierte Schnittstelle im ODMG-Datenmodell bereitzustellen (Abb.8.2, nach [Rad95]). Dabei kann jeweils nur ein System integriert werden.

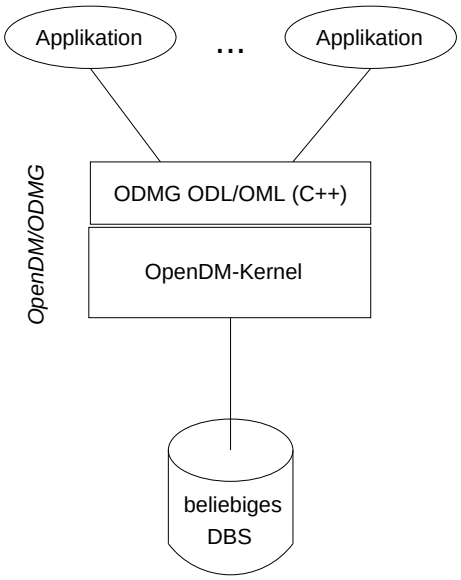

Abbildung 8.2: Ein einheitliches ODMG-Interface mit OpenDM/ODMG

Mit dem föderierten Datenbanksystem OpenDM/Efendi befindet sich ein weiteres Produkt der OpenDM-Familie in der Entwicklung, das im Gegensatz zu OpenDM/ODMG in der Lage ist, mehrere Komponenten gleichzeitig zu integrieren und dabei heterogene Datenmodelle und Schemata zuzulassen. Dabei kann auch Software mit auf Filesystemen aufbauenden Datenhaltungskonzepten als Komponenten-Datenbanksystem integriert werden. Efendi unterstützt die volle Funktionalität eines FDBS, ist nach der in Kapitel 8.1 beschriebenen Architektur aufgebaut und wurde objektorientiert erstellt. Datenbankadapter zu unterschiedlichsten Komponenten können auf einfache Weise in C++ implementiert werden, wobei wichtige Standardschnittstellen bereits vorliegen. Das Interface von Efendi zu Applikationen implementiert eine erweiterte Version des ODMG-Standards, dem Sprachelemente für die Schemaintegration hinzugefügt wurden.

Weitere Eigenschaften des OpenDM/Efendi sind: Wahrung der Autonomie der Komponentensysteme, Migrationsfunktionen für Datenbewegung und Datenreplikation zwischen Komponenten, Performance-Optimierung durch *caching* und modularer Aufbau.

8.3 Anbindung DOS-basierter Systeme an ein FDBS

Wie bereits in Kapitel 3.1, S. 19, ausgeführt, stößt die Anbindung MS-DOS-basierter Systeme (oder anderer Filesysteme mit unzureichender Parallelisierung und Zugriffsregelung) auf Probleme bei der Entwicklung föderierter Datenbanklösungen. Die Überwindung dieser Schwierigkeiten ist z.Z. Gegenstand von Untersuchungen auch am Institut für Technische Informationssysteme.

Eine Architektur für die Kopplung dateibasierter *MS-DOS*-Applikationen an ein föderiertes Datenbanksystem wurde von M. Höding vorgeschlagen (Abb. 8.3).

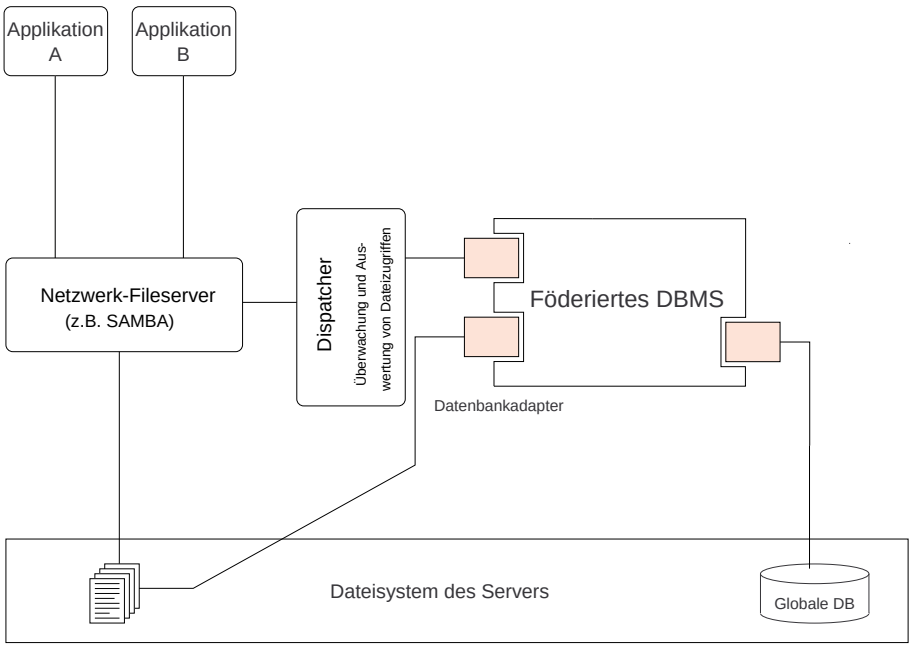

Abbildung 8.3: Kopplung dateibasierter Systeme an FDBS

Das Grundproblem bei der Entwicklung geeigneter Datenbankadapter für dateibasierte Systeme stellt der Zugriff auf Daten der Komponentensysteme dar. *MS-DOS*-Applikationen sind *single-task–single-user*–Systeme, die aufgrund der Architektur des *DOS*-Dateiprotokolls weder mit einem Dateizugriff durch andere Applikationen während ihrer Laufzeit rechnen, noch solche Zugriffe geeignet

behandeln können. Außerdem ist ein FDBS darauf angewiesen, über alle Dateizugriffe durch die Komponentensysteme informiert zu werden, um die gültigen Datenobjekte jederzeit identifizieren und entsprechend reagieren zu können. Nach den bisherigen Untersuchungen ist die Implementation solcher Funktionen nur auf UNIX-basierten Systemen möglich. Eine Software zum „Abfangen" von Dateizugriffen ist auf ein vollständig (also präemptiv) parallel arbeitendes Betriebssystem angewiesen, was die Verwendung von MS-DOS, *Windows 3.1* und *Windows 95* ausschließt. Neben UNIX könnte das Betriebssystem *Windows NT* für eine solche Aufgabe ebenfalls geeignet sein; dies wurde allerdings bisher nicht untersucht.

8.3.1 SAMBA als Fileserver für die Überwachung von Dateizugriffen

Eine Lösung des Problems der gemeinsamen Zugriffe auf Dateien und der Überwachung dieser Zugriffe könnte das File- und Printserver-System **SAMBA** darstellen, das als *Public-Domain*-Software unter der *GNU Public License* vorliegt. SAMBA implementiert das von *Windows for Workgroups, Windows 95, Windows NT* und *OS/2* benutzte SMB-Protokoll (*Session Message Block,* [SMB88]) zum Zugriff auf Netzlaufwerke. Da der Quelltext unter der GNU-Lizenz offenliegt, ist eine Anpassung an die Zwecke der FDBS-Anbindung möglich. Eine lauffähige Version von SAMBA und ein PC zum Test dieser Vorgehensweise sind am ITI erst seit kurzem verfügbar. Aus ersten Testergebnissen lassen sich jedoch bereits Rückschlüsse auf die Durchführbarkeit der geplanten Vorgehensweise ziehen. Die bisher erzielten Ergebnisse sollen kurz vorgestellt werden.

Zunächst ist zu bemerken, daß mit dem vorgestellten Konzept nur die Netzanbindung eines PC auf Grundlage des SMB-Protokolls durchgeführt werden kann. Daher ist die Verwendung von mindestens *Windows 3.11* auf dem FACTOTUM-PC notwendig. Alle FACTOTUM-Anwendungen müssen dann in einer virtuellen *MS-DOS-Shell* („MS-DOS-Eingabeaufforderung") des *Windows* betrieben werden, um die SMB-Netzanbindung benutzen zu können. Das führt allerdings nicht zu Problemen, da Tests ergeben haben, daß alle FACTOTUM-Komponenten entweder als Windows-Versionen verfügbar sind oder stabil in einer *DOS-Shell* arbeiten.

Die zweite Voraussetzung ist, daß alle FACTOTUM-Komponenten auf dem Netzlaufwerk installiert sind oder wenigstens alle relevanten Daten dort halten. Dies ist die Grundvoraussetzung für ein Abfangen von Dateizugriffen durch das föderierte DBMS. Auch hier wurde durch Tests bestätigt, daß abgesehen von leichten Geschwindigkeitseinbußen der Komponenten durch die Server-Benutzung keine Probleme mit dieser Verfahrensweise auftreten. Alle FACTOTUM-Komponenten können problemlos auf einem Netzlaufwerk arbeiten.

Nach korrekter Installation des Systems in der beschriebenen Weise kann ein SAMBA-Fileserver-Prozeß, der auf einem UNIX-System läuft, alle Dateizugriffe durch DOS-Prozesse des PC abfangen. Dabei ist zu beachten, daß SAMBA für praktisch alle wichtigen UNIX-Versionen verfügbar ist, so daß ein nahezu beliebiger UNIX-Computer als Fileserver eingesetzt werden kann. In der SAMBA-Originalversion wird nun zunächst eine Datei angelegt, in die alle Zugriffsinformationen eines bestimmten *client* (z.B. des PC) sofort nach ihrem Auftreten geschrieben werden. Die Anzahl und Genauigkeit der abgelegten Informationen kann dabei über einen sogenannten *debug-level* gesteuert werden und reicht von der ausschließlichen Ausgabe von Systemfehlermeldungen (*debug-level* 0) bis zur ausführlichen Auflistung aller Dateioperationen (öffnen, schließen, Block lesen, Block schreiben usw.) mit Parametern im *debug-level* 5. In einer ersten Testserie wurde diese Datei permanent ausgewertet und so Informationen über Dateizugriffe gewonnen.

Für den Betrieb eines föderierten Datenbanksystems reicht es aus, wenn das Datenbank-Management-System über das Öffnen und Schließen von Dateien und den Zugriffsmodus (nur Lesen - Lesen und Schreiben) informiert wird. Diese Aufgabe ist mit Hilfe der SAMBA-Logdatei recht einfach zu lösen, da diese Dateioperationen durch entsprechende Schlüsselwörter angezeigt werden.

Zur Verdeutlichung soll ein kurzer Auszug aus einem SAMBA-*logfile* (Dateiname `log.sigmafdb`) gezeigt werden:

```
07/17/96 12:51:38 sigmafdb opened file ./factotum/data.dat read=Yes write=Yes (numopen=1 fnum=1)
07/17/96 12:51:38 sigmafdb closed file ./factotum/data.dat (numopen=0)
```

Die zwischen den *open*- und *close*-Operationen liegenden Schreib- oder Lesezugriffe wurden aufgrund des hier gewählten *debug-level* (s.o.) nicht in das *logfile* aufgenommen. In den Fällen, in denen eine PC-Software eine Datei datensatzweise und nicht vollständig schreibt und liest, ist eine Auswertung ebenfalls möglich, wenn auch hier z.Z. noch Schwierigkeiten mit der vorliegenden SAMBA-Version auftreten. Eine gesonderte Auswertung solcher teilweisen Dateiänderungen ist jedoch nur dann nötig, wenn ein Lesen oder Schreiben der gesamten Datei zu nicht mehr akzeptablen *performance*-Einbußen führt, ein Umstand, der im vorliegenden System FACTOTUM aufgrund der relativ kleinen zu verwaltenden Datenmenge nicht gegeben ist.

Da sich der zugreifende PC durch seinen Benutzernamen (hier `sigmafdb`) identifizieren läßt, ist eine Auswertung sehr einfach möglich. So kann bereits durch einen UNIX-Kommandozeilen-Befehl wie

```
tail -f log.sigmafdb | grep "sigmafdb"
```

eine einfache aktuelle Liste der Dateizugriffe erstellt werden, mit deren Hilfe sich leicht eine Auswertung von Operationen zum Dateiöffnen und -schließen vorneh-

men läßt. Für eine Verwendung von SAMBA in Datenbankadaptern eines föderierten Datenbanksystems wäre natürlich die Neuprogrammierung der Abfangroutine unter Verwendung der SAMBA-Quellen notwendig. Dies konnte jedoch aufgrund der späten Verfügbarkeit der erforderlichen Ressourcen und insbesondere aufgrund des Fehlens eines föderierten Datenbanksystems als Entwicklungsziel nicht weiter verfolgt werden. Die Tauglichkeit der vorgestellten Konzepte für die Anbindung dateibasierter Datenhaltungssysteme unter MS-DOS dürfte jedoch mit den bereits durchgeführten Arbeiten bewiesen sein.

8.4 Stand der Entwicklung

Mit der in Abschnitt 8.1 vorgestellten Struktur und dem angebotenen Leistungsumfang stellt OpenDM/Efendi ein System dar, das den Anforderungen der in dieser Arbeit behandelten Thematik der Datenbankintegration dateibasierter Systeme voll gerecht wird. Bei Verfügbarkeit einer kommerziellen Version dieses Produkts ist die Implementation der beschriebenen föderierten Datenbankarchitektur aller Wahrscheinlichkeit nach möglich.

Auch die Anbindung der im FACTOTUM vorhandenen dateibasierten Datenhaltungssysteme unter MS-DOS, die aufgrund des nötigen Investitionsschutzes weiterhin verwendet werden sollen, ist mit Hilfe eines Servers, der das Microsoft-SMB-Protokoll implementiert, möglich, wenn das föderierte Datenbanksystem auf einem UNIX-basierten System läuft. Die dafür notwendigen Konzepte wurden aufgestellt, während allerdings eine Realisierung von Datenbank-Adaptern aus den genannten Gründen noch nicht möglich war. Eine sinnvolle Implementation von Adaptern ist ohnehin erst möglich, wenn eine endgültige Schnittstellen-Spezifikation für die Komponentenseite des föderierten Datenbankkerns vorliegt. Eine Weiterentwicklung und ein praktischer Test der in dieser Arbeit vorgestellten Konzepte hängt darum im wesentlichen von der Entwicklung lauffähiger und verfügbarer föderierter Datenbanksysteme ab.

Kapitel 9

Potentiale der Nutzung von FDBS

Das Produktionsplanungssystem FACTOTUM stellt den Versuch dar, den Prozeß der Fabrikplanung mit all seinen Phasen von der Erfassung von Produktionsdaten bis hin zur Visualisierung der optimierten Fabrikhalle mit ihren Maschinen, Transportmitteln und Materialflüssen mit computerbasierten Werkzeugen zu unterstützen. Dabei kommen Komponentensysteme unterschiedlicher Hersteller zum Einsatz, deren Datenhaltungskonzepte und Schnittstellen sich grundlegend voneinander unterscheiden. Die daraus resultierenden Probleme bei der Arbeit mit dem FACTOTUM (Datenübertragung zwischen Systemen, Konsistenzprobleme) stellen die wesentlichste Einschränkung für die effiziente Nutzung dieses Systems dar.

Hier kann eine Integration der Komponenten, die zu einer einheitlichen Sicht auf die Daten und zur Möglichkeit eines gemeinsamen Datenmanagement führt, entscheidende Vorteile bringen. Dabei sind zwei Stufen der Integration zu berücksichtigen: eine einheitliche Modellierung der Komponentensysteme und die Implementation eines Softwaresystems zur Verwaltung und konsistenten Haltung der Komponentendaten.

9.1 Integration dateibasierter Komponentensysteme

Dateibasierte Datenhaltungssysteme, also Softwareprodukte, die keine Datenbankfunktionalität im herkömmlichen Sinne anbieten, sind primär nicht für eine Integration in ein übergeordnetes Datenbanksystem ausgelegt. Soll eine solche Integration versucht werden, sind geeignete Konzepte zu finden, um die von den Komponenten gespeicherten Daten zu modellieren und zur Laufzeit der Systeme

zu extrahieren bzw. Updates vorzunehmen. Dabei treten im untersuchten System FACTOTUM Schwierigkeiten auf, da es sich bei den Komponenten um Software handelt, die auf dem nicht *multitasking*-fähigen Betriebssystem MS-DOS basiert. So sind spezielle Konzepte für die Integration der Daten erforderlich.

Der Autor ist der Meinung, daß die vorgeschlagene Methode, Dateizugriffe über einen UNIX-basierten Fileserver zu erkennen und auszuwerten, einen vielversprechenden Ansatz für ein solches Vorhaben liefert. Wichtig ist, das jeweils laufende Komponentensystem in seinem Programmablauf möglichst wenig zu behindern (was beispielsweise durch häufiges und langes Sperren von Dateien geschehen würde) und gleichzeitig eine möglichst schnelle Benachrichtigung des übergeordneten Datenbanksystems zu erreichen.

Aufgrund der durch eine FDBS-Nutzung zu erreichenden Vorteile erscheint eine Integration dateibasierter Komponenten, besonders im vorliegenden Falle des FACTOTUM, sinnvoll. Es ist jedoch zu berücksichtigen, daß aufgrund der genannten Schwierigkeiten bei der Überwachung von Dateizugriffen die Verwendung eines „Nicht-*MS-DOS*“-Computers, also eines zweiten Computers wie etwa einer UNIX-*Workstation* notwendig wird. Das verringert erheblich die Transportfähigkeit des Systems, das bisher auf einem Laptop installiert und so leicht zu einem Kunden transportiert werden konnte. Abhilfe könnte hier erst die Verwendung eines Betriebssystems schaffen, das neben entsprechenden Dateizugriffs-Operationen auch die Möglichkeit bietet, MS-DOS-Software ohne Änderungen in Echtzeit ablaufen zu lassen. Hierfür steht bisher nur *Windows NT* zur Verfügung, für das allerdings keine Erkenntnisse hinsichtlich geeigneter Treiber oder Software für die genannten Dateizugriffs-Probleme vorliegen. Außerdem ist nicht abzusehen, ob in Zukunft geeignete FDBS-Software für diese Plattform zur Verfügung stehen wird.

9.2 Auswirkungen einer FDBS-Nutzung auf das System FACTOTUM

Neben der Frage der technischen Realisierung des Datenzugriffs steht bei der Integration die Entscheidung über die Art des zu verwendenden übergeordneten Datenbanksystems. Aufgrund der Heterogenität der Komponentensysteme und der Forderung nach Autonomie und Investitionsschutz (keine Änderung der Komponenten möglich), entsteht der Bedarf nach einem föderierten Datenbanksystem.

Die Integration der FACTOTUM-Komponenten in ein föderiertes Datenbanksystem bietet wesentliche Vorteile. Zunächst wird durch die ohnehin notwendige Schemaintegration ein Datenbankschema erzeugt. Dadurch wird für jedes Komponentensystem ein Schema in einem einheitlichen Datenmodell abgeleitet, wo-

raufhin das föderierte Schema dann eine einheitliche Sicht auf alle Komponentensysteme bietet.

Ist diese Schemaintegration abgeschlossen, können im föderierten Schema dann globale Integritätsbedingungen festgelegt werden, die bestehende Inkonsistenzen der Komponentensysteme beheben. Teilweise erfolgt die Behebung solcher Probleme schon im Prozeß der Schematransformation oder -integration. So konnten im vorliegenden Fall einige Inkonsistenzen behoben werden, die aufgrund mehrfacher Abspeicherung semantisch gleicher Objekte auftraten. Auch können bei einer Implementation der vorgeschlagenen Konzepte die separat vorliegenden und schwierig zu bedienenden Transformatoren (Kommandozeilenprogramme) entfallen, indem sie in das gemeinsame Datenbanksystem integriert werden.

Damit können für den Benutzer des remodellierten Systems FACTOTUM neue Bedienfolgen aufgestellt werden, die die Durchführung einer kompletten Fabrikplanung mit ihren Phasen vereinfachen und die Qualität der Ergebnisse verbessern. Dabei bleiben aber die Bedienschritte innerhalb der einzelnen Komponenten erhalten, was die Akzeptanz des neuen Systems wesentlich erhöht. Außerdem können auf der Grundlage des föderierten Datenbanksystems neue Applikationen bzw. *front ends* implementiert werden, die neue Funktionalität wie z.B. eine globale Datenschnittstelle, eine vollständige Projektverwaltung oder Funktionen zum *facility management* anbieten könnten. Der wichtigste Vorteil der FDBS-Nutzung bleibt jedoch die Konsistenzsicherung für die Daten zur Laufzeit, die erst durch eine integrierte Lösung wie die hier beschriebene erreicht werden kann.

Aus dieser Sicht erscheint es wünschenswert, eine Integration der FACTOTUM-Komponenten mit Hilfe eines föderierten Datenbanksystems vorzunehmen. Nachteilig wirkt sich aus, daß ein einsatzfähiges föderiertes Datenbanksystem z.Z. nicht bzw. nur prototypisch zur Verfügung steht, so daß die Erarbeitung von Ansätzen zur Realisierung der vorgestellten Föderierungslösung im Rahmen der Diplomarbeit nicht möglich war. Es ist allerdings die Auffassung des Autors, daß unter der Voraussetzung der Beibehaltung der im FACTOTUM enthaltenen Komponenten die Integration mittels FDBS die einzig effiziente Möglichkeit ist, zu einer Verbesserung der Funktionalität dieses Systems in der beschriebenen Richtung zu gelangen.

Kapitel 10

Schlußbetrachtung und Ausblick

In der vorliegenden Arbeit sollte an Systemen zur Fabrikplanung untersucht werden, wie eine Integration dateibasierter Datenhaltungssysteme in ein föderiertes Datenbanksystem prinzipiell möglich ist, und es sollten Konzepte für eine solche Datenbankintegration aufgestellt werden. Der Hintergrund dazu ist die Notwendigkeit, die von heterogenen Ingenieursystemen gehaltenen Daten zu integrieren, um einen einheitlichen Datenzugriff und eine Konsistenzsicherung zu ermöglichen. Das kann, wie in der vorliegenden Arbeit gezeigt wurde, mit einem föderierten Datenbanksystem geschehen.

Die Arbeit umfaßte die Lösung mehrerer Teilprobleme:

- Die vorhandenen Komponenten des Produktionsplanungssystems FACTOTUM wurden analysiert und ihre Datenhaltungskonzepte detailliert untersucht und beschrieben. Dabei wurde eine Methode zur Beschreibung von Dateien mit Hilfe von Konzepten der Dokumentbeschreibungssprache SGML entwickelt.

- Anhand dieser und vorheriger Untersuchungen wurde eine Klassifizierung von Datenhaltungskonzepten vorgenommen, wobei für jede Klasse Integrationskonzepte, Probleme und Lösungen dargestellt wurden. Damit konnte eine Zuordnung der Datenhaltungskonzepte in FACTOTUM zu diesen Klassen erfolgen, und es konnten entsprechende Schlüsse für die Integration der einzelnen Werkzeuge gezogen werden.

- Die Eignung des am ITI entwickelten FDBS-Integrationsverfahrens mittels GIM für die Integration des FACTOTUM wurde festgestellt, woraufhin dieses Verfahren erstmals zur Schematransformation und -integration eines existierenden heterogenen Softwaresystems in ein föderiertes Datenbanksystem eingesetzt wurde.

- Nach Abschluß der konzeptionellen Integration wurden Möglichkeiten für eine praktische Realisierung der FDBS-Anbindung des FACTOTUM untersucht, wobei die Hauptschwierigkeit darin bestand, eine Möglichkeit zur Echtzeitanbindung der *MS-DOS*-basierten Komponenten des FACTOTUM an ein föderiertes DBMS zu finden und zu erproben.

Außerdem wurden die Grundlagen föderierter Datenbanksysteme zusammenfassend dargelegt, und es wurde eine kurze Einführung in die Integrationsmethodik mit Hilfe des *Generic Integration Model* gegeben.

Die Einführung in FDBS erfolgte dabei unter Verwendung von Terminologie und Methodik von A. P. Sheth und J. A. Larson [SL90], wobei auch weitere Arbeiten auf diesem Gebiet vergleichend ausgewertet wurden. Die dort entwickelte Schemaarchitektur wurde im weiteren Verlauf der Arbeit als Grundlage für die Ausführung der FDBS-Integration verwendet. Ein Überblick über verwandte Ansätze zur Integration heterogener Datenbanksysteme zusammen mit einer kurzen Bewertung wurde ebenfalls gegeben.

Das *Generic Integration Model (GIM)* stellt ein Datenmodell für das integrierte Schema eines föderierten Datenbanksystems dar, wobei für die Ableitung von Schemata in diesem Datenmodell eine Anzahl von Algorithmen und Konzepten vorliegen, so daß eine Datenbankintegration anhand dieser Konzepte möglich ist. Die Eignung der GIM-Konzepte für eine FDBS-Integration wurde untersucht, wobei Anforderungen an ein Integrationsmodell mit den von GIM angebotenen Eigenschaften verglichen wurden. Dabei konnten wesentliche Vorteile dieser Methodik gegenüber anderen, teilweise gegensätzlichen Ansätzen in der Literatur herausgearbeitet werden. Dieser Teil der Arbeit basiert im wesentlichen auf der zu GIM vorhandenen Literatur [Sch95, SS95, SS96a, CHJ+96, Jan96].

Nach Untersuchung der im FACTOTUM verwendeten Datenhaltungskonzepte wurde eine Klassenaufteilung solcher Konzepte vorgenommen, wobei Vorarbeiten von M. Höding [Höd96] berücksichtigt wurden. Dabei wurde die Verwendung der Sprache SGML mit einigen Erweiterungen zur Modellierung von Dateien vorgeschlagen. Diese Modellierung dient zur Identifizierung von Datenelementen in Dateien sowie zur einheitlichen Darstellung unterschiedlicher Dateiformate und soll eine Unterstützung im Prozeß der Erstellung von Transformatoren zur Datenbankanbindung dateibasierter Software geben.

Anhand des Produktionsplanungssystems FACTOTUM wurden die vorgestellten Konzepte, Erweiterungen und Vorschläge verwendet und in der Praxis erprobt. Dabei wurden ausgehend von einer verbalen Beschreibung der FACTOTUM-Komponenten nacheinander die Datenbankschemata der lokalen und globalen Schemaebenen nach Sheth/Larson entwickelt und wichtige Nebenbedingungen dieser Schemata aufgestellt. Dieser gesamte Prozeß verfolgte das Ziel, eine vollständige Beschreibung einer Föderierung des Systems FACTOTUM vorzuneh-

men, um so bei gegebener Verfügbarkeit eines einsatzfähigen föderierten Datenbanksystems eine Implementation geeigneter Transformatoren und eine Schemaaufstellung und somit eine Integration der FACTOTUM-Komponenten zu ermöglichen. Abschließend wurde eine Bewertung des Erreichten vorgenommen.

Nach Beendigung dieser Arbeiten wurden Möglichkeiten zur praktischen Realisierung der Integration dateibasierter Komponenten in ein FDBS untersucht. Die Notwendigkeit dazu erwuchs aus der mangelhaften Kooperationsfähigkeit der untersuchten System, die sich teilweise aus ihrer Implementation unter dem Betriebssystem *MS-DOS* ergab. Zur Lösung einiger Probleme einer solchen Konstellation wurde ein Rechnerverbund eines UNIX-basierten Systems mit einem *MS-DOS*-PC untersucht und einige Implementationsaspekte der dafür benötigten Software betrachtet. Hier mußte eine Einschränkung der Arbeiten vorgenommen werden, da geeignete Soft- und insbesondere Hardware erst im Endstadium der Diplomarbeit zur Verfügung stand.

Eine Bewertung der Möglichkeiten und Potentiale einer Nutzung föderierter Datenbanksysteme im Bereich der Fabrikplanung unter besonderer Berücksichtigung des Produktionsplanungssystems FACTOTUM wurde in Kapitel 9 vorgenommen.

Weiterführende Arbeiten zu den in der vorliegenden Diplomarbeit betrachteten Themen sind erforderlich und Gegenstand zukünftiger Untersuchungen. Das Gebiet föderierter Datenbanksysteme ist jung, und allgemeingültige Standards sind noch nicht entwickelt worden. GIM stellt ebenfalls ein Konzept dar, das sich in der Entwicklung befindet. Insbesondere hier sind weitere Verfeinerungen der Integrationsmethodik und eine Abdeckung möglichst vieler auftretender Konflikte nötig und zu erwarten. Weiterhin stellt die Verfügbarkeit eines föderierten Datenbanksystems wie z.B. OpenDM/Efendi eine Voraussetzung für eine Weiterführung praktischer Arbeiten am vorgestellten System dar. Aus einer praktischen Umsetzung des in dieser Arbeit ausgeführten Systemdesigns ergäbe sich die Möglichkeit einer Verifizierung und eines Tests der abgeleiteten Schemata und Implementationskonzepte. Schließlich müssen die begonnenen Arbeiten zur stabilen Anbindung PC-basierter Software an ein FDBMS weitergeführt und deren Konzepte verallgemeinert werden.

Literaturverzeichnis

[Aut91] Autorenkollektiv. *CAD FAIF, Allgemeine Charakteristik*. Fachbereich Maschinenbau II der TU Chemnitz, Technologische Betriebsprojektierung, Chemnitz, 1991.

[BEK+94] R. Böttger, Y. Engel, G. Kachel, S. Kolmschlag, D. Nolte, and E. Radeke. Problem Analysis and Requirement Specification on Heterogeneous Database Systems. Cadlab Report 40/94, Cadlab Paderborn, Germany, 1994.

[BH91] M. W. Bright and A. R. Hurson. Linguistic Support for Semantic Identification and Interpretation in Multidatabases. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan*, pages 306–313. IEEE Computer Society Press, 1991.

[BHP92] M. Bright, A. Hurson, and S. Pakzad. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, 25(3):50–56, March 1992.

[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.

[Bob96] A. R. Bobak. *Distributed and Multi-Database Systems*. Artech House, Inc., Norwood, MA, 1996.

[Bor92] G. Born. *Referenzhandbuch Dateiformate*. Addison-Wesley, Bonn, München, Paris [u.a.], 1992.

[Cat94] R. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[CHJ+96] S. Conrad, M. Höding, S. Janssen, G. Saake, I. Schmitt, and C. Türker. Integrity Constraints in Federated Database Design. Preprint Nr. 2, Fakultät für Informatik, Universität Magdeburg, April 1996.

[CL94] J. Chomicki and W. Litwin. Declarative Definition of Object-Oriented Multidatabase Mappings. In T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 375–392. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H.-D. Ehrich. Conceptual Modelling of Database Applications Using an Extended ER Model. *Data & Knowledge Engineering, North-Holland*, 9(2):157–204, 1992.

[FS94] R. Fietz and C. Schmidt. Weiterentwicklung des Arbeitsplatzes "Logistikgerechte Fabrikplanung". In *Jahresbericht 1994*, pages 42–45. Fraunhoferinstitut für Fabrikautomatisierung und -betrieb Magdeburg, 1994.

[Grö96] S. Gröpke. Untersuchungen zur Aufbereitung von FAST/pro als Fabrikanalysewerkzeug für den Logistikarbeitsplatz „LOGAP". Studienarbeit, Otto-von-Guericke-Universität Magdeburg, 1996.

[Här94] M. Härtig. *Objektorientierte Integration von autonomen Datenhaltungssystemen.* Dissertation, Verlag Dr. Kovač, Hamburg, 1994.

[Höd96] M. Höding. Föderierung von Dateien: Finden, Verwalten und Nutzen von Metainformationen. In S. Conrad, G. Saake, I. Schmitt, and C. Türker, editors, *Kurzfassungen – 8. Workshop "Grundlagen von Datenbanken", Friedrichsbrunn/Harz (28.05.–31.05.96)*, pages 46–50. Institut für Technische Informationssysteme, Universität Magdeburg, May 1996.

[HS93] C. Hübel and B. Sutter. *DB-Integration von Ingenieuranwendungen.* Friedr.Vieweg & Sohn Verlags-GmbH, Braunschweig/Wiesbaden, 1993.

[HS95] A. Heuer and G. Saake. *Datenbanken — Konzepte und Sprachen.* International Thomson Publishing, Bonn, 1995.

[HTJ+95] M. Höding, C. Türker, S. Janssen, K.-U. Sattler, S. Conrad, G. Saake, and I. Schmitt. Föderierte Datenbanksysteme — Grundlagen und Ziele des Projektes **SIGMA**$_{FDB}$. Preprint Nr. 12, Fakultät für Informatik, Universität Magdeburg, December 1995.

[Hüs95] F. Hüsemann. Forschungsergebnisse der Fakultät für Mathematik und Informatik: Konzepte der Datenintegration—Eine Bestandsaufnahme. Technical Report Math/95/19, Friedrich-Schiller-Universität Jena, 1995.

[ISO86] ISO (International Organization for Standardization). *The Standard Generalized Markup Language. (ISO 8879)*, 1986.

[Jan96] S. Janssen. Integration und Erhaltung von Integritätsbedingungen in FDBSen. In S. Conrad, G. Saake, I. Schmitt, and C. Türker, editors, *Kurzfassungen – 8. Workshop "Grundlagen von Datenbanken", Friedrichsbrunn/Harz (28.05.–31.05.96)*, pages 56–60. Institut für Technische Informationssysteme, Universität Magdeburg, May 1996.

[KCGS95] W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. In W. Kim, editor, *Modern Database Systems*, pages 521–550. ACM Press, New York, 1995.

[KLK91] R. Krishmamurthy, W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan*, pages 144–151. IEEE Computer Society Press, April 1991.

[Kre95] B. Krenzel. *Integration der Computersimulation in die Planung und Gestaltung der Arbeitsorganisation im Rahmen der Fabrikplanung.* Dissertation, Otto-von-Guericke-Universität Magdeburg, 1995.

[Rad94] E. Radeke. Efendi: Federated Database System of Cadlab. Technical Report 39/94, University of Paderborn & Siemens Nixdorf, October 1994.

[Rad95] E. Radeke. *OpenDM/ODMG–Skriptum.* Cadlab, Paderborn, 1995.

[Rah94] E. Rahm. *Mehrrechner-Datenbanksysteme.* Addison-Wesley, Bonn, 1994.

[Ric92] M. Rickert. Praktische Neustrukturierung eines Fertigungsbetriebes mit Hilfe rechnerintegrierter Planungsmethoden. In *VDI-Berichte 949: Rechnergestützte Fabrikplanung '92.* VDI-Verlag GmbH, Düsseldorf, 1992.

[SCG91] F. Saltor, M. Castellanos, and M. Garcia-Solaco. Suitability of Data Models as Canonical Models for Federated Databases. *ACM SIGMOD RECORD*, 20(4):44–48, December 1991.

[Sch90] M. Schulze Dieckhoff. *Integriertes System zur ablauforientierten Fabrikplanung.* Darmstädter Forschungsberichte für Konstruktion und Fertigung. Carl Hanser Verlag München Wien, Darmstadt, 1990.

[Sch95] I. Schmitt. Flexible Integration and Derivation of Heterogeneous Schemata in Federated Database Systems. Preprint Nr. 10, Fakultät für Informatik, Universität Magdeburg, November 1995.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SMB88] Microsoft Corporation, Intel Corporation. *Microsoft Networks/OpenNET File Sharing Protocol*, September 1988.

[SS95] I. Schmitt and G. Saake. Managing Object Identity in Federated Database Systems. In M. Papazoglou, editor, *Proc. of the 14th Int. Conf. on Object-Oriented and Entity-Relationship Modeling (OOER'95), Gold Coast, Australia*, pages 400–411. LNCS 1021, Springer-Verlag, December 1995.

[SS96a] I. Schmitt and G. Saake. Flexible Generation of Global Integrated Schemata using GIM. In S. Conrad, M. Höding, S. Janssen, and G. Saake, editors, *Kurzfassungen des Workshops "Föderierte Datenbanken", Magdeburg, 22.04–23.04.96*, pages 29–43. Bericht 96–01, Institut für Technische Informationssysteme, Universität Magdeburg, April 1996.

[SS96b] I. Schmitt and G. Saake. Integration of Inheritance Trees as Part of View Generation for Database Federations. In *Proc. of the 15th Int. Conf. on Conceptual Modelling (ER'96)*. LNCS, Springer-Verlag, Berlin, October 1996. *To appear.*

[SS96c] I. Schmitt and G. Saake. Schema Integration and View Generation by Resolving Intensional and Extensional Overlappings. In *Proc. of the 9th Int. Conf. on Parallel and Distributed Computing Systems (PDCS'96)*, September 1996. *To appear.*

[TK78] D.C. Tsichritzis and A. Klug. The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems. *Information Systems*, 3(3):173–191, 1978.

[vH90] E. van Herwijnen. *Practical SGML.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

Anhang A

Struktur der Dateien in FACTOTUM

Im folgenden sollen die Dateien der einzelnen FACTOTUM-Komponenten, die in Abschnitt 5.1 vorgestellt wurden, untersucht werden. Dabei werden alle von den jeweiligen Systemen genutzten Dateien auf ihre Bedeutung für die Datenbankintegration untersucht, und für die zu integrierenden Dateien Strukturinformationen abgeleitet. Hier wird auf die in Kapitel 3 eingeführten Konzepte zurückgegriffen.

A.1 Dateistruktur FASTpro

Alle für die weiteren FACTOTUM-Komponenten (LAPLAS, DUMAS, TAYLOR) wichtigen Daten finden sich in einer einzigen Tabelle der relationalen Datenbank, der Tabelle AG. Die Struktur dieser Tabelle ist in Tabelle A.1 beschrieben.

Dabei bedeuten Ln die Länge und Nk die Nachkommastellen eines Attributs in der Tabelle. Die drei letzten Attribute finden sich nicht in [Grö96], sind aber offensichtlich in der Tabelle vorhanden, da auf sie mit SQL-Anfragen zugegriffen wird. Ein Record in dieser Tabelle ist also ein Arbeitsgang, in dem eine Anzahl gleichartiger Werkstücke auf einer bestimmten Maschine bearbeitet wird. Für den FACTOTUM wichtig sind die folgenden Attribute :

- NAU. Die Nummer des Auftrags. Die Notation richtet sich hier nach der Herkunft der Daten. NAU kann insbesondere sowohl ein String wie `0153MTR_1` als auch ein numerischer Wert, also eine Maschinennummer sein. Dieses Attribut wird benötigt, um eine Reihe von Arbeitsgängen, die an einem Werkstück ausgeführt werden sollen, eben diesem Werkstück zuzuordnen.

Inhalt	Feld	Typ	Pos	Ln	Nk
Satzart	SATZ_ART	CHAR	1	4	
Bearbeitungsart	BEARB_ART	CHAR	5	4	
Auftragsnummer	NAU	CHAR	9	16	
Arbeitsgangnummer	NAG	NUMBER	25	5	
Externer Status	STE	NUMBER	30	4	
Planmenge	MZ	NUMBER	34	13	3
Mengeneinheit	ME	CHAR	47	5	
Bereich	BER	CHAR	52	10	
Kostenstelle	KOS	CHAR	62	10	
Machinengruppe	MGR	CHAR	72	10	
Planrüstzeit	ZRP	NUMBER	82	13	3
Planeinzelzeit	ZEP	NUMBER	95	13	3
Sollstarttermin	SOLL_START	DATUM	108	11	
Sollendtermin	SOLL_ENDE	DATUM	119	11	
Bearbeitungsstation	PNAS	NUMBER			
Bearb.stat.-Vorgänger	PNAS_VOR	NUMBER			
Bearb.stat.-Nachfolger	PNAS_NACH	NUMBER			

Tabelle A.1: Attribute der Relation AG

- NAG. Die Nummer eines Arbeitsgangs. Für ein Werkstück gibt die aufsteigende Reihenfolge der Arbeitsgangnummern die Reihenfolge der Bearbeitungsschritte an. Zur Bedeutung dieses Attributs siehe unten.

- PNAS, PNAS_VOR, PNAS_NACH. Die Nummer der aktuellen, der vorherigen und der nächsten Bearbeitungsstation (Maschine). Die vorherigen und nächsten Stationen werden benötigt, um „erste“ und „letzte“ Arbeitsgänge zu identifizieren, also Transporte *in* die Fabrikhalle und *aus* der Fabrikhalle bestimmen zu können.

- MZ. Die Planmenge gibt die Anzahl der Werkstücke an, die während dieses Arbeitsgangs auf dieser Maschine bearbeitet werden. Dieser Wert wird benötigt zur Dimensionierung der benötigten Transportmittel.

- ZEP, ZRP. Die Planeinzel- und -rüstzeiten werden vom Simulationssystem TAYLOR für die Simulation der Materialflüsse benötigt.

A.1.1 Schnittstelle FAST/pro-LAPLAS

FAST/pro stellt für LAPLAS Materialflußdaten bereit. Ein Record dieser Materialflußdaten enthält eine Quelle und ein Ziel für einen Materialfluß sowie seine

Intensität (die Transportmenge). Diese Daten werden mit Hilfe von drei verschiedenen SQL-Anfragen bereitgestellt. Die Unterscheidung beruht darauf, daß „normale“ Arbeitsgänge von solchen zu unterscheiden sind, die als erste oder letzte für ein Werkstück durchzuführen sind. Solche Arbeitsgänge erfordern den Transport aus dem Lager und in das Lager und müssen so im Layoutplanungsprozeß gesondert behandelt werden. Weitere für LAPLAS benötigte Informationen sind die Art des zu verwendenden Transportmittels sowie der zurückzulegende Weg. Ersteres ist in FAST/pro nicht verfügbar und wird deshalb mit einem konstanten Wert „TMA“ (Transportmittel A) vorbelegt. Der Weg kann erst ermittelt werden, wenn die Position der Maschinen feststeht und wird deshalb mit 0 vorbelegt. Die SQL-Anfragen zur Erstellung dieser drei Exportdateien von FAST/pro sehen folgendermaßen aus:

```
select NAU, NAG, PNAS_VOR, PNAS,'TMA', Mz, 0
from AG
where PNAS_VOR is NULL;

OUTPUT TO mfm_l_m.txt FORMAT ascii
```

Ergebnis(Beispiel):

```
'081465-N',20,,6,'TMA',1.000,0
'153',10,,97,'TMA',7.000,0
'0153MTR_1',10,,1,'TMA',145.000,0
'0153MTR_2',10,,1,'TMA',145.000,0
'0154MTR_3',5,,1,'TMA',51.000,0
...

select NAU, NAG, PNAS, PNAS_NACH ,'TMA', Mz, 0
from AG
where PNAS_NACH is NULL;

OUTPUT TO mfm_m_l.txt FORMAT ascii

select NAU, NAG, PNAS_VOR, PNAS,'TMA', Mz, 0
from AG
where PNAS_VOR > 0 and PNAS_NACH > 0

OUTPUT TO mfm_m_m.txt FORMAT ascii
```

Die Ergebnisse der Ausgabe sind analog dem ersten Beispiel.

Im folgenden sind die SGML-Strukturinformationen für die erste Datei angegeben. Die Informationen für die anderen beiden Dateien lassen sich leicht ableiten.

```
<!ELEMENT MFM_L_M      - -  (Datensatz)+                >
<!ELEMENT Datensatz    - -  (NAU, NAG, PNAS_VOR, PNAS,
                                    TM, Mz, Null) >
<!ELEMENT NAU          - -  (#PCDATA)                   >
<!ELEMENT NAG          - -  (#PCDATA)                   >
<!ELEMENT PNAS_VOR     - -  (#PCDATA)                   >
<!ELEMENT PNAS         - -  (#PCDATA)                   >
<!ELEMENT TM           - -  (#PCDATA)                   >
<!ELEMENT Mz           - -  (#PCDATA)                   >
<!ELEMENT Null         - -  (#PCDATA)                   >
```

Name	**Endtoken**	**Datentyp**
NAU	,	String
NAG	,	Integer
PNAS_VOR	,	*leer*
PNAS	,	Integer
TM	,	String
Mz	,	Real
Null	EOL	Integer

Tabelle A.2: Datentypen der Dateien `mfm*.txt`

Bemerkungen zu Tabelle A.2:
Strings sind eingeschlossen in Hochkommata (' ').
Die von FAST/pro für LAPLAS erzeugten Dateien müssen in das Arbeitsverzeichnis von LAPLAS geschrieben werden.

Diese Dateien werden nun mit Hilfe von in C++ für DOS geschriebenen Transformatoren in das für LAPLAS nötige Format für Materialflußmatrizen (`*.MFM`-Dateien) übersetzt. Die Attribute NAU und NAG werden hier zwar mit Hilfe der SQL-Anfragen aus der Tabelle AG extrahiert, aber dann bei der Transformation nicht übertragen, da sie in LAPLAS nicht benötigt werden. Auch ist das Eintragen der Dummy-Daten für Transportmittel und Weglänge in der SQL-Anfrage unnötig, da es auch im Transformator geschehen könnte. Für das genaue Format der vom Transformator erzeugten Materialflußdateien siehe Kapitel 5.1.2.

A.1.2 Schnittstelle FAST/pro-TAYLOR

Das Simulationssystem TAYLOR erhält von FAST/pro Informationen über Planrüst- und Planeinzelzeiten für jeden Arbeitsgang. Folgende SQL-Anfrage wird benutzt:

```
select NAU, NAG, PNAS, ZEP, ZRP
from AG
order by NAU, NAG;

OUTPUT TO taylor.txt FORMAT ascii
```

Ergebnis(Beispiel):

```
'0153MTR_1',10,1,.001295139,.006944444
'0153MTR_1',20,79,.014652778,0.000000000
'0153MTR_1',40,74,.014009722,.086805556
'0153MTR_1',50,75,.011300694,.069444444
...
```

Die SGML-Definition stellt sich wie folgt dar:

```
<!ELEMENT TAYLOR       - -  (Datensatz)+                >
<!ELEMENT Datensatz    - -  (NAU, NAG, PNAS, ZEP, ZRP)  >
<!ELEMENT NAU          - -  (#PCDATA)                   >
<!ELEMENT NAG          - -  (#PCDATA)                   >
<!ELEMENT PNAS         - -  (#PCDATA)                   >
<!ELEMENT ZEP          - -  (#PCDATA)                   >
<!ELEMENT ZRP          - -  (#PCDATA)                   >
```

Name	**Endtoken**	**Datentyp**
NAU	,	String
NAG	,	Integer
PNAS	,	Integer
ZEP	,	Real
ZRP	EOL	Real

Tabelle A.3: Datentypen der Datei `taylor.txt`

Bemerkung zu Tabelle A.2:
Strings sind eingeschlossen in Hochkommata (''').
Die Datei befindet sich im Arbeitsverzeichnis von TAYLOR.

Diese Datei wird mit einem weiteren Transformator in drei Dateien umgewandelt, die von TAYLOR eingelesen werden können. Die erste Spalte (Auftragsnummer) der SQL-Anfrage wird dabei zwar nicht mehr benutzt, ist aber für die Sortierung der Arbeitsgänge durch die SQL-Schnittstelle notwendig. Für das Format der drei erzeugten Dateien siehe Kapitel 5.1.4.

A.2 Dateistruktur LAPLAS

Die zu importierenden Grafikdateien für Maschinenformen (und auch den Grundriß einer Fabrikhalle) müssen im Grafikformat DXF vorliegen und können demzufolge z.B. in AutoCAD erzeugt werden[1]. Da das DXF-Format standardisiert ist [Bor92] und außerdem ein *parsing* von Dateien im DXF-Format hier nicht notwendig ist, soll hier nur das für Materialflußmatrizen benötigte Format beschrieben werden, das ebenfalls für den Import und Export von Daten notwendig ist. Danach folgt eine Beschreibung der von LAPLAS intern verwendeten Dateien.

A.2.1 Materialflußdateien in LAPLAS

Eine Materialflußmatrix hat 5 Spalten, die jeweils durch ein Leerzeichen getrennt sind. Die Bedeutung der Spalten ist wie folgt:

- *Quelle.* Die Nummer bzw. der Name der Maschine, von der das Material transportiert werden muß. Im FACTOTUM steht hier der String „EIN“, wenn der Transport aus dem Lager zu einer Maschine erfolgt.
- *Senke.* Die Nummer bzw. der Name der Maschine, zu der das Material transportiert werden muß. Hier steht der String „AUS“, wenn der Transport von einer Maschine in das Lager erfolgt.
- *Transportmittel.* Der Name des zu verwendenden Transportmittels. Bei Verwendung von Daten aus FAST/pro muß hier durch einen Transformator ein Wert für das Transportmittel eingetragen werden. In der zur Zeit im Einsatz befindlichen Version des FACTOTUM steht hier immer „TMA“.
- *Anzahl Teile.* Die Anzahl der Werkstücke, die für einen Arbeitsgang transportiert werden muß.
- *Weg.* Der Weg (in einer Längeneinheit), der für einen Transport zwischen den angegebenen Maschinen zurückgelegt werden muß.

Die Weglängen dienen dabei nur als Platzhalter für die Abspeicherung der Daten nach erfolgter Layoutplanung. Eine Zeile in einer Matrix bedeutet demzufolge einen Transportvorgang, bei dem eine Anzahl Werkstücke von einem Punkt zu einem anderen Punkt transportiert werden muß. Die drei Materialflußmatrizen für Transporte vom Lager zu einer Maschine, von einer Maschine zum Lager, und von einer Maschine zu einer anderen Maschine sehen also z.B. wie folgt aus:

[1] Diese `*.dxf`-Dateien müssen bestimmte semantische Voraussetzungen erfüllen, s. Kapitel 5.1.6

```
1. Datei mfm_l_m.mfm
EIN 6 TMA 1 0
EIN 97 TMA 7 0
EIN 1 TMA 145 0
...

2. Datei mfm_m_l.mfm
112 AUS TMA 7 0
112 AUS TMA 145 0
112 AUS TMA 145 0
...

3. Datei mfm_m_m.mfm
103 103 TMA 1 0
39 103 TMA 1 0
39 39 TMA 1 0
...
```

Die SGML-Definitionen sehen wie folgt aus:

```
<!ELEMENT MFM_L_M      - -  (Datensatz)+                >
<!ELEMENT Datensatz    - -  (Quelle, Senke, TM_name,
                             Mz, Entf)                  >
<!ELEMENT Quelle       - -  (#PCDATA)                   >
<!ELEMENT Senke        - -  (#PCDATA)                   >
<!ELEMENT TM_name      - -  (#PCDATA)                   >
<!ELEMENT Mz           - -  (#PCDATA)                   >
<!ELEMENT Entf         - -  (#PCDATA)                   >
```

Name	**Endtoken**	**Datentyp**
Quelle	*Leerzeichen*	String
Senke	*Leerzeichen*	String
TM_name	*Leerzeichen*	String
Mz	*Leerzeichen*	Integer
Entf	EOL	Integer

Tabelle A.4: Datentypen der Datei `mfm_l_m.mfm`

Bemerkung zu Tabelle A.4:
Das Attribut Quelle enthält den konstanten Wert `EIN`.
Die gleiche Beschreibung gilt auch für die Dateien `mfm_m_l.mfm` und `mfm_m_m.mfm`, mit folgenden Einschränkungen:

`mfm_m_l.mfm`: Das Attribut Quelle enthält variable Werte, das Attribut Senke enthält den konstanten Wert `AUS`.
`mfm_m_m.mfm`: Die Attribute Quelle und Senke enthalten variable Werte.

A.2.2 LAPLAS-interne Dateien

Die intern von LAPLAS verwendeten Dateien sind insofern von Interesse, als sich die dort enthaltenen Daten teilweise mit den Daten anderer FACTOTUM-Komponenten überschneiden und sie daher den Kern einer Integration bilden. Daher sollen diese im folgenden beschrieben werden.

In der folgenden Aufzählung umschließen spitze Klammern (<. . .>) variable Dateinamen, d.h. Dateinamen, die in Abhängigkeit von einem Projekt entweder vom System oder vom Benutzer gewählt werden. Die Einteilung der Dateien in Klassen anhand ihrer Formate folgt [Höd96] und wird in Kapitel 3.2 beschrieben.

1. `projekte.dat`
   Diese Datei enthält die Namen der in LAPLAS erzeugten Projekte.

   Format: satzorientiert, 2 Felder pro Zeile, durch 2 Leerzeichen getrennt:

   - Nummer Integer als 2-stelliger String
   - Name String, max. 6 Zeichen, gibt Filenamen für Projekt an

   Beispiel:

   ```
   01  CEBIT
   02  FISE
   03  CBGES
   ```

   SGML-Beschreibung (vgl. Tabelle A.5):

   ```
   <!ELEMENT PROJEKTE      - -  (Datensatz)+              >
   <!ELEMENT Datensatz     - -  (PNR, PNM)                >
   <!ELEMENT PNR           - -  (#PCDATA)                 >
   <!ELEMENT PNM           - -  (#PCDATA)                 >
   ```

2. `laplasii.flb`
   Die globale Liste aller Maschinenbibliotheken. Maschinenbibliotheken können mit LAPLAS interaktiv erzeugt werden. Dabei können Maschinen einzeln als AutoCAD-DXF-Files importiert werden.

Name	Länge	Trennzeichen	Datentyp
PNR	2	*2 Leerzeichen*	Integer
PNM	6	EOL	String

Tabelle A.5: Datentypen der Datei `projekte.dat`

Format: satzorientiert, 1 Feld pro Zeile:

- Name String, max. 8 Zeichen, gibt Filenamen für Bibliothek an

Beispiel:

```
FRANCKE
MASHBIBO
```

Diese Datei soll nicht in SGML modelliert werden, da ein *Parsing* durch das FDBMS nicht notwendig ist.

3. `<Maschinen-Bibliothek>.lis`
   Liste von Maschinen-Namen. Für jede in `laplasii.flb` aufgeführte Bibliothek existieren genau eine `*.lis`-Datei mit den Namen der dort definierten Maschinen sowie eine `*.dat`-Datei (siehe dort) mit den eigentlichen Definitionen. Die Maschinendefinitionen sind nicht projektspezifisch.

   Format: satzorientiert, 1 Feld pro Zeile:

   - Name String

   Beispiel(`mashbibo.lis`):

   ```
   BKENC01
   DLZ01
   DR01
   FKFS01
   ```

   Eine Strukturbeschreibung mittels SGML ist nicht erforderlich.

4. `<Bibliothek>.dat`
   Abmessungen und Attribute für eine Anzahl von Maschinen („Flächendaten“). Beim interaktiven Aufbau eines Projekts mittels LAPLAS kann

jeweils eine Maschinenbibliothek „aktiviert“ werden, aus der dann Maschinendaten in die jeweilige Projektdatei übernommen werden. Die Bibliothekendaten werden also kopiert und in `<version>.lyt` abgelegt. So ist die gemeinsame Verwendung sowohl von selbstdefinierten als auch von Maschinen aus verschiedenen Bibliotheken möglich. Das Format ist spezifisch für LAPLAS und nicht dokumentiert. Die Reihenfolge der Maschinendefinitionen ist nicht notwendigerweise die in `*.lis` definierte.

Format: stark strukturierter Text. Es werden Koordinaten der Eckpunkte der Maschinengrundfläche abgelegt. Außerdem werden Darstellungsoptionen gesetzt (3 einstellige Zahlen). Es werden jeweils drei Flags gesetzt (Attributfläche, Sympathien, Materialflüsse).

Beispiel(`wolf12.dat`):

```
27
        0.00        0.00&
        0.00        0.31&
        0.80        0.31&
        0.80        0.00
        0.40        0.16
        0.40        0.16
        1 6 0
        ATT:J
        PSEUDO1
        3.00
        SYMP:N
        MAT:J
        **********************************
        45
        0.00        0.00&
        ...
```

Eine Strukturbeschreibung ist nicht erforderlich, da kein *Parsing* dieser Datei stattfindet.

5. `<Projekt>.att`
   Liste von user-definierten Attributen, die in einem Projekt verwendet werden. Attribute sind Eigenschaften von besonderen Flächen. Jede Fläche (Maschine, Hallengrund, Stütze, Wegefläche etc.) kann ein oder mehrere Attribute haben, die bei der Anordnung der Elemente durch das System berücksichtigt werden (z.B. Tragfähigkeit eines Krans oder einer Hallendecke).

Format: stark strukturierter Text.

Beispiel(`cebit_.att`):

```
Liste der Attribute zum Projekt: CEBIT
 NAME :                  EINHEIT :          1
 NAME :     HALLENKRAN   EINHEIT :         KG
```

Eine Strukturbeschreibung ist nicht notwendig, da der Inhalt dieser Datei als Attribut Attr der GIM-Klasse VAR mit dem Datentyp `binär` definiert ist, also eine Untersuchung hier enthaltener Daten nicht stattfindet.

6. `<Projekt>.far`
   Die Farben für die Darstellung eines Projektes.

   Format: stark strukturierter Text.

   Beispiel(`cebit_.far`):

```
*
Farben-Einstellungsdatei
*
Hintergrund           : 0
Skala                 :15
Fehlerrahmen          : 4
Fehlertext            ;15
Statusrahmen          :15
...
```

   Das Semikolon in der vierten Datenzeile kennzeichnet dabei eine Abweichung von der globalen Farbendatei.
   Der Inhalt dieser Datei ist ebenfalls als „binäres“ Attribut definiert, muß also nicht mittels SGML modelliert werden.

7. `farben.dat`
   Die globalen Farbeinstellungen, die als Voreinstellungen für neue Projekte genutzt werden.

   Format: wie vor.

8. `<Projekt>.gwt`
   Konfigurations-Einstellungen für ein Projekt

   Format: stark strukturierter Text.

Beispiel (`cbges_.gmt` aus Projekt `cbges`):

```
*
* STEUERDATEI FUER LAPLASII
*
Entfernungsmess. : RECHTWINKLIG
Kontakt-Faktor   :        1
Intensitaets-Fak.:        1
Transportleis.Fak:        1
Sympathie-Fak.   :        1
Entfernungsgr.Fak:        1
Groessen-Fak.    :        1
Wege-Fak.        :        1
Kosten-Fak.      :        1
ZW_Bei_Verschieb.:     NEIN
```

Ein *Parsing* findet nicht statt, daher wird keine SGML-Beschreibung angegeben.

9. `laplasii.gwt`
   Die globale Steuerdatei von LAPLAS als Voreinstellung für ein neues Projekt. Die Faktoren sind globale Wichtungsfaktoren, mit denen die lokalen Informationen wie z.B. Sympathien zwischen zwei Maschinen multipliziert werden. Damit können globale Prioritäten gesetzt werden. *Format:* wie vor.

10. `<Projekt>.kst`
    Kostensätze der Transportmittel für ein Projekt. Solche Kostentabellen können auch für einzelne Varianten erzeugt werden.

    Format: satzorientiert mit Header
    Das Format läßt sich aus dem Beispiel erkennen. In einem LAPLAS-Projekt können maximal 6 verschiedene Transportmittel verwendet werden, die Tabelle hat also genau 6 Zeilen.

    Beispiel(`cebit_.kst`):

```
KOSTENSAETZE FUER TRANSPORTMITTEL :
TRANSPORTMIT. HILFSMITTEL    KAPAZITAET     FIXKOSTEN      VARKOSTEN PRO METER
!********!*********!*********!*********!*********!*********!*********!*********!
TMA          X             1.000          1.000          1.000
TMB          X             1.000          1.000          1.000
...
```

11. `laplasii.kst`
    Die globale Kostentabelle als Standard für ein neues Projekt. Format wie

oben. Diese Datei wird vom FDBMS „geparst“. Daher soll eine SGML-Beschreibung vorgenommen werden (vgl. Tabelle A.6):

```
<!ELEMENT KOSTEN        - -  (Datensatz)+                >
<!ELEMENT Datensatz     - -  (Header, Name, Hilf, Kap,
                              Fix_k, Var_k)              >
<!ELEMENT Header        - -  (#PCDATA)                   >
<!ELEMENT Name          - -  (#PCDATA)                   >
<!ELEMENT Hilf          - -  (#PCDATA)                   >
<!ELEMENT Kap           - -  (#PCDATA)                   >
<!ELEMENT Fix_k         - -  (#PCDATA)                   >
<!ELEMENT Var_k         - -  (#PCDATA)                   >
```

Name	**Länge**	**Trennzeichen**	**Datentyp**
Header	*3 Zeilen (Ende mit EOL)*		String
Name	6	*8 Leerzeichen*	String
Hilf	1	*14 Leerzeichen*	String
Kap	5	*10 Leerzeichen*	Real
Fix_k	5	*10 Leerzeichen*	Real
Var_k	5	*EOL*	Real

Tabelle A.6: Datentypen der Datei `laplasii.kst`

12. `<Projekt>.var`
Varianten eines Projekts. Diese Namen werden vom Anwender festgelegt und lassen demzufolge nicht notwendigerweise auf das zugehörige Projekt schließen. Das Projekt ist allerdings in der Varianten-Layout-Datei selbst (`*.lyt`) definiert. Sollen die Daten einer Variante mit DUMAS weiterverarbeitet werden, sind die Variantendateien `*.atb` und `*.lyt` nötig, sowie eine (interaktiv aus LAPLAS exportierte) `*.mfm`-Datei mit dem gleichen Dateinamen wie die Variante. Die Informationen dieser `*.mfm`-Datei (Materialflüsse) sind auch in der zugehörigen `*.lyt`-Datei enthalten, werden von DUMAS dort aber nicht gefunden.

Format: satzorientiert, 2 Felder pro Zeile, durch 2 Leerzeichen getrennt.

- Nummer Integer als 2-stelliger String
- Name String, max. 6 Zeichen, gibt Namen der Variante an

Beispiel(`cebit_.var`):

```
01  VAR01
02  VAR02
03  VAR05
```

Die SGML-Beschreibung ist ähnlich der Beschreibung für `projekte.dat` (vgl.Tabelle A.7):

```
<!ELEMENT VARIANTE      - -  (Datensatz)+              >
<!ELEMENT Datensatz     - -  (NR, Name)                >
<!ELEMENT NR            - -  (#PCDATA)                 >
<!ELEMENT Name          - -  (#PCDATA)                 >
```

Name	**Länge**	**Trennzeichen**	**Datentyp**
NR	2	*2 Leerzeichen*	Integer
Name	6	EOL	String

Tabelle A.7: Datentypen der Dateien <projekt>.`var`

13. `<Projekt><Variantennummer>.atb`
    Ein File mit (zusätzlichen) Attributen für jede Variante eines Projekts.

    Format: stark strukturierter Text

    Beispiel (`cbges03.atb`, Variante 3 des Projekts `cbges`:

```
Liste der Attribute zur Variante: VAR03
 NAME :              FR  EINHEIT :          S
 NAME :            WKZB  EINHEIT :          1
```

    Bezüglich der Dateibeschreibung gilt das für die lokalen Attributdefinitionen Gesagte.

14. `<Projekt><Variantennummer>.lyt`
    Die Layoutdaten für jede Basis-, Planungs-, Attribut- und Wegefläche in einer Variante eines Projekts. Enthalten sind Informationen wie:
    Setup-Informationen, Position und Form der Maschinen, Größe der benötigten Fläche, Darstellungsoptionen.

Neben Flächen, die Grundflächen eigentlicher Maschinen bedeuten, werden auch andere Flächen wie der Hallengrund, Krane, Stützen oder freizuhaltende Wege angegeben.
Außerdem können Materialflußinformationen und Informationen über Attribute und Sympathien der Planungsflächen enthalten sein.
In einer `*.lyt`-Datei sind alle Informationen abgelegt, die für die Bearbeitung einer Variante eines Projekts nötig sind. Alle anderen Dateien (insbesondere `*.mfm`) sind nur zu Import- bzw. Export-Zwecken nötig.

Format: stark strukturierter Text hoher Komplexität.

Beispiel (`cebit_03.lyt`, Variante 3 von Projekt `cebit`):
Bemerkung: Variante 3 entspricht `VAR05` - siehe `cebit_.var`

```
 VERSION          : 1.00
 PROJEKT          : CEBIT
 VARIANTE         : VAR05
 DATUM            : 07.04.95
 X-Achsenlaenge m:         45.00
 RASTER           : EIN         1        10
 Matfluss-Raster  : AUS         1         1
 VERSCHIEBEGROES.:           5.00
 Flaechennamen    : EIN
 Quellen / Senken: EIN
 Materialfluss    : AUS
 Texte            : EIN
 Attributfl"achen  : AUS
 Sympathien       : EIN
$BF******!*********!*********!*********!*********!*********!*********!
 Flaechenname     : HALLENGRUND
 Flaechengroesse  :      1080.00
 Fixierung        :            2
 Koordinaten      :         0.00         0.00         0.00        30.00             &
                          36.00        30.00        36.00         0.00
 Rechtecke        :         0.00         0.00        36.00        30.00             &
                           0.00         0.00        36.00        30.00
 Darstellung      : 1 6 0
$RF******!*********!*********!*********!*********!*********!*********!*********!
 Flaechenname     : KRAN
 Flaechengroesse  :       544.00
 Koordinaten      :         1.00         1.00         1.00        17.00             &
                          35.00        17.00        35.00         1.00
 Rechtecke        :         1.00         1.00        35.00        17.00             &
                           1.00         1.00        35.00        17.00
 Attribute        :    HALLENKRAN      2000.00
 Darstellung      : 1 4 0
 ...
 ...
 ...
$PF******!*********!*********!*********!*********!*********!*********!*********!
 Flaechenname     : FUCNC02
 Flaechengroesse  :        10.14
 Fixierung        :            1
 Koordinaten      :         1.10        19.20         1.10        20.10             &
                           2.10        20.10         2.10        20.60             &
                           1.60        20.60         1.60        21.80             &
                           1.10        21.80         1.10        23.10             &
                           1.60        23.10         1.60        22.30             &
                           2.80        22.30         2.80        23.70             &
                           3.50        23.70         3.50        22.30             &
                           5.50        22.30         5.50        20.60             &
```

```
                          2.80       20.60        2.80       19.20
Rechtecke       :         1.10       19.20        5.50       23.70            &
                          1.10       19.20        1.60       20.10            &
                          1.10       21.80        1.60       23.10            &
                          1.60       19.20        2.10       20.10            &
                          1.60       20.60        2.10       22.30            &
                          2.10       19.20        2.80       22.30            &
                          2.80       20.60        3.50       23.70            &
                          3.50       20.60        5.50       22.30
Quelle          :         4.70       21.30
Senke           :         4.70       21.30
Materialfluss  N:         HAND02 1                          X            1 14JN &
M_Start/M_Ziel :         4.70       21.30        7.80        9.45
                          LAGER 1                          X          48 14JN &
M_Start/M_Ziel :         4.70       21.30       29.20       15.40
                          HAND01 1                         X            1 14JN
M_Start/M_Ziel :         4.70       21.30        7.80        5.25
                          Attribute       : KEINE ANGABEN
Sympathie       :        FUCNC03           5
Darstellung     : 1 5 0
...
...
```

Eine SGML-Modellierung führt zu folgendem Ergebnis (s. auch Tab. A.8):

```
<!ELEMENT LAYOUT    - -  (Einst, BF, (RF)*, (WF)*, (PF)*)          >
<!ELEMENT Einst     - -  (#PCDATA)                                  >
<!ELEMENT BF        - -  (Name, Groesse, Fix, koor, rec, Darst)     >
<!ELEMENT RF        - -  (Name, Groesse, koor, rec, Darst, Attr)    >
<!ELEMENT WF        - -  (Name, Groesse, Fix, koor, rec, Darst)     >
<!ELEMENT PF        - -  (Name, Groesse, Fix, koor, rec, Quelle,
                          Senke, MF, Attr, symp, Darst)             >
<!ELEMENT Name      - -  (#PCDATA)                                  >
<!ELEMENT Groesse   - -  (#PCDATA)                                  >
<!ELEMENT Fix       - -  (#PCDATA)                                  >
<!ELEMENT koor      - -  (#PCDATA)                                  >
<!ELEMENT rec       - -  (#PCDATA)                                  >
<!ELEMENT Darst     - -  (#PCDATA)                                  >
<!ELEMENT Attr      - -  (#PCDATA)                                  >
<!ELEMENT Quelle    - -  (#PCDATA)                                  >
<!ELEMENT Senke     - -  (#PCDATA)                                  >
<!ELEMENT MF        - -  ((Eintrag)*, letzter_Eintrag)              >
<!ELEMENT Eintrag   - -  (Zielfl, TMNummer, Dat1, Mz, Dat2, next,
                          MFS, MFZ                        )         >
<!ELEMENT letzter_Eintrag
                    - -  (Zielfl, TMNummer, Dat1, Mz, Dat2,
                          MFS, MFZ                        )         >
<!ELEMENT Zielfl    - -  (#PCDATA)                                  >
<!ELEMENT TMNr      - -  (#PCDATA)                                  >
```

```
<!ELEMENT Dat1      - -  (#PCDATA)                    >
<!ELEMENT Mz        - -  (#PCDATA)                    >
<!ELEMENT Dat2      - -  (#PCDATA)                    >
<!ELEMENT next      - -  (#PCDATA)                    >
<!ELEMENT MFS       - -  (#PCDATA)                    >
<!ELEMENT MFZ       - -  (#PCDATA)                    >
<!ELEMENT symp      - -  (#PCDATA)                    >
<!ELEMENT Darst     - -  (#PCDATA)                    >
```

Name	**Anfang**	**Skip**	**Datentyp**	**Ende**
Einst			String	(BF|RF|WF|PF)
BF	`$BF`	77		
RF	`$RF`	77		
WF	`$WF`	77		
PF	`$PF`	77		
Name	`Flaechenname :`	1	String	EOL
Groesse	`Flaechengroesse :`	1	Real	EOL
Fix	`Fixierung :`	1	Integer	EOL
koor	`Koordinaten :`	1	String	*Leerz.*+EOL
rec	`Rechtecke :`	1	String	*Leerz.*+EOL
Darst	`Darstellung :`	1	String	EOL
Attr	`Attribute :`	1	String	*Leerz.*+EOL
Quelle	`Quelle :`	1	String	EOL
Senke	`Senke :`	1	String	EOL
MF	`Materialfluss N:`	8		
Zielfl			String	*Leerz.*
TMNr			Integer	*Leerz.*
Dat1			String	`X`
Mz			Integer	*Leerz.*
Dat2			String	*Leerz.*
next	`&`		*kein*	
MFS	EOL+ `M_Start/M_Ziel :`		Punkt	
MFZ			Punkt	EOL
symp	`Sympathie :`	1	String	EOL
Darst	`Darstellung :`	1	String	EOL

Tabelle A.8: Datentypen in den Dateien <variante>.`lyt`

A.3 Dateistruktur DUMAS

Für die Übernahme von Daten aus LAPLAS werden keine eigenen Dateien erzeugt. Einzige Einschränkung ist, daß der Benutzer von LAPLAS vor einer Projektbearbeitung mit DUMAS explizit eine Materialflußdatei (`*.mfm`-Datei) erzeugen muß, um ein Einlesen der Daten durch DUMAS zu ermöglichen. Andere Informationen über ein Projekt erhält DUMAS offensichtlich durch Parsen der entsprechenden Layout-Datei (`*.lyt`-Datei) von LAPLAS.

Intern hält DUMAS eine Datei namens `dumas.kst`. Diese enthält die Kostendaten für Transportmittel und Hebezeuge, mithin Daten, die teilweise auch in LAPLAS gehalten werden.

Format: strukturierter Text
In einer Zeile befindet sich der Name eines Transportmittels oder Hebezeugs (Name endet auf Caret ^). In der nächsten Zeile finden sich dann jeweils die Kosten- und Zeitinformationen für das genannte Transportmittel/Hebezeug. Dabei gelten folgende Zuordnungen:

Transportmittel:

9 Felder pro Zeile, durch je 2 Leerzeichen getrennt:

- Fördergeschwindigkeit belastet	Real-Wert in ASCII-Darstellung
- Fördergeschwindigkeit leer	Real-Wert in ASCII-Darstellung
- Fixkosten	Real-Wert in ASCII-Darstellung
- Variable Kosten	Real-Wert in ASCII-Darstellung
- Zeitrichtwert Aufnahme	Integer-Wert in ASCII-Darstellung
- Zeitrichtwert Abgabe	Integer-Wert in ASCII-Darstellung
- Zeitzuschlag Kurve	Integer-Wert in ASCII-Darstellung
- Zeitzuschlag Steigung	Integer-Wert in ASCII-Darstellung
- Zeitzuschlag Tor	Integer-Wert in ASCII-Darstellung

Hebezeug:

7 Felder pro Zeile, durch je 2 Leerzeichen getrennt:

- Fördergeschwindigkeit Brücken	Real-Wert in ASCII-Darstellung
- Fördergeschwindigkeit Katzen	Real-Wert in ASCII-Darstellung
- Fördergeschwindigkeit Heben	Real-Wert in ASCII-Darstellung
- Fixkosten	Real-Wert in ASCII-Darstellung
- Variable Kosten	Real-Wert in ASCII-Darstellung
- Zeitrichtwert Aufnahme	Integer-Wert in ASCII-Darstellung
- Zeitrichtwert Abgabe	Integer-Wert in ASCII-Darstellung

Die Liste der Kosteninformationen endet mit dem Token „`** Ende **`“.

Beispiel:(`dumas.kst`):

```
TMA
5.0000000000E-01  1.0000000000E+00  2.0000000000E+00  7.0000000000E-02 30 30 6 6 4
TMB
1.1000000000E+00  1.5000000000E+00  2.5000000000E+00  7.0000000000E-02 12 10 7 7 4
B1KRAN^
7.0000000000E-01  5.0000000000E-01  3.0000000000E-01  1.1110000000E+01  3.5000000000E+01 300 200
B2KRAN^
7.0000000000E-01  5.0000000000E-01  3.0000000000E-01  2.0000000000E+01  3.5000000000E+01 150 200
** Ende **
```

Die SGML-Beschreibung gestaltet sich wie folgt (vgl. Tabelle A.9):

```
<!ELEMENT KOSTEN        - -  (Datensatz)+                    >
<!ELEMENT Datensatz     - -  (((THeader, TDaten)|
                              (HHeader, HDaten))+, Ende)  >
<!ELEMENT THeader       - -  (#PCDATA)                        >
<!ELEMENT TDaten        - -  (TGeschw, Fixk, Vark, TZeiten) >
<!ELEMENT HHeader       - -  (#PCDATA)                        >
<!ELEMENT HDaten        - -  (HGeschw, Fixk, Vark, HZeiten) >
<!ELEMENT Fixk          - -  (#PCDATA)                        >
<!ELEMENT Vark          - -  (#PCDATA)                        >
<!ELEMENT TGeschw       - -  (#PCDATA)                        >
<!ELEMENT TZeiten       - -  (#PCDATA)                        >
<!ELEMENT HGeschw       - -  (#PCDATA)                        >
<!ELEMENT HZeiten       - -  (#PCDATA)                        >
<!ELEMENT Ende          - -  (#PCDATA)                        >
```

Wird mit DUMAS ein Projekt abgespeichert, wird diese Datei komplett in das entstehende File mit der Endung `*.dms` übernommen (kopiert). In einem solchen File finden sich auch Informationen zu den Materialflüssen und alle anderen für DUMAS benötigten Informationen. Da nach Aussage des IAF diese Speicherfunktion des DUMAS nicht benötigt wird und sie auch nur zur Weiterbearbeitung eines unterbrochenen DUMAS-Projektes verwendet werden kann, wurde auf eine Modellierung der `*.dms`-Dateien verzichtet.

A.4 Datenimport und -export in TAYLOR

Die benötigten Daten für den Materialfluß werden aus FAST/pro mit Hilfe einer SQL-Anfrage erzeugt (s. Kapitel 5.1.1) und in einer Datei mit dem Namen

Name	Ende	Skip am Ende	Datentyp
THeader	[0-9|A-Z]+EOL		String
TDaten	EOL		
HHeader	∧+EOL		String
HDaten	EOL		
Fixk		*2 Leerzeichen*	Real
Vark		*1 Leerzeichen*	Real
TGeschw		*2 Leerzeichen*	String (Länge 34)
TZeiten	Element TDaten		String
HGeschw		*2 Leerzeichen*	String (Länge 52)
HZeiten	Element HDaten		String
Ende			Konstante „`** Ende **`“

Tabelle A.9: Datentypen der Datei `dumas.kst`

`taylor.txt` gespeichert. Die so gewonnenen Daten werden mittels eines weiteren in C++ für *DOS* geschriebenen Transformators in drei neue Dateien geteilt und umformatiert. Diese Dateien dienen dann als Importdateien für TAYLOR. Im folgenden soll das Format dieser Files kurz beschrieben werden. Die Dateinamen sind willkürlich, da der Import in TAYLOR ohnehin interaktiv angestoßen werden muß. Sie sind lediglich durch den z.Z. benutzten Transformator festgelegt.

1. `tay_ag.txt`
   Diese Datei enthält die Arbeitsgänge für jedes Werkstück.
   Format: stark strukturierter Text, 1...n Elemente pro Zeile, jeweils durch Komma getrennt. Ein Element ist eine Maschine (Integer in ASCII-Darstellung), eine Zeile ist die Reihenfolge der Arbeitsgänge an verschiedenen Maschinen, die für einen Auftrag ausgeführt werden müssen. Dabei sind die Aufträge nur implizit durchnumeriert, da eine Bestimmung der Reihenfolge oder eine Identifizierung durch TAYLOR nicht notwendig ist. Die in `taylor.txt` noch vorhandenen Namen bzw. Nummern der Arbeitsgänge, die in FAST/pro gehalten werden, gehen bei der Konvertierung verloren.

   Beispiel:

```
6,44,39,39,39,103,103
97,79,80,76,93,67,112,87,90,97,77,67,83,112
1,79,74,75,76,80,67,96,94,67,112
...
```

   Eine SGML-Beschreibung soll nur für diese Datei angegeben werden, für die anderen Dateien ist sie leicht ableitbar (vgl.Tabelle A.10):

```
<!ELEMENT ARBEITSGAENGE - -  (Arbeitsgang)+              >
<!ELEMENT Arbeitsgang   - -  (Maschine)+                 >
<!ELEMENT Maschine      - -  (#PCDATA)                   >
```

Name	Trennzeichen	Datentyp
Arbeitsgang	EOL	
Maschine	,	String

Tabelle A.10: Datentypen der Dateien `tay_ag.txt`

2. `tay_bz.txt`
   Diese Datei enthält die Bearbeitungszeiten für jeden Arbeitsgang an jedem Werkstück.
   Format: stark strukturierter Text, 1...n Elemente pro Zeile, jeweils durch Komma getrennt. Dabei bedeutet ein Wert in einer bestimmten Zeile und einer bestimmten Spalte die Bearbeitungszeit für *den* Arbeitsgang, dessen Nummer an der gleichen Position in `tay_ag.txt` vermerkt ist.
   Beispiel:

```
.012152778,.011111111,.021444444,.016909722,.007125000,.000000694,.000000694
.052083333,.010347222,.006775694,.004583333,.002083333,.000000694,.000000694,
   .003125000,.011527778,.003750000,.005861111,.000000694,.001215278,.000000694
.001295139,.014652778,.014009722,.011300694,.014922222,.058512500,.000000694,
   .035844444,.027690972,.000000694,.000000694
...
```

3. `tay_rz.txt`
   Diese Datei enthält die Rüstzeiten für jeden Arbeitsgang an jedem Werkstück.
   Format: wie vor
   Beispiel:

```
0.000000000,.020833333,.131944444,.125000000,.083333333,0.000000000,0.000000000
.010416667,.055555556,.055555556,.041666667,.006944444,0.000000000,0.000000000,
   .006944444,.013888889,.013888889,.069444444,0.000000000,.020833333,0.000000000
.006944444,0.000000000,.086805556,.069444444,.166666667,.128472222,0.000000000,
   .006944444,.006944444,0.000000000,0.000000000
```

4. `<name>.plt`
   Der Hallengrundriß im HPGL-Format kann ebenfalls importiert werden. LAPLAS ist in der Lage, eine entsprechende Datei zu erzeugen. Hinweise zum HPGL-Format finden sich in [Bor92].

TAYLOR kann, wie bereits erwähnt, mit Hilfe der Exportfunktion beliebige interne Variablen zur Laufzeit in eine Datei schreiben. Einziger Standard für solche Dateien ist eine kommagetrennte Datenstruktur ähnlich der Importdateien. Da im FACTOTUM keine feststehenden Funktionen für die Erzeugung solcher Exportdateien definiert sind, sondern diese je nach Bedarf erzeugt werden, ist eine semantische Modellierung nicht möglich.

Intern verwendet TAYLOR, analog zu DUMAS, ein proprietäres Dateiformat, das einzig dem Zweck dient, ein Zwischenspeichern von TAYLOR-Projekten zu ermöglichen. Da hier ein binäres Dateiformat vorliegt, dessen Analyse nicht erfolgreich war, konnte keine Integration erfolgen. Somit ist eine Betrachtung dieser Dateien nicht nötig.

Anhang B

Attributinformationen für die Komponentenschemata

Im folgenden finden sich tabellarische Aufstellungen der Attribute der einzelnen Klassen in den FACTOTUM-Komponentensystemen. Sie ergänzen die in Kapitel 6 angegebenen Komponentenschemata. Die aufgrund der anfänglichen ER-Modellierung auftretenden Schlüsselattribute sind jeweils kursiv gesetzt. Der Datentyp „Punkt“ besteht aus zwei `real`-Werten, die X- und Y-Koordinaten bedeuten. Der Datentyp „binär“ bezeichnet Daten, deren genaue Bedeutung für das FDBS irrelevant ist. Daher ist im FDBS nur Speicherplatz für diese Daten bereitzustellen, bzw. eine Abspeicherung der Daten als FDBS-Attribute unnötig. Dies könnte evtl. als Filterung im Sinne eines Exportschemas verstanden werden.

Bemerkungen zu Tabelle B.2:

Alle Dateien des Systems LAPLAS befinden sich im Arbeitsverzeichnis von LAPLAS.

Der String <projekt> ist der mit Unterstrichen auf 6 Zeichen aufgefüllte Name des Projekts (z.B. `cebit_`).

Der String <variante> wird ermittelt aus dem Projekt, zu dem die Variante gehört. Dazu wird an den Namen des Projekts (String <projekt>) eine zweistellige Zahl in ASCII-Kodierung angehängt, die sich aus der Stellung des zur Variante gehörenden Eintrags in der Datei <projekt>.`var` ergibt. Die erstgenannte Variante in einem Projekt namens `CEBIT` hieße demzufolge `cebit_01.lyt`.

Bemerkungen zu Tabelle B.3:

Das Attribut DXF wurde eingeführt, um einen Datenaustausch mit AutoCAD im DXF-Format auf der Grundlage von Im- und Exportfunktionen von LAPLAS modellieren zu können. In Form der Attribute koor und rec liegen diese Informationen nochmals vor, allerdings nicht in für AutoCAD auswertbarer Form.

Ein Umwandlung dieser Formate ineinander durch das FDBMS erschien aus Gründen des Aufwands und der aus einer automatischen Konvertierung resultierenden Komplizierung der Bedienung von LAPLAS nicht gerechtfertigt.

Bemerkung zu Tabelle B.5:

Die Referenz auf TM.mf_ref ist keine durch Normalisierung entstandene GIM-Referenz. Hier ist für die Existenz eines Objekts in TM die Existenz eines entsprechenden Verweises aus MF nicht nötig. Das Attribut TM.mf_ref enthält vielmehr lediglich eine laufende Nummer des Transportmittel, auf die von der Klasse MF aus zugegriffen wird.

Bemerkungen zu Tabelle B.6:

Die Dateien <variante>.`lyt` und <variante>.`mfm` befinden sich im LAPLAS-Arbeitsverzeichnis.
Die Datei `dumas.kst` befindet sich im DUMAS-Verzeichnis.

Bemerkung zu Tabelle B.7:

Die Datei `taylor.txt` befindet sich im FAST/pro-Verzeichnis. Um mit TAYLOR Daten exportieren zu können, sind diese wie in Abschnitt A.4 beschrieben umzuwandeln.

Bemerkung zu Tabelle B.8:

Die verwendeten Dateien befinden sich im LAPLAS-Verzeichnis, da LAPLAS nicht in der Lage ist, auf andere Verzeichnisse zuzugreifen. Der Name der Datei <name>.`dxf` ist willkürlich, da der Benutzer von LAPLAS den Import dieser Dateien ohnehin manuell vornehmen muß. Eventuell könnte ein fester Dateiname für alle Importoperationen festgelegt werden.

Klasse MATERIALFLUSSMATRIX (MFM)			
Name	**Bedeutung**	**Typ**	**Datei**
NAU	*Nummer Auftrag*	char	`mfm*.txt`
NAG	*Nummer Arbeitsgang*	char	
PNAS_vor	vorige Maschine	number	
PNAS_nach	folgende Maschine	number	
PNAS	aktuelle Maschine	number	
TM	Transportmittel	char	
Mz	Menge	number	

Klasse TAYLOR_ZEITEN (T_Z)			
Name	**Bedeutung**	**Typ**	**Datei**
NAU	*Nummer Auftrag*	char	`taylor.txt`
NAG	*Nummer Arbeitsgang*	char	
PNAS	aktuelle Maschine	number	
ZEP	Planeinzelzeit	number	
ZRP	Planrüstzeit	number	

Tabelle B.1: Attribute der FAST/pro-Klassen

Klasse PROJEKT (PRJ)			
Name	**Bedeutung**	**Typ**	**Datei**
PNR	*Projektnummer*	number	`projekte.dat`
PNM	*Projektname*	char	`projekte.dat`
Farben	Farbeinstellungen	binär	<projekt>`.far`
Konfig	Konfigurationseinstellungen	binär	<projekt>`.gwt`
var_ref	Referenz auf VAR	GIM-Referenz	

Klasse VARIANTE (VAR)			
Name	**Bedeutung**	**Typ**	**Datei**
prj_ref	Referenz auf PRJ	GIM-Referenz	
NR	*Variantennummer*	number	<projekt>`.var`
Name	*Variantenname*	char	<projekt>`.var`
Attr	Definition von Attributen	binär	<variante>`.att`
Einst	Einstellungen	binär	<variante>`.lyt`
pf_ref	Referenz auf PF	GIM-Referenz	
wf_ref	Referenz auf WF	GIM-Referenz	
rf_ref	Referenz auf RF	GIM-Referenz	
bf_ref	Referenz auf BF	GIM-Referenz	

Tabelle B.2: Attribute der LAPLAS-Klassen PRJ und VAR

Klasse BASISFLÄCHE (BF)			
Name	**Bedeutung**	**Typ**	**Datei**
var_ref	Referenz auf VAR	GIM-Referenz	
Name	*Flächenname*	char	<variante>.lyt
Groesse	Größe der Fläche	number	<variante>.lyt
Fix	Fixierung	number	<variante>.lyt
koor	Flächenkoordinaten	binär	<variante>.lyt
rec	Punktkoordinaten für Flächen	binär	<variante>.lyt
Darst	Darstellungsattribute	binär	<variante>.lyt
Klasse ATTRIBUTFLÄCHE (RF)			
Name	**Bedeutung**	**Typ**	**Datei**
var_ref	Referenz auf VAR	GIM-Referenz	
Name	*Flächenname*	char	<variante>.lyt
Groesse	Größe der Fläche	number	<variante>.lyt
koor	Flächenkoordinaten	binär	<variante>.lyt
rec	Punktkoordinaten für Flächen	binär	<variante>.lyt
Darst	Darstellungsattribute	binär	<variante>.lyt
Attr	Attribute	binär	<variante>.lyt
Klasse WEGEFLÄCHE (WF)			
Name	**Bedeutung**	**Typ**	**Datei**
var_ref	Referenz auf VAR	GIM-Referenz	
Name	*Flächenname*	char	<variante>.lyt
Groesse	Größe der Fläche	number	<variante>.lyt
Fix	Fixierung	number	<variante>.lyt
koor	Flächenkoordinaten	binär	<variante>.lyt
rec	Punktkoordinaten für Flächen	binär	<variante>.lyt
Darst	Darstellungsattribute	binär	<variante>.lyt

Tabelle B.3: Attribute der LAPLAS-Klassen BF, RF und WF

Klasse PLANUNGSFLÄCHE (PF)			
Name	**Bedeutung**	**Typ**	**Datei**
var_ref	Referenz auf VAR	GIM-Referenz	
Name	*Flächenname*	char	<variante>.lyt
Groesse	Größe der Fläche	number	<variante>.lyt
Fix	Fixierung	number	<variante>.lyt
DXF	Darstellung der Maschine	DXF	<name>.dxf
koor	Flächenkoordinaten	binär	<variante>.lyt
rec	Punktkoordinaten für Flächen	binär	<variante>.lyt
Darst	Darstellungsattribute	binär	<variante>.lyt
Attr	Attribute	binär	<variante>.lyt
Quelle	Quelle für Materialfluß	Punkt	<variante>.lyt
Senke	Senke für Materialfluß	Punkt	<variante>.lyt
mf_ref	Referenz auf MF	GIM-Referenz	
symp	Sympathien	binär	<variante>.lyt

Tabelle B.4: Attribute der LAPLAS-Klasse PF

Klasse MATERIALFLUSS (MF)			
Name	**Bedeutung**	**Typ**	**Datei**
pf_ref	Referenz auf PF	GIM-Referenz	
Zielfl	*Name der Zielfläche*	char	<variante>.lyt
tm_ref	Referenz auf TM	number	
Mz	*Intensität („Mengenzahl“)*	number	<variante>.lyt
Einst	*Einstellungen*	binär	<variante>.lyt
MFS	*Materialflußstart*	Punkt	<variante>.lyt
MFZ	*Materialflußziel*	Punkt	<variante>.lyt
Klasse Transportmittel (TM)			
Name	**Bedeutung**	**Typ**	**Datei**
mf_ref	Referenz auf MF	number	
Name	*Name des Transportmittels*	char	laplasii.kst
Hilf	Transporthilfsmittel	boolean	
Kap	Transportkapazität	number	
Fix_k	Fixkosten	number	
Var_k	variante Kosten	number	

Tabelle B.5: Attribute der LAPLAS-Klassen MF und TM

Klasse MASCHINENPOSITION (MP)			
Name	**Bedeutung**	**Typ**	**Datei**
Name	*Name der Maschine*	char	<variante>.lyt
Quelle	Quelle für Materialfluß	Punkt	<variante>.lyt
Senke	Senke für Materialfluß	Punkt	<variante>.lyt
Klasse TRANSPORT (Tr)			
Name	**Bedeutung**	**Typ**	**Datei**
QName	*Start des Transports*	char	<variante>.mfm
ZName	*Ziel des Transports*	char	<variante>.mfm
TM_Name	*Name des Transportmittels*	char	<variante>.mfm
Mz	Transportmenge	number	<variante>.mfm
Entf	Entfernung	number	<variante>.mfm
Klasse TRANSPORTMITTELBIBLIOTHEK (TMB)			
Name	**Bedeutung**	**Typ**	**Datei**
TM_name	*Name des Transportmittels*	char	dumas.kst
Fixk	Fixkosten	number	dumas.kst
Vark	variante Kosten	number	dumas.kst
Geschw	Geschwindigkeiten	char	dumas.kst
Zeiten	Transportzeiten	char	dumas.kst

Tabelle B.6: Attribute der Klassen in DUMAS

Klasse TAYLOR-ZEITEN (T_Z)			
Name	**Bedeutung**	**Typ**	**Datei**
PNAS	aktuelle Maschine	number	taylor.txt
NAU	*Nr_Auftrag*	char	
NAG	*Nr_Arbeitsgang*	char	
ZEP	Planeinzelzeit	number	
ZRP	Planrüstzeit	number	

Tabelle B.7: Attribute der Klassen in TAYLOR

Klasse MASCHINE (MA)			
Name	**Bedeutung**	**Typ**	**Datei**
DXF	Maschinengeometrie	DXF	<name>.dxf

Tabelle B.8: Attribute der Klassen in AutoCAD

Anhang C

Beispiel für ein SGML-Dokument

Zur Verdeutlichung der Vorgehensweise bei der SGML-Beschreibung von Dokumenten soll ein vollständiges Beispiel für eine einfache SGML-Dokumenttypdefinition und eine dazu passende Dokumentinstanz angegeben werden. Das Beispiel wurde aus [vH90] übernommen und leicht vereinfacht.

Dokumenttypdefinition

Die verwendeten SGML-Konstrukte `ELEMENT` und `ATTLIST` wurden in Kapitel 3.3 erläutert. Ein Kommentar wird mit der Zeichenfolge `<!--` eingeleitet und mit `-->` abgeschlossen. Zur Bedeutung der „Minimierung“ (`MIN`) siehe [vH90]. Die DTD definiert einen Dokumenttyp `Memo`, der aus einer Einleitung, einem Hauptteil und einem Ende besteht. Die Einleitung beinhaltet einen Empfänger und einen Absender, die beide angegeben werden müssen. Der Hauptteil besteht aus Absätzen (*paragraph*), die an beliebigen Stellen von Zitaten (*quotations*) unterbrochen sein können. Als Ende folgt optional eine schließende Bemerkung (*close*).

```
<!-- DTD for simple office memoranda                          -->
<!--       ELEMENTS    MIN CONTENT                            -->
<!ELEMENT  Memo        - - ((To & From), Body, Close?)          >
<!ELEMENT  To          - O (#PCDATA)                            >
<!ELEMENT  From        - O (#PCDATA)                            >
<!ELEMENT  Body        - O (P*)                                 >
<!ELEMENT  P           - O (#PCDATA|Q)                          >
<!ELEMENT  Q           - - (#PCDATA)                            >
<!ELEMENT  Close       - O (#PCDATA)                            >
<!--       ELEMENTS NAME   VALUE                DEFAULT       -->
<!ATTLIST  Memo     status (confiden|public)    public          >
```

Ein Dokument dieses Typs kann mit Hilfe des für `Memo` definierten Attributs `status` als vertraulich (*confidential*) oder öffentlich (*public*) deklariert werden.

Dokumentinstanz

Eine Dokumentinstanz, die die Vorgaben dieser Dokumenttyp-Definition erfüllt, könnte folgendermaßen aussehen:

```
<Memo status=confiden>
<To>Comrade Napoleon</To>
<From>Snowball</From>
<Body>
<P>In Animal Farm, George Orwell says:<Q>...the pigs had to
expend enormous labour every day upon mysterious things called
files, reports, minutes and memoranda. These were large sheets of
paper which had to be closely covered with writing, and as soon
as they were so covered, they were burnt in the
furnace...</Q>. Do you think SGML would have helped the pigs?
</P>
</Body>
<Close>Comrade Snowball</Close>
<Memo>
```

Ein solcher „Quelltext“ kann nun mit Hilfe eines SGML-Parsers in ein formatiertes Textdokument übersetzt werden. Für die angegebene Dokumentinstanz könnte ein Dokument ähnlich dem folgenden erzeugt werden:

Memo
top secret

To: Comrade Napoleon
From: Snowball

In Animal Farm, George Orwell says:
“*...the pigs had to expend enormous labour every day upon mysterious things called files, reports, minutes and memoranda. These were large sheets of paper which had to be closely covered with writing, and as soon as they were so covered, they were burnt in the furnace...*”.
Do you think SGML would have helped the pigs?

Comrade Snowball

Die Formatierungen sind dabei nicht im SGML-Dokument definiert, sondern werden von der Anwendung erzeugt, die das Dokument liest. So wird z.B. die Zeile

top secret

vom SGML-Parser erzeugt, wenn das Attribut `status` des Memos den Wert `confiden` besitzt.

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 12. September 1996

Andreas Köller

Thesen

1. Softwareprodukte, die Daten in Dateien speichern und keine Datenbankfunktionalität anbieten, sind wichtige Komponenten vorhandener Systemlösungen in verschiedenen Anwendungsbereichen. Daher besteht die Notwendigkeit einer Einbindung solcher Komponenten in integrierte Datenbanksysteme.

2. Die von dateibasierten Datenhaltungskomponenten verwendeten Dateiformate sind zu einem großen Teil so aufgebaut, daß eine Erkennung ihrer Struktur und der Bedeutung dort enthaltener Datenelemente möglich ist.

3. Der Ansatz föderierter Datenbanksysteme bildet ein hinreichend flexibles Konzept, um die Integration auch solcher „Nicht-Datenbanksysteme“ zu ermöglichen und dabei eine ausreichende Geschwindigkeit und Sicherheit der Datenverarbeitung zu gewährleisten.

4. Die Einbindung von auf *MS-DOS* basierenden Applikationen in föderierte Datenbanksysteme ist nur unter Verwendung eines echt *multitasking*-fähigen Betriebssystems wie UNIX möglich, wobei praktische Probleme dieser Anbindung wie z.B. das Abfangen von Dateizugriffen durch das DOS-System lösbar sind.

5. Die Integration der Komponenten des Produktionsplanungssystems FACTOTUM in ein föderiertes Datenbanksystem stellt bei Verfügbarkeit einer entsprechenden FDBS-Lösung eine vom Aufwand her akzeptable Aufgabe dar, durch die sich entscheidende Verbesserungen in Bedienung, Konsistenz und Vielseitigkeit dieses Produktes erzielen lassen.