Universität-Gesamthochschule Paderborn
Fachbereich 17 - Mathematik/Informatik

Ein middlewarebasierter, generativer Ansatz zur Unterstützung von Schemaevolution in komplexen Ingenieurinformationssystemen

Diplomarbeit
für den integrierten Studiengang Informatik
im Rahmen des Hauptstudiums II

von

Markus Westerfeld
Salentinstraße 2a
33102 Paderborn

vorgelegt bei

Prof. Dr. Wilhelm Schäfer
AG Softwaretechnik

und

Prof. Dr. Gregor Engels
AG Datenbanken und Informationssysteme

Paderborn, im Mai 2000

Danksagung

Diese Arbeit wurde am Lehrstuhl Softwaretechnik der Universität-Gesamthochschule Paderborn erstellt. Mein Dank gilt Herrn Dr. rer. nat. Jens Jahnke für die interessante Aufgabenstellung und seine wertvollen Anregungen und Hinweise, sowie Herrn Prof. Dr. W. Schäfer für die begleitende Betreuung und Herrn Prof. Dr. G. Engels für die Zweitkorrektur der Arbeit. Außerdem danke ich allen, die mir im Verlauf dieser Arbeit zur Seite standen.

Eidesstattliche Erklärung

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Paderborn, den 31. Mai 2000

_ _ _ _ _ _ _ _ _ _ _

(Markus Westerfeld)

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Der Einsatz von Rechnern zur Unterstützung und Verwaltung großer Datenmengen hat auch vor den Ingenieurwissenschaften keinen Halt gemacht. Der Schwerpunkt des Rechnereinsatzes liegt hier ganz klar in der Unterstützung der Ingenieure bei ihrer Arbeit und Forschung. Wichtige Einsatzgebiete sind hierbei die Simulation komplexer Vorgänge, aufwendige Modell-Berechnungen im zwei- und dreidimensionalen Bereich, sowie die Verwaltung der daraus gewonnenen Daten.

Dem Fachbereich Maschinentechnik der Universität Paderborn ist seit einigen Jahren das *Institut für Kunststofftechnik (KTP)* angegliedert. Ein wesentliches Forschungsgebiet ist die Entwicklung und Optimierung von Plastifiziereinheiten. Diese Maschinen werden in weiten Teilen der kunststofferzeugenden und kunststoffverarbeitenden Industrie eingesetzt.

Im Rahmen der Forschung sind für die Plastifiziereinheiten Simulationsprogramme zur Berechnung und Darstellung wichtiger Verfahrenskenngrößen entwickelt worden. Neben der Berücksichtigung der Maschinenbauteile, wie Zylinder, Schnecke oder Werkzeug, spielt das zu verarbeitende Material eine wichtige Rolle.

Das Material wird durch eine Vielzahl von Einzelkriterien, den Materialdaten, beschrieben. Diese decken verschieden Bereiche, wie geometrische Angaben, physikalische oder chemische Eigenschaften ab.

Aus der Problematik, daß viele der Materialien in mehr als einem der Simulationsprogramme zum Einsatz kommen, entstand der Bedarf nach einem programmübergreifenden Verwaltungs- und Informationssystem *(EvolMat*[1]*)*. Dieses sollte nicht nur bestehende Daten verwalten, sondern auch neue Daten aufnehmen, analysieren und für verschiedene Simulationsprogramme aufbereiten können.

Wird ein solches System für die tägliche Arbeit eingesetzt, lassen sich viele Vorteile daraus gewinnen. Werden die Materialdaten in einem Datenbank-Management-System (DBMS) abgelegt, läßt sich die Redundanz deutlich verringern, bzw. sogar ganz beseitigen. Sind die Daten für ein Material erst einmal erfaßt, können sie anderen Anwendungen zur Verfügung gestellte werden, ohne das eine erneute Eingabe erforderlich ist. Desweiteren lassen sich mit Hilfe eines Verwaltungssystems die Daten vielfältig analysieren und entsprechende Ergebnisse interpretieren.

---

1. EVOLutionsorientiertes MATerialverwaltungssystem

Um die gewünschte Funktionalität zu erreichen, benötigt der Anwender Unterstützung bei der Erfassung der Materialdaten, sowie die Möglichkeit einer effizienten Speicherung der Daten.

Ziel der Diplomarbeit ist die Konzeption eines Werkzeugs, mit dem sich die Materialdaten erfassen und verwalten lassen. Zur Unterstützung des Entwicklungsprozeß wird das am Lehrstuhl für Softwaretechnik der Universität Paderborn entwickelte Reengineering-Werkzeug *VARLET* (Verified Analysis and Reengineering of Legacy Database Systems Using Equivalence Transformations) eingesetzt und um zusätzliche Fähigkeiten erweitert. Zu den Erweiterungen gehören unter anderem die Implementierung des standardisierten Datenaustauschverfahrens XMI (*XML* Metadata Interchange), sowie die Generierung von Schnittstellendefinitionen für die Anbindung einer Middleware (*ObjectDRIVER*).

Die zu verwaltenden Materialdaten unterscheiden sich nicht nur zwischen den einzelnen Simulationsprogrammen, sondern auch innerhalb einer Simulationssoftware. Dies ist bedingt durch verschiedene Programmversionen, die während der einzelnen Projektphasen entstanden sind. Diese Problematik der schrittweisen Änderung der Datenmodelle ist im Bereich des Softwareentwurfs als Schemaevolution bekannt. Um den Änderungsaufwand möglichst gering und anwenderfreundlich zu halten, soll die Anpassung der Datenmodelle und der Datenbasis möglichst automatisch geschehen. Dabei sollen die Änderungen der Datenmodelle mit Hilfe von *VARLET* durchgeführt werden.

1.2 Aufbau der Arbeit

Im weiteren Verlauf der Arbeit wird zunächst die Anwendungsdomäne beleuchtet. Dabei wird zuerst das Institut und seine Forschungsschwerpunkte im Bereich der Kunststofftechnik sowie die grundlegende Funktionsweise der verwendeten Maschinen vorgestellt. Danach folgt eine Beschreibung der Simulationsprogramme, die im KTP entwickelt wurden. Diese Programme stellen die Grundlage für die Idee der Diplomarbeit dar, ein Verwaltungs- und Informatonssystem für Materialdaten zu entwickeln.

In Kapitel 3 wird das Konzept für das Informationssystem EvolMat entworfen. Beispiele der Dateneingabe in den Simulationsprogrammen und die Berücksichtigung des späteren Einsatzbereiches sowohl im industriellen wie auch universitären Bereich fließen dabei in die Zusammenstellung der Anforderungen mit ein. Die Analyseergebnisse sollen die Designentscheidungen bezüglich der Datenstrukturen und Anwendungsspezifikation erleichtern.

Im Kapitel „Grundlagen“ werden zwei verwandte Arbeiten vorgestellt, und die gemeinsamkeiten und Unterschiede zur vorliegenden Arbeit herausgearbeitet. Desweiteren werden in diesem Kapitel die in der Arbeit verwendeten Konzepte und Methoden vor-

gestellt. Dazu zählen der Einsatz einer Zugriffsschicht und des Reengineering-Werkzeugs VARLET, die Sichtenintegration und Schemaevolution sowie die Generierung von XML-Code.

Der Aufbau des Verwaltungs- und Informationssystems EvolMat beginnt mit der Entwicklung eines Gesamtkonzeptes. Dies beinhaltet die Schnittstellen zu den Simulationsprogrammen, sowie die Modellierung der Datenstrukturen und die Anbindung an eine Datenbank.

Dazu werden zuerst die zu erfassenden Daten analysiert und klassifiziert, um daraus die Schemata für die einzelnen Simulationsprogramme zu entwickeln. Diese werden dann im nächsten Schritt zu einem globalen Schema integriert. Die gesamte Modellierung wurde mit Hilfe objektorientierte Methoden vorgenommen, wodurch Lesbarkeit und Wartbarkeit der Datenstrukturen deutlich verbessert wurden. Für den praktischen Einsatz von EvolMat ist jedoch die Möglichkeit zur Nutzung vorhandener Datenbanksysteme sehr wichtig. Dem weitverbreiteten Einsatz relationaler Datenbanksysteme (RDBMS) wird durch die Verwendung einer objektorientierten Zugriffsschicht Rechnung getragen. Sie stellt das Bindeglied zwischen dem RDBMS und der Anwendung dar.

Das Informationssystem EvolMat soll nicht nur die Verwaltung und Bereitstellung der Materialdaten für die Simulationssoftware übernehmen, sondern auch auf Änderungen der Datenstrukturen und damit des Datenbankschemas reagieren können. Diese Änderungen betreffen nur das zu Beginn des Kapitels entwickelte Gesamtschema und werden auch nur an diesem vorgenommen. Sich daraus ergebende Änderungen für die Schnittstellen der Anwendungsprogramme werden von EvolMat durch Anpassung von Exportfiltern berücksichtigt. Die aufgezeigten Theorien und Ergebnisse werden zum Abschluß des Kapitels anhand von Beispielen erläutert.

Das letzte Kapitel bietet eine Zusammenfassung der im Rahmen der Arbeit gewonnenen Ergebnisse. Desweiteren wird hier ein Ausblick auf mögliche Weiterentwicklungen und Verbesserungen gegeben, wie z.B. eine verbesserte Generierung der Exportfilter und mehr Funktionalität der Beispielanwendung.

2 Anwendungsdomäne

2.1 Das Institut für Kunststofftechnik

Seit April 1980 existiert im Fachbereich Maschinentechnik das *Institut für Kunststofftechnik (KTP)* unter der Leitung von Prof. Dr.-Ing. Helmut Potente. Ziel ist die Weiterentwicklung kunststofftechnischer Verarbeitungsprozesse. Aus der intensiven Zusammenarbeit mit der Industrie entstand 1993 als zusätzliches Bindeglied zwischen KTP und Industrie der *Verein zur Förderung der Kunststofftechnik*. Wichtige Aufgaben des Vereins sind der Wissenstransfer vor allem in mittelständische Unternehmen und die Integration praxisorientierter Probleme in die Forschungsarbeit des KTP.

Die Forschungsschwerpunkte des KTP liegen auf den folgenden Gebieten:

- Optimierung von Prozessen der Kunststoffverarbeitung
- Entwicklung und Anwendung neuer Verarbeitungsverfahren
- Qualitätssicherung in der Fertigung
- Modellbildung und Entwicklung passender Software

Der wichtigste Forschungsbereich des KTP ist die Erarbeitung mathematisch-physikalischer Grundlagen zum Energie- und Stofftransport in Schneckenmaschinen. Das zum KTP gehörende Verarbeitungslabor ist mit mehreren Spritzgießmaschinen, Einschnekkenextrudern und Doppelschneckenextrudern ausgestattet. Diese Ausstattung ist Grundlage für die experimentelle Überprüfung der entwickelten Berechnungsmethoden. Die Methoden sind Basis für die Auslegung von Schneckenmaschinen und bilden in vielen Industrieunternehmen die Grundlage der Entwicklung. Ziel der aktuellen Forschung ist letztendlich die Erstellung einer Software, die zumindest in Teilbereichen eine selbsttätige Optimierung des Verarbeitungsprozesses ermöglicht. Der Prozeß der Selbstoptimierung erfolgt durch die Variation der Parameter, die den Plastifizierprozeß beeinflussen. Dazu gehören bekannte physikalische Größen wie z.B. Druck oder Temperatur, aber auch spezifische Eigenschaften des verarbeiteten Materials.

Durch die langjährige Forschung auf dem Gebiet der Schneckenberechnung sind die theoretischen Modelle so weit entwickelt worden, daß eine industrielle Nutzung der Ergebnisse möglich ist. Zu diesem Zweck wurden am KTP mehrere Simulationsprogramme entwikkelt, die ständig dem aktuellen Forschungsstand angepaßt werden. Alle Simulationsprogramme sind im Rahmen von Gemeinschaftsforschungsprojekten zwischen dem KTP und mehreren Industrieunternehmen entstanden. Zu den wichtigsten Teilnehmern zählen namhafte Hersteller aus dem Maschinenbau, wie Krauss-Maffei, Battenfeld oder Arburg, sowie große Chemieunternehmen, z.B. die Bayer AG.

2.1.1 Beschreibung der Spritzgießmaschine

Das grundlegende Funktionsprinzip ist bei den verschiedenen Schneckenmaschinen gleich. Die Unterschiede liegen in der Anzahl der in der Maschine eingebauten Schnekken (Ein- bzw. Doppelschneckenextruder) sowie im Ablauf des Plastifizierprozesses. Dabei wird zwischen den beiden Verfahren *Extrusion* und *Spritzgießen* unterschieden.

Der Extrusionsprozeß liefert kontinuierlich eine konstante Masse an aufgeschmolzenem Material. Dieses Verfahren wird zur Produktion von Halbzeugen, wie Folienschläuchen, Profilen oder Rohren benutzt. Im Gegensatz dazu ist das Spritzgießen ein diskontinuierliches Verfahren zur Herstellung von Einzelteilen. Die Maschine liefert in einem festen Zyklus eine bestimmte Materialmasse, die dann in eine Werkzeugform eingespritzt wird. Die Produktgrößen reichen von Bruchteilen von Millimetern bis hin zu kompletten Auto-Halbschalen.

In den Schneckenmaschinen werden polymere Werkstoffe zu den unterschiedlichsten Teilen verarbeitet. Umfangreiche Kenntnisse über den Plastifiziervorgang polymerer Werkstoffe sind die Grundvorraussetzung für eine optimale Nutzung der Maschinen. Ziele der Optimierung sind die Steigerung der Produktqualität und des Durchsatzes einer Plastifiziereinheit, also die geförderte Masse pro Zeiteinheit. Entscheidungen zur Optimierung werden dabei bis heute oft aufgrund von Erfahrungswerten und experimentellen Untersuchungen getroffen. Dieses „trial and error“ Verfahren ist nicht nur sehr zeitaufwendig, sondern auch sehr kostspielig. Die hohen Kosten entstehen bei der Herstellung der Schnecken. Diese sind, gerade bei Neuentwicklungen, häufig Unikate und liegen bei den Herstellungskosten leicht oberhalb der 100.000 DM. Die Entwicklung der Schnecken findet, sofern sie nicht durch CAD/CAM Anwendungen unterstützt wird, am Zeichentisch statt. So lassen sich die geometrischen Eigenschaften und eventuelle Fehlkonstruktionen erst nach der Herstellung überprüfen.

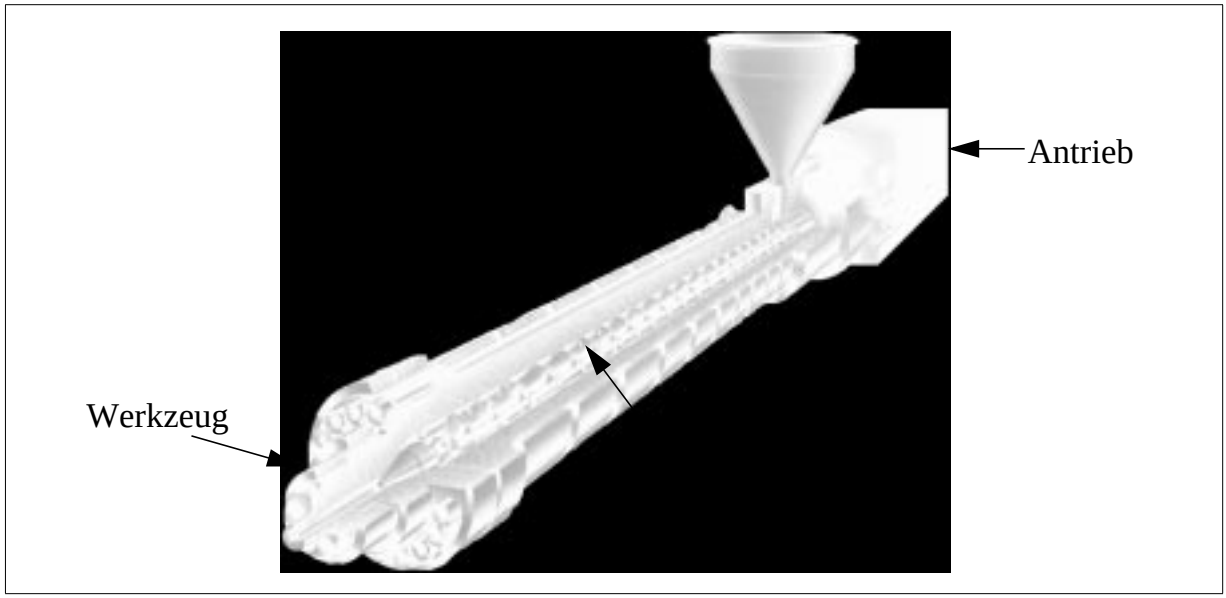


Abbildung 1 Modell einer Spritzgießmaschine

Das Funktionsprinzip einer Schneckenmaschine wird beispielhaft an Abbildung 1 erläutert. Das zu verarbeitende Material (meist in Form eines Granulates) wird über den Trichter in die Schecke eingerieselt. Durch den am hinteren Ende der Schnecke angebrachten Antriebsmotor wird die Schnecke gedreht, wodurch das Material in Richtung Schneckenspitze transportiert wird. Der Aufbau der Schnecke mit den unterschiedlich geformten Schnekkengängen und die starke Reibung an der Zylinderinnenwand bewirken ein aufreiben und aufschmelzen des Materials. Das Aufschmelzen kann zusätzlich durch außen am Zylinder angebrachte Heizbänder verstärkt und Temperaturvorgaben reguliert werden. Läuft der Verarbeitungsprozeß optimal ab, hat das Material an der Schneckenspitze die richtige Temperatur und Viskosität um zu den diversen Zwischen- und Endprodukten weiterverarbeitet zu werden.

2.2 Beschreibung der Simulationsprogramme

Jedes der hier vorgestellten Simulationsprogramme weist einige besondere Fähigkeiten auf, da es für einen bestimmten Anwendungsbereich entwickelt wurde. Es lassen sich aber auch allgemeinere Vorteile, die sich durch den Einsatz der Simulationsprogramme ergeben, aufzeigen.

Wie bereits zuvor erwähnt, ist die Herstellung der Schnecken oft sehr kostenintensiv und nimmt einige Zeit in Anspruch. Hier können durch den Einsatz der Simulationssoftware sowohl die Kosten, als auch die Entwicklungszeit erheblich reduziert werden. Durch die Möglichkeit, alle wichtigen Kenngrößen des Verarbeitungsprozesses, wie z.B. der Temperatur- oder Aufschmelzverlauf über der Schnecke (siehe auch Abbildung 4) oder die Drehzahl, zu variieren, ergibt sich eine größere Transparenz des gesamten Prozesses für den Kunststoffverarbeiter. Die verwendeten Verfahren zur Berechnung der Kenngrößen basieren weitestgehend auf einfachen analytischen Verfahren, so daß sich recht kurze Simulationszeiten ergeben. Eine vollständige Simulation, bei der ca. 50 verschiedene Geometrie bzw. Parameter berechnet werden dauert nur wenige Minuten[1].

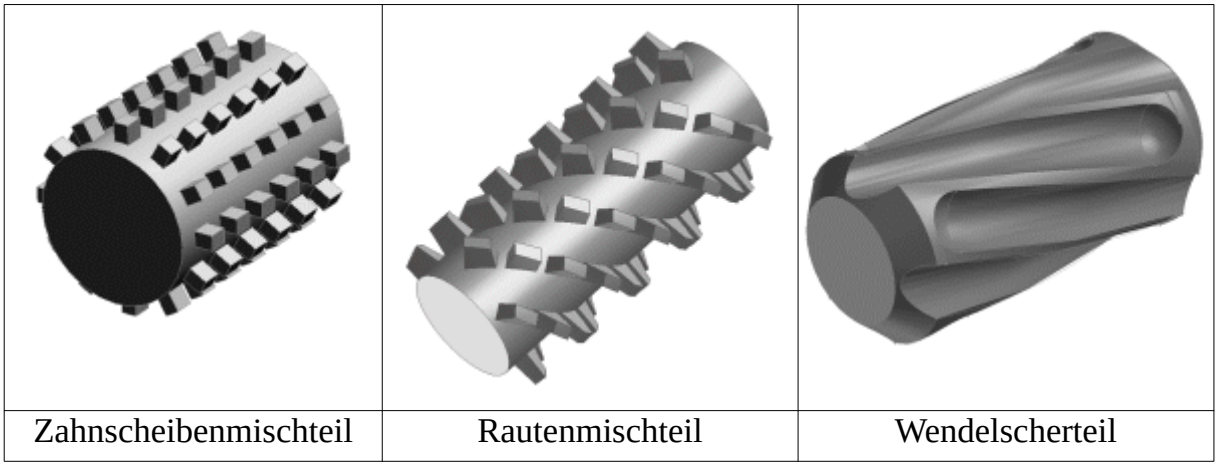
Die folgende Beschreibung der Simulationsprogramme soll einen kurzen Einblick in die jeweiligen Besonderheiten und die speziellen Einsatzgebiete geben. Dabei handelt es sich um drei Programme für Plastifiziereinheiten, und ein Programm für eine Sonderbauform eines Werkzeugs. Da auch dieses mit den gleichen Materialdaten arbeitet, wurde es in die vorliegende Arbeit mit aufgenommen.

1. Gemessen auf einem PC mit PentiumII-300 MHz

2.2.1 REX und PSI

Die *Rechnergestütze EXtruderauslegung (REX)* und das *Paderborner SImulationsprogramm für Spritzgießplastifiziereinheiten (PSI)* sind die beiden Projekte mit der längsten Entwicklungszeit, die zur Zeit am KTP betreut werden. Sie existieren seit ca. 10 Jahren und dienen der Auslegung und Optimierung von Einschneckenmaschinen. Eine Besonderheit ist die Möglichkeit, neue Schnecken auf Basis bereits erprobter Schnecken zu entwickeln. Dabei wird die Schnecke ausgehend von einer existierenden Vorlage hoch- bzw. herunterskaliert. Durch die zonenweise Analyse der Modelle ist eine komplexe und genaue Berechnung der ausgewählten Schneckenkonfiguration möglich.

Bei der Verarbeitung des Werkstoffes stehen die Produktqualität und der erzielte Durchsatz im Vordergrund. Oftmals vermindert eine Steigerung des Durchsatzes aber zugleich die Homogenität der Schmelze und somit die Qualität. In der Praxis werden daher zur Homogenisierung der Schmelze häufig Scher- und Mischteile, wie sie in Abbildung 2 dargestellt sind, eingesetzt. Gängige Elemente sind das Rautenmischteil und das Maddockscherteil



Zahnscheibenmischteil	Rautenmischteil	Wendelscherteil

Abbildung 2 Typische Elemente zur Homogenisierung der Schmelze

Die analytischen Berechnungen der Simulationssoftware basieren auf sogenannten Funktionszonenmodellen. Dabei wird die Schnecke in Zonen unterteilt, denen unterschiedliche Aufgaben während des Plastifizierprozesses zukommen. Die Entwicklung solcher Modelle ist eine der Aufgaben der wissenschaftlichen Mitarbeiter am KTP. Eine wichtige Grundlage der Modellbildung sind mathematische Methoden, wie z.B. die FEM[1]-Simulation oder die Regressionsanalyse[2].

---

1. FEM = Finite Elemente Methode. Die Finite-Elemente-Methode (FEM) ist heute wohl das wichtigste Näherungsverfahren zur numerischen Lösung elliptischer, partieller Differentialgleichungssysteme. Die Grundidee der FEM ist die Diskretisierung des zu Grunde liegenden Kontinuums mit unendlich vielen Freiheitsgraden in einfach berechenbare, finite (endliche) Elemente mit endlichen vielen Freiheitsgraden. Zunächst wurde sie in der linearen Strukturmechanik zur Berechnung von Spannungen und Verformungen in statisch und dynamisch beanspruchten Bauteilen eingesetzt. Erst danach erkannte man ihre allgemeine Anwendbarkeit auf lineare und nichtlineare Feldprobleme.

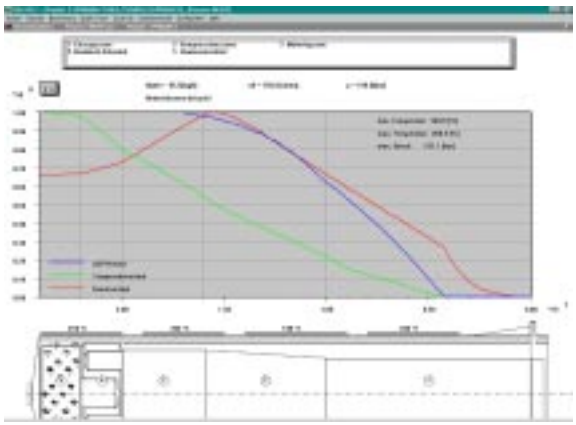
Durch eine am KTP entwickelte Kopplung der einzelnen Zonen ist auch die Berechnung komplexer Schnecken mit hoher Genauigkeit möglich. Zwei typische Mehrzonenschnecken sind in Abbildung 3 dargestellt. Bei der linken Schnecke handelt es sich um ein einfaches Modell einer Dreizonenschnecke. in der rechten Abbildung ist eine ähnliche Schnecke dargestellt, jedoch sind hier zwei weitere Zonen, ein Maddock-Scherteil und ein Rautenmischteil, zu erkennen

Dreizonenschnecke

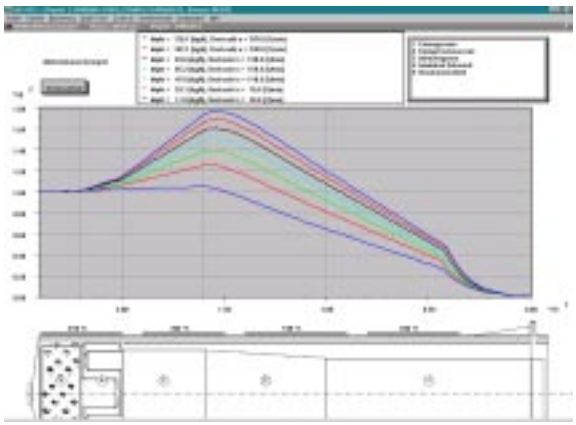
Fünfzonenschnecke mit Scher- und Mischelement

Abbildung 3 Mit REX bzw. PSI simulierte Schneckengeometrien

Mit Hilfe der Simulationsprogramme können die Verläufe mehrere Parameter, wie z.B. Druck, Temperatur oder Leistung über der Schnecke berechnet werden (Abbildung 4a). Zusätzlich ist die Variation von einem oder zwei Parametern möglich, um so verschiedene Betriebspunkte direkt miteinander vergleichen zu können (Abbildung 4b).



a) Gesamtergebnis Aufschmelzen, Druck und Temperatur



b) Auswirkung unterschiedlicher Schnekkendrehzahlen auf den Druck

Abbildung 4 Simulationsergebnisse mit REX

2. Ziel der Regressionsanalyse ist die Bestimmung des funktionalen Zusammenhanges zwischen einer abhängigen und einer oder mehreren unabhängigen Variablen. Je nach Art des Zusammenhanges und der Anzahl der unabhängigen Variablen wird zwischen linearer und nicht-linearer bzw. einfacher und multipler Regressionsanalyse unterschieden.[Kre97]

2.2.2 SIGMA

Die Simulationssoftware SIGMA (*SImulation Gleichläufiger DoppelschneckenMAschinen*) ist im Gegensatz zu der Einschneckensimulation von REX und PSI für die Auslegung von Doppelschneckenmaschinen konzipiert. Ziel der Entwicklung ist die schnelle und globale Beurteilung des Maschinenverhaltens, um damit Optimierungen der Zylinder- und Schneckenkonfiguration sowie der Verfahrensparameter zu ermöglichen. Dabei geschieht auch hier, wie schon in REX und PSI, die Umsetzung der mathematischen Modelle vorwiegend auf analytischer Basis bzw. mit geschlossenen Bewegungsgleichungen. Dadurch kann auf rechen- und somit zeitintensive numerische Verfahren, wie z.B. die Finite-Elemente-Methode, verzichtet werden.

Die Berechnungsmöglichkeiten von SIGMA sind denen von REX und PSI ähnlich. Es können der Druck-, Aufschmelz-, und Temperaturverlauf berechnet werden, sowie die Leistungsaufnahme der Schneckenmaschine. Zusätzlich kann der Füllgrad überwacht und eine Abschätzung der *Dispergiergüte* gemacht werden. Die Dispergiergüte ist ein Mittel zur Beschreibung des Zerkleinerungsgrades von mineralischen Füllstoffen in Polymermischungen. Je kleiner der Füllstoff zerrieben ist, desto höher ist der Gütegrad der Dispersion. Dieser Gütegrad ist u.a. ein wichtiges Kriterium bei der Verteilung von farbigen Zusätzen um Kunststoffe gleichmäßig einzufärben.

Die Besonderheit von SIGMA liegt in der Möglichkeit, neben „reinen“ polymeren Werkstoffen auch Aussagen über Mischungen von zwei Werkstoffen, sogenannten *Polymerblends*, und Mischungen von Polymeren und Füllstoffen (*Polymer-Füllstoff Compounds*) zu treffen. Dazu sind weitere Modelle von Schneckenelementen, sogenannte Knetblöcke, implementiert worden. Knetblöcke ermöglichen eine sehr starke Zerkleinerung und Vermengung von Materialien. Zusätzlich ist eine Modul zur Abschätzung des Druckverlustes in Extrusionswerkzeugen in SIGMA integriert.

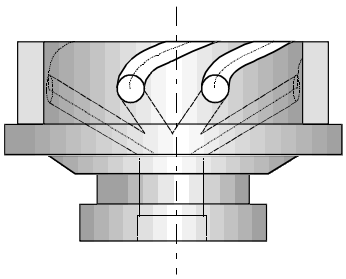
2.2.3 WENDEL

Beim Wendelverteiler handelt es sich nicht um eine Spritzgießmaschine, sondern um eine besondere und komplexe Form eines Werkzeugs. Bei der Herstellung von Rohren, Schläuchen und ähnlichen Produkten aus thermoplastischen Kunststoffen werden überwiegend Werkzeuge auf Basis des Wendelverteilerprinzips eingesetzt. Es hat sich gezeigt, daß dieses Werkzeugkonzept sehr gut an die jeweiligen Anforderungen angepaßt werden kann.

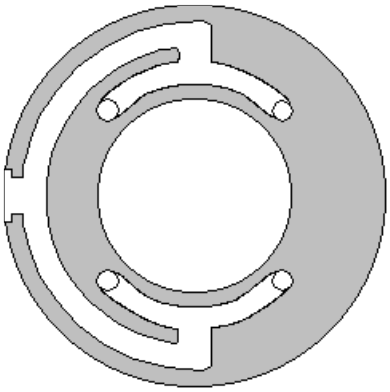
Wie in Abbildung 5 dargestellt, wird beim Wendelverteilerwerkzeug der vom Extruder angelieferte Schmelzestrom zunächst in mehrere Einzelströme aufgeteilt. Hierbei wurden über lange Zeit ausschließlich Verteilersysteme eingesetzt, die die Schmelze über sternartig angeordnete Bohrungen in die Wendeln einspeisen (Abbildung 5a). Außerdem waren Ringkanäle als Vorverteiler zu finden; vorzugsweise dann, wenn große

Durchlaßbohrungen im Dorn (Abbildung 6) notwendig waren (z.B. bei der Ummantelung von Rohren).

Neueste Entwicklungen zeigen in diesem Bereich Alternativen derart, daß die Schmelze zunächst mit Hilfe einer Kleiderbügelgeometrie (Abbildung 5b) in einen Ringspalt verteilt wird, bevor sie in die Wendelverteilerkanäle strömt. Vorteile erhofft man sich hieraus insbesondere bei Coextrusionswerkzeugen, bei denen die mögliche Anzahl einzelner Vorverzweigungsbohrungen aus konstruktiven Gründen begrenzt ist.



a) Sternverteiler



b) Ringkanalverteiler

Abbildung 5 Vorverteilsysteme für Wendelverteiler

Den sogenannten Primärverteilern (Stern- bzw. Ringkanalverteiler) schließen sich die wendelförmigen Kanäle an (Abbildung 6), die in den Dorn eingearbeitet sind und diesen in Form eines Mehrfachgewindes umlaufen. Hierbei nimmt die Kanaltiefe ständig ab, wobei sich der Spalt zwischen Dorn und äußerem Werkzeugteil im allgemeinen in Extrusionsrichtung stetig vergrößert.

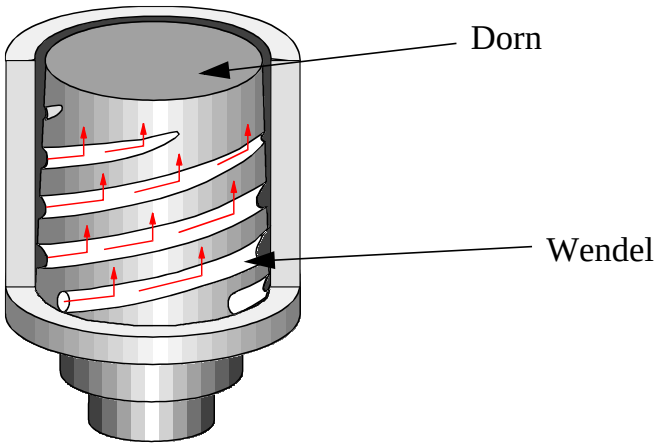


Abbildung 6 Wendelverteiler

Der Vorteil der Wendelverteilerwerkzeuge ergibt sich im wesentlichen daraus, daß durch das Aufteilen des Schmelzeflusses in mehrere Teilströme, eine sich überlagernde

Radial- und Axialströmung entsteht. Bindenähte, die bei anderen Werkzeugkonzepten strukturell immer vorhanden sind, werden somit vermieden. Hierdurch kann neben der gewünschten mechanischen auch eine hohe thermische Homogenität der Schmelze erreicht werden.

Bei der Gestaltung eines solchen Werkzeugs stehen dem Werkzeugkonstrukteur zur Beeinflussung der Schmelzeverteilung eine Reihe von Geometriegrößen zur Verfügung. Dies sind im einzelnen:

- die Baugröße
- die Kanalanzahl,
- die Kanalbreite,
- die Kanalanfangstiefe,
- der Kanaltiefenverlauf,
- die Anfangsspalthöhe,
- der Spalthöhenverlauf und
- der Steigungswinkel der Wendel

Im speziellen Fall des Wendelverteilers ist die Verteilung der Schmelze über dem Umfang das wichtigste Kriterium.

Das Programm WENDEL soll die Probleme bei der Untersuchung von Wendelverteilern lösen helfen. Allerdings handelt es sich hierbei um ein relativ junges Forschungsgebiet, so daß auch die Funktionalität des Programmes noch nicht sehr groß ist. Es können jedoch ähnlich wie bei den zuvor erwähnten Simulationsprogrammen Aussagen über die Druck- und Temperaturverteilungen getroffen werden. Zusätzlich kann das Verteilungsverhalten der Masse im Wendelverteiler simuliert werden.

3 Anforderungen an das Informationssystem EvolMat

In diesemKapitel wird untersucht, welche Anforderungen an das Informationssystem gestellt werden. Um das System im produktiven Umfeld einsetzen zu können, müssen Daten ein- und ausgegeben werden. Daher wird zuerst die Aufnahme eines Materials in das Informationssystem untersucht und anschließend wie die Daten dem Benutzer zur Verfügung gestellt werden können. Zusätzlich wird die Möglichkeit, das System an Änderung der Datenstruktur anzupassen, betrachtet.

3.1 Analyse

3.1.1 Eingabe eines Materials

Damit ein Material in die Datenbank aufgenommen werden kann, müssen zuerst diejenigen Eigenschaften, die das Material als ganzes kennzeichnen, eingegeben werden. Dazu gehören der Materialname und Materialtyp, der Bearbeiter und die Chargennummer, aus der die Materialprobe stammt, sowie das Datum der Messung. Zusätzlich kann ein freier Text für weitere Informationen eingegeben werden. Dabei läßt sich oftmals aus dem Materialnamen der zugehörige Materialtyp herleiten.

Die Datenaufnahme wird hier beispielhaft für das Material Akulon[1] durchgeführt. Dabei handelt es sich um polyamiden Kunststoff (PA). PA, oft auch als technischer Kunststoff bezeichnet, wird häufig in der Automobilindustrie verwendet, da er unter anderem die Eigenschaft besitzt, hohen mechanischen Belastungen standzuhalten.

Die Eingabe der Daten erfolgt in einem grafischen Dialogsystem (Abbildung 7). Die Gestaltung und Aufteilung der Dialogmasken orientiert sich an kunststofftechnischen Vorgaben. Die Plausibilitätsprüfung der eingegebenen Daten erfolgt programmintern, so daß fehlerhafte bzw. unsinnige Eingaben vermieden werden. Dazu werden für nahezu alle Einagben gültige Wertebereiche definiert. Die Wertebereiche begründen sich zum einen auf physkalische oder chemische Vorgaben wie z.B. Granulatgrößen oder der Schmelzpunkt eines Materials. Zum anderen werden die geometrischen Größen auf sinnvolle Eingaben hin überprüft. So kann z.B. der Schneckendurchmesser nicht größer sein, als der Zylinderdurchmesser.

---

1. Genaue Bezeichnung des Materials: Akulon 123 DSM PA 6

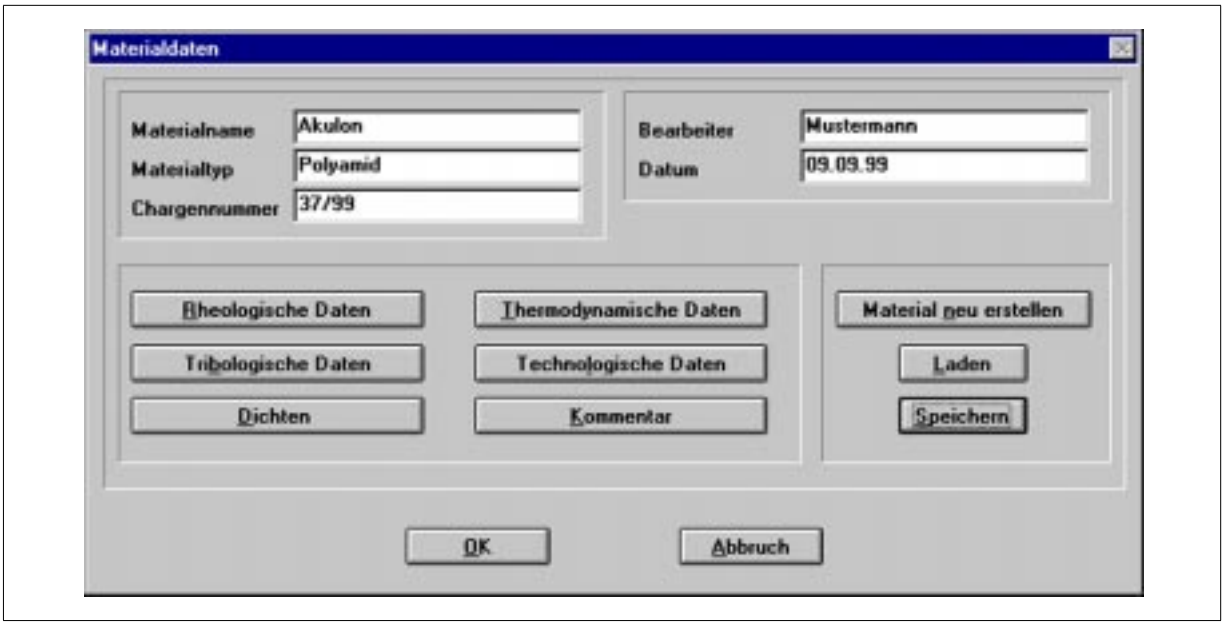


Abbildung 7 Eingabemaske Materialdaten

Im weiteren können jetzt die materialspezifischen Eigenschaften eingegeben werden. Die Eingabe ist dabei nach physikalischen Eigenschaften gegliedert, wodurch sich die weiteren Eigenschaften eindeutig einer Gruppe zuordnen lassen. Für viele Materialien sind gängige Eingabewerte in der einschlägigen Literatur nachzulesen ([Rao89]).

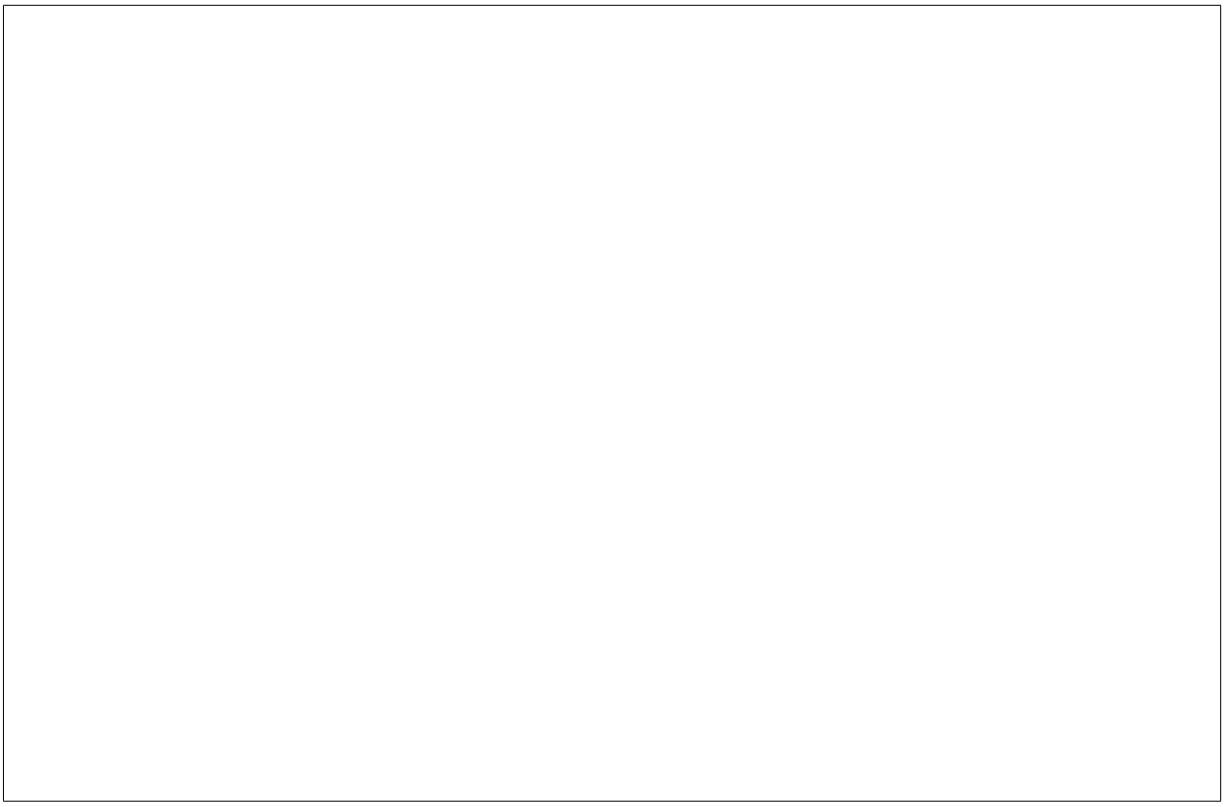


Abbildung 8 pvT-Diagramm

Oftmals ist es schwierig eine vollständige Beschreibung der Materialeigenschaften in der Literatur zu finden. Viele der Parameter lassen sich aber auch aus Funktionsgraphen

oder Diagrammen ablesen. Eine häufig verwendete Darstellung ist das *pvT-Diagramm* (Abbildung 8). Hierbei wird das spezifische Volumen des Materials (v) über der Temperatur (T) aufgetragen. Als Parameter für die Kurvenschar wählt man verschiedene Drücke (p). Mit Hilfe dieses Diagramms lassen sich die Werte v_0[1] (Volumen bei 0 ˚C) und v_m (Steigung der Volumenfunktion) ermitteln.

Neben der reinen Datenverwaltung sollen mit Hilfe des Informationssystems auch bestimmte Materialeigenschaften, wie z.B. die spezifische Wärme oder die Enthalpie, aus im Labor gemessenen Größen berechnet und visualisiert werden. Dazu können an geeigneter Stelle die gemessenen Wertelisten eingegeben bzw. aus einer Datei eingelesen werden. Aus den Wertelisten lassen sich mit Hilfe mathematischer Methoden (Regressionsanalyse) Funktionen berechen. Anhand der Funktionsdarstellung können weitere wichtige Kenngrößen der Materialeigenschaften abgelesen werden.

3.1.2 Bereitstellung der Daten für den Anwender

Das Informationssystem EvolMat soll seine Daten überwiegend den im vorherigen Kapitel vorgestellten Simulationsprogrammen zur Verfügung stellen. Diese verwenden die erhaltenen Daten zur Berechnung innerhalb der Simulation. Typischerweise wird ein Anwender des Informationssystems ein ganz konkretes Material suchen oder eine Klasse von Materialien eines bestimmten Typs. Dabei ist die Selektion nach einem bestimmten Typ gängige Praxis. Jeder Materialtyp wie z.B. Polyamide oder Polyethyläne besitzt für ihn markante Eigenschaften. Typische Eigenschaften sind hohe Bruchfestigkeit, starke Elastizität oder eine hohe Widerstandsfähigkeit gegen Wärme bzw. Kälte.

Um ein entsprechendes Material zu wählen, sind grundlegende kunststofftechnische Kenntnisse erforderlich. Das Einsatzgebiet für das Informationssystem EvolMat ist dabei zweigeteilt. Zum einen wird es im Rahmen der Forschung und Lehre am KTP in Paderborn verwendet. Hier werden Materialdaten aus neuen Untersuchungen in die Datenbank eingepflegt, um sowohl den Mitarbeitern als auch den Studenten neueste Ergebnisse und Daten bieten zu können. Das zweite Arbeitsumfeld sind die an den Forschungsprojekten beteiligten Hersteller und Verarbeiter von Kunststoffen. Diesen Firmen bietet sich die Möglichkeit, ihre Materialen und Meßergebnisse zentral zu halten und den entsprechenden Abteilungen, wie z.B. der Entwicklung, zur Verfügung zu stellen.

3.1.3 Berücksichtigung von Strukturänderungen

Neben der Verwaltung der Daten und der Möglichkeit der Auswertung, soll das Informationssystem auch Mechanismen zur Konvertierung bieten. Die Konvertierung bezieht sich auf die Möglichkeit, die Materialdaten nicht nur den verschiedenen Simu-

1. siehe Anhang B: Materialdaten

lationsprogrammen zur Verfügung zu stellen, sondern auch verschieden Dateiformate innerhalb der Programme zu unterstützen. Bedingt durch die teilweise recht lange Entwicklungszeit der Simulationsprogramme von bis zu zehn Jahren, haben sich die Anforderungen an die Materialdaten im Laufe der Zeit geändert. Dieser Wandel der Materialdaten wird sich weiterhin abzeichnen, da die Simulationsprogramme stets die neuesten Forschungsergebnisse beeinhalten sollen.

Dieser Wandel muß sich auch in der Struktur, die zur Modellierung der Materialdaten verwendet wird, wiederspiegeln. Daher muß die Struktur den neuen Anforderungen an die Materialdaten angepaßt werden. Die meisten Änderungsoperationen für das entwikkelte konzeptionelle Schema beziehen sich auf das Löschen oder Hinzufügen von einzelnen Attributen oder ganzen Klassen. Es kann jedoch auch vorkommen, daß Attribute durch Änderungen von Materialdaten zwischen Klassen verschoben werden müssen. Daher muß auch für solche Verschiebeoperationen eine entsprechende Methode zur Verfügung gestellt werden. Bei all diesen Operationen, die zur Anpassung des konzeptionellen Schemas eingesetzt werden, soll der Anwender entsprechend unterstützt werden. Diese Thematik wird in Kapitel 5 genauer betrachtet.

3.2 Anforderungen

Die im vorherigen Kapitel vorgestellten Simulationsprogramme dienen als Unterstützung zur Weiterentwicklungen von Schneckenmaschinen und Werkzeugen. So verschieden die einzelnen Einsatzgebiete auch sind, haben alle Programme eines gemeinsam: Zur Berechnung des Plastifiziervorgangs in den Schneckenmaschinen bzw. der Materialverteilung im Wendelverteiler sind umfangreiche Kenntnisse über die Eigenschaften des verarbeiteten Materials notwendig. Die Materialeigenschaften sind dabei unabhängig von den Simulationsprogrammen.

Die meisten der untersuchten Materialen lassen sich aber nicht nur in einem, sondern in mehreren oder oft sogar in allen Programmen verwenden. Bei der Entwicklung der Simulationsprogramme ist diesem gemeinsamen Aspekt nur wenig Beachtung geschenkt worden. So sind verschiedene Formate zur Speicherung der Daten durch die unterschiedlichen Simulationsprogramme entstanden, die keinen Austausch der Materialeigenschaften erlauben. Zusätzlich gibt es innerhalb der Simulationsprogramme durch die teilweise lange Entwicklungsphase von bis zu zehn Jahren Unterschiede in den Datenformaten. Dieser mittlerweile recht große Bestand an Legacy-Daten steht somit nicht allen Simulationsprogrammen zur Verfügung. Unter Legacy-Daten versteht man dabei den Bestand der bereits vorhandenen Daten, die bei einer Änderung der Datenschemata mit angepaßt werden müssen. Dies stellt oftmals ein nicht geringes Problem dar, da über die Organisation dieser Datenbestände aufgrund ihres Alters nur selten ausreichende Dokumentation vorhanden ist. Werden neue Materialen untersucht und in

eines der Simulationsprogramme aufgenommen, so können auch diese Daten wiederum nur durch manuelle Eingabe in die anderen Programme übernommen werden.

Hierfür soll das Informationssystem EvolMat, basierend auf den Anforderungen der einzelnen Simulationsprogramme, eine globale Verwaltung der Materialdaten bieten. Die Daten werden in einer Datenbank abgelegt, auf die EvolMat zugreifen kann. Das Informationssystem dient dabei als Schnittstelle zwischen der Datenbank und den Anwendungsprogrammen (Abbildung 9).

Wichtigster Aspekt des Informationssystems ist die Möglichkeit, aus den in der Datenbank abgelegten Materialeigenschaften auf Anfrage diejenigen zu extrahieren, die für die jeweiligen Simulationsprogramme benötigt werden. Dazu muß einerseits ein globales Schema entwickelt werden, das alle Eigenschaften des Materials enthält, sowie die Abhängigkeiten der Materialeigenschaften untereinander berücksichtigt. Zum anderen ist es erforderlich, genaue Kenntnisse über die jeweils benötigten Materialdaten für die Simulationsprogramme zu haben, um diese entsprechend zur Verfügung stellen zu können.

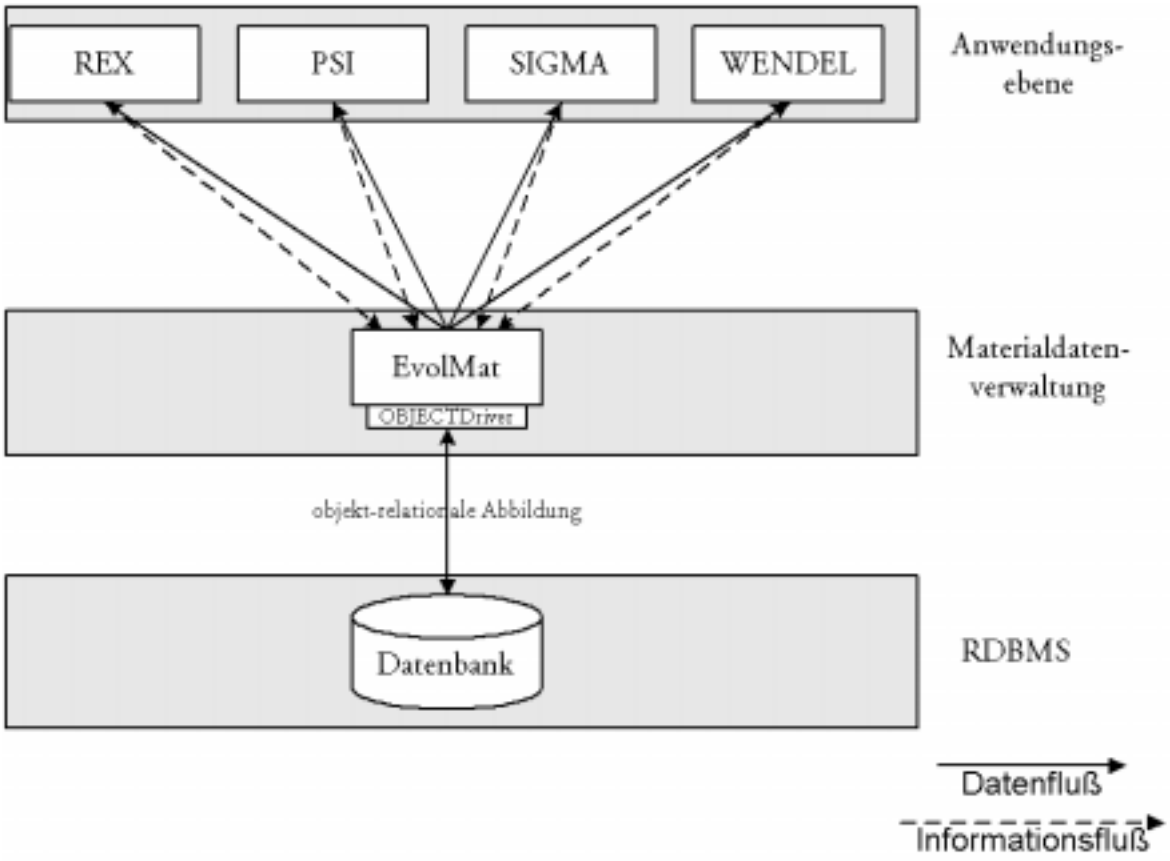


Abbildung 9 EvolMat als Schnittstelle

EvolMat wird als Schnittstelle zwischen dem RDBMS und den Anwendungsprogrammen konzipiert. Dazu werden in EvolMat Informationen über die von den Simulationsprogrammen benötigten Informationen abgelegt (gestrichelte Pfeile). Da EvolMat nach objekt-orientierten Methoden entwickelt und implementiert wurde, müs-

sen die internen Datenstrukturen auf die relationale Struktur der Datenbank abgebildet werden. Diese Aufgabe übernimmt die Middleware ObjectDRIVER. Sie kapselt alle relationalen Zugriffe und verhält ich gegenüber dem Informationssystem wie ein OO-DBMS. EvolMat seinerseits stellt die aus der Datenbank ausgelesenen Informationen den Simulationsprogrammen zur Verfügung. Dafür wird ein Filtermechanismus entwikkelt, der nur die für die Anwendungen relevanten Daten aus EvolMat exportiert.

Die gesamte Anwendung soll später in einem heterogen Arbeitsumfeld lauffähig sein. Diese Anforderungen entstammen sowohl dem universitären wie auch industriellen Umfeld. In beiden Gebieten werden unterschiedliche Betriebsysteme und Datenbanken eingesetzt. Um eine möglichst starke Plattformunabhängigkeit zu gewährleisten sind die Anwendungsmodule in C++ bzw. Java geschrieben. Bei der Wahl der Datenbank wird der weiten Verbreitung von relationalen Datenbanksystemen (RDBMS) Rechnung getragen. Um gleichzeitig die Vorteile der objektorientierten Methoden und Modellierungstechniken zu nutzen, wird eine Software integriert, die die objekt-relationale Abbildung gewährleistet.

4 Grundlagen

4.1 Verwandte Arbeiten

4.1.1 Die Komponentenbibliothek Copia

Die Arbeit von U. Nickel [Nic97] beschreibt den Aufbau einer objektorientierten Bibliothek für gemischt analog/digitale Schaltkreise. Für diese Schaltkreise wurde im C-LAB in Paderborn ein regelbasiertes Analysesystem entwickelt. Aus diesem System können Informationen über die Daten eines Schaltkreises und das damit verbundene Strukturwissen gewonnen werden.

Dazu wurden verschiedene Betrachtungsweisen eines Schaltkreises analysiert und die ergebnisse mit Hilfe der OMT[1]-Notation modelliert. Dieses Modell ergab eine baumartige Klassenstruktur. Die so gewonnenen Strukturbäume waren Grundlage für die Spezifikation einer Klasse, die einen Schaltkreis modelliert, in dem die Datenstrukturen in einen semantischen Zusammenhang gebracht werden. Diese Zusammenhänge wurden auch bei der Konzeption einer einfachen textuellen Benutzungsschnittstelle verwendet, wodurch eine Konsistentprüfung und Unterstützung des Benutzers bei der Dateneingabe möglich ist.

Zur dauerhaften Speicherung des Schaltkreisdaten wurde eine Anbindung an ein relationales Datenbank-Management-System (RDBMS) realisiert. Die Umsetzung der objektorientierten Datenstrukturen auf die relationale Ebene erfolgt mittels einer Zugriffsschicht. Die objekt-relationale Zugriffsschicht beinhaltet zusätzlich eine Transaktionsmanagement und die Gewährleistung von eindeutigen Identitäten der Objektinstanzen. Aus Gründen der Portabilität fand die SQL-Schnittstelle ODBC Verwendung.

Um den häufigen Änderungen an den Strukturbäumen Rechnung zu tragen, wird von der Anwendung der Mechanismus der Schemaevolution unterstützt. Dazu wurden das Konzept der Metaschemata verwendet. Dadurch wird die Schematransformation mittels Änderungsoperationen und die Konsistenzsicherung gewährleistet. Änderungen am Schema werden in entsprechenden Objekten gespeichert, deren Informationen zur Reorganisation der Datenbasis ausgewertet werden können. Die Änderungsoperationen selbst werden durch die Veränderung der Informationskapazität kategorisiert.

Im Vergleich zur vorliegenden Arbeit lassen sich einige gemeinsame Ideen und Ansätze wiederfinden. Das Informationssystem EvolMat soll neben der Verwaltung der Materi-

1. OMT = Object-Modeling-Technique nach James Rumbaugh

aldaten zusätzlich die Möglichkeit bieten, aus den vorhanden Daten weitere Ergebnisse abzuleiten. Dies sind z.B. Berechnungen auf den eingelesen Meßwerten. Nach Analyse der vorhanden Materialdaten werden diese mit Hilfe der UML-Notation modelliert. Da die Daten hier jedoch im Unterschied zur Komponentenbibliothek aus mehreren Quellen stammen, werden die unterschiedlichen Sichten zur weiteren Verarbeitung konsolidiert.

Die Speicherung der Daten erfolgt ebenfalls in einem RDBMS, so daß auch hier eine objektrelationale Abbildung nötig ist. Dies wird durch den Einsatz der Middleware ObjectDRIVER realisiert. Diese Zugriffsschicht bietet die gewünschten Funktionalitäten, wie die Persistenz von Klassen, die Kapselung der relationalen Daten, Transaktionsmanagement und die Verwaltung der Objektidentitäten.

Die Berücksichtigung von Änderungen an der Datenstruktur erfolgt durch die Schemaevolution. Die Änderungsoperationen für EvolMat betreffen allerdings keine grundlegenden Strukturänderungen wie dies bei Copia der Fall sein kann. Vielmehr ist die baumartige Datenstruktur für EvolMat durch die langjährige Forschung des KTP in diesem Bereich sehr ausgereift, wodurch sich nur geringfügige Änderungen auf der untersten Ebene, wie das Hinzufügen, Löschen oder Umbenennen von Attributen, ergeben.

4.1.2 Integration von JAVA-Anwendungen mit relationalen Informationssystemen

Die Arbeit von H. Schalldach [Sch98] beschäftigt sich mit der Entwicklung von OCSI[1]-Informationssystemen. Diese Informationssysteme stellen eine Verbindung der aktuellen Schlüsseltechnologien, wie Objektorientierung, Client/Server-Architekturen und den Einsatz des Internets, dar.

Bei der Entwicklung und Nutzung solcher Systeme greifen viele Unternehmen auf ihre vorhanden Informationsbestände zurück. Diese Legacy-Daten sind historisch bedingt häufig in relationalen Datenbank-Management-System (RDBMS) abgelegt. Ein sehr wichtiger Punkt bei der Modernisierung dieser Systeme ist daher die Aufarbeitung der vorhandenen Datenbestände, das sogenannte Reengineering. Erst wenn dieser Prozeß abgeschlossen ist, können die Legacy-Informationssysteme modernisiert werden.

Die Anforderungen an aktuelle Systeme, wie z.B. die Verwendung von objektorientierten Methoden und Technologien, lassen sich aber nur schwer mit den relationalen Techniken vereinbaren. Daher muß ein Mechanismus zur Konvertierung bereitgestellt werden, der es ermöglicht, die komplexen Objekte auf ein Relationen-Schema abzubilden.

H. Schalldach hat hierfür einen Mechanismus entwickelt, der die automatische Generierung der Mittelschicht nach dem ODMG[2]-Standard erlaubt. Die benötigten

1. OCSI = Objektorientiert, Client/Server, Internetbasiert

Informationen werden mit Hilfe des Reengineering-Werkzeugs VARLET erzeugt. VARLET wird dabei zur Unterstützung des kompletten Reengineering-Prozesses eingesetzt. Dieser Prozeß erstreckt sich von der Analyse des relationalen Schemas bis zur Migration in ein objektorientiertes Schema.

Reengineering und Datenmigration mit Hilfe von VARLET[1] sind Thema zahlreicher Arbeiten und Veröffentlichungen am Lehrstuhl Softwaretechnik der Universität Paderborn. Ein kurzer Überblick über die Ideen und Techniken wird in Abschnitt 4.2.5 gegeben.

Auch in der Arbeit von H. Schalldach finden sich einige Gemeinsamkeiten zur vorliegenden Arbeit. Zuerst müssen vorhandene Daten analysiert und strukturiert werden, um auf sie in neu entwickelten System zugreifen zu können. Die Verbindung zwischen einem vorhanden RDBMS und der neuen Anwendung erfolgt mittels einer Zugriffsschicht. Viele der notwendigen Schritte für diesen Migrationsprozeß können mit VARLET durchgeführt werden. Der für diese Arbeit interessanteste Aspekt ist aber die automatische Generierung der Mittelschicht basierend auf Informationen, die aus VARLET gewonnen werden können. Diese Konzept entspricht im wesentlichen den Anforderungen an die automatische Generierung der Export-Filter für EvolMat.

4.2 Verwendete Konzepte

4.2.1 XML

4.2.1.1 Die Entwicklung von XML

Das World Wide Web (WWW) ist der wohl am meisten genutze Dienst im Internet. Die heutzutage am weitesten verbreitete Beschreibungssprache für Webseiten ist HTML (Hyper Text Markup Language). HTML ist eine stark eingeschränkte Teilmenge der SGML (Standard Generalized Markup Language), einer auf sogenannten Tags[2] basierenden Sprache, die zwischen Inhalt und Darstellung, sowie Daten und Metadaten unterscheidet. Diese Komplexität und die aufwendigen Tools haben die weite Verbreitung von SGML verhindert.

Ein HTML Dokument besteht im wesentlichen aus dem textuellen Inhalt, gegliedert in Blöcke, die von Tags eingeschlossen werden. Mit Hilfe der Tags wird die Darstellung am Bildschirm festgelegt. Eine gut gestaltete Webseite mag dabei schön aussehen, wenn

2. Object Data Management Group

1. Verified Analysis and Reengineering of Legacy Database Systems Using Equivalence Transformations

2. Tags (engl.) sind Textmarken, die Formatierungsanweisungen enthalten, die für die Darstellung am Bildschirm von einem Browser ausgewertet werden.

sie in einem entsprechenden Browser dargestellt wird. Die HTML-Tags enthalten dabei im Gegensatz zu SGML nur Informationen darüber, wie der von ihnen umgebene Textblock darzustellen ist. Die Möglichkeit, Daten auszutauschen oder über Metadaten eine Semantik festzulegen, ist für die HTML-Tags nicht vorgesehen. Daher ist HTML als generelles elektronisches Austauschformat nur sehr bedingt geeignet.

XML, die Extensible Markup Language, ist ein neues Format um strukturierte Informationen im Web zu präsentieren. Es ist eine web-basierte Sprache, um Daten elektronisch auszutauschen. XML ist offener technischer Standard des WWW Konsortium (W3C[1]), einer Gruppe, die für die Weiterentwicklung von HTML, XML und anderen web-bezogenen Standards verantwortlich zeichnet.

Auch XML ist nur eine Teilmenge von SGML, jedoch bewahrt XML die wichtigen Grundsätze der Trennung zwischen Inhalt und Daten und verzichtet lediglich auf nicht benötigte Erweiterungen. In einem XML Dokument wird der Inhalt in Tags eingebettet, die im Gegensatz zu HTML nicht die Darstellung sondern die Struktur beschreiben. Zusätzlich bietet XML die Möglichkeit, ein Regelwerk für die Struktur eines Dokumentes anzugeben. Diese beiden Eigenschaften erlauben die bei der Sprache HTML fehlende Trennung zwischen Daten und Metadaten und ermöglichen so eine Validierung der XML Dokumente gegenüber ihrer Grammatik.

Im Gegensatz zu HTML fehlen bei XML die Informationen zur Darstellung der Inhalte im Dokument. Stattdessen kann die visuelle Präsentation mit Hilfe von *layout styles* erfolgen, wie sie z.B. die XSL Technologie (Extensible Style Language) bietet. Sowohl Web-Browser wie auch Webseiten bieten zunehmend diese Funktionalität.

4.2.1.2 Vorteile durch den Einsatz von XML

Es gibt viele Gründe, ein OMG (Object Management Group) konformes Metadaten-Austauschformat auf Basis von XML zu entwickeln. Einige der allgemeinen Gründe sind:

- XML ist ein offener, plattform- und herstellerunabhängiger Standard
- XML ist Meta-Modell-neutral, d.h. es können alle Meta-Modelle, die konform zum OMG Meta-Meta-Modell (MOF - Meta Object Facility) sind, repräsentiert werden.
- Der XML Standard ist unabhängig bzgl. Programmiersprachen und APIs[2]. Gebräuchliche APIs sind DOM, SAX und Web-DAV[3]

---

1. Internetadresse: http://www.w3c.org
2. Application Programming Interface - definierte Schnittstelle bei der Programmierung
3. DOM, SAX und Web-DAV sind verschieden Domänen-Modelle zur Bearbeitung von XML, Bedeutung siehe Glossar

- Geringe Einstiegskosten. XML Dokumente können mit jedem Text-Editor erstellt werden, so daß keinen teuren Werkzeuge gekauft werden müssen. Ebenso sind eine Vielzahl von hilfreichen Programmen zur automatischen Generierung von XML Dokumenten frei im Netz erhältlich (z.B. auf den Web-Seiten von IBM: www.alphaworks.ibm.com).

Durch die Integration einer XML-Schnittstelle wird das Reengineering-Werkzeug VARLET um die Möglichkeit des standardisierten Datenaustausches erweitert. Die Informationen des objekt-orientierten Schemas können so jeder XML-fähigen Anwendung zur Verfügung gestellt werden.

4.2.1.3 Funktionsweise von XML

Die grundlegende Struktur eines XML Dokumentes ist baumartig aufgebaut. Die einzelnen Äste bestehen dabei aus den im Dokument enthaltenen Daten und den Tags, die die Daten kapseln. Die Regeln, wie die Tags strukturiert sind, wird in der der *Document Type Definition (DTD)* festgelegt.

Strukturelemente

Im einfachsten Fall besteht ein XML *Tag* aus dem Namen, eingeschlossen in spitze Klammern („<“ und „>“). Die Tags treten immer paarweise auf, d.h. es gibt ein öffnendes und ein schließendes Tag, ähnlich der Klammersetzung. Dabei besteht das schließende Tag aus dem gleichen Namen wie das öffnende Tag, zusätzlich wird jedoch ein Schrägstrich („/“) vorangestellt. Formal wird ein solches Konstrukt aus öffnendem und schließendem Tag als *Element* bezeichnet, und der Text zwischen den Tags als *Inhalt* des Elements. Ein einfaches Beispiel für das Element Klasse:

<Klasse> Name der Klasse </Klasse>

Der Inhalt eines XML Elements kann dabei wiederum Elemente enthalten, so daß sich eine beliebig tiefe Schachtelung ergibt. Jedoch gilt für XML Strukturen im Gegensatz zu HTML die strenge Regel, daß erst die schließenden Tags für die inneren Elemente gesetzt sein müssen, bevor die sie umgebenden Tags geschlossen werden.

Beispiel

Ein einfaches Beispiel um die Struktur eines XML-Dokumentes zu verdeutlichen (Einrückungen und Zeilenumbrüche dienen nur der Lesbarkeit und haben keine semantische Bedeutung).

```
<Material>
    <Name> Akulon 123 DSM PA 6 </Name>
    <Typ> PA </Typ>
```

```
    <Bearbeiter> Kretschmer </Bearbeiter>
    <Erstellungsdatum> 1.9.1999 <Erstellungsdatum>
    <Kommentar> Teil einer Testreihe </Kommentar>
</Material>
```

Das Element Material enthält fünf geschachtelte Elemente, die es detaillierter beschreiben: Name, Typ, Bearbeiter, Erstellungsdatum und Kommentar.

Attribute

Zusätzlich zu den Inhalten kann ein XML-Element auch *Attribute*[1] beinhalten. Attribute werden im öffnenden Tag des Elements als Wertepaar nach dem Namen aufgeführt. Beispiel:

```
<Material xmi.label="mat1"> Akulon 123 DSM PA 6 </Material>
```

Ein spezielles Attribut, das XML verwendet, ist das ID-Attribut. Dieses ermöglicht die eindeutige Identifizierung eines Elementes innerhalb des Dokumentes. Die ID ermöglicht Querverweise zwischen den Elementen, die man nur mit der XML-Syntax allein nicht ausdrücken kann.

4.2.1.4 Document Type Definitions

Mit Hilfe der Document Type Definition (DTD) legt XML die Struktur seiner Dokumente fest. Eine XML DTD definiert sowohl die unterschiedlichen Typen von Elementen, die in einem gültigen Dokument vorkommen können, so wie auch deren Kardinalität. Die Syntax der Elemente ist einfach aufgebaut. Sie besteht aus einem Paar von spitzen Klammern, die ein XML-Schlüsselwort (gekennzeichnet durch das vorangestellte Ausrufezeichen) sowie eine Liste von Parametern enthalten:

```
<!Schlüsselwort Parameterliste>
```

Eine DTD für das obige Beispiel eines Materials würde u.a. folgende Definition enthalten:

```
<!Element Material (Name, Typ, Bearbeiter, Erstellungsdatum, Kommentar?)>
```

Dies bedeutet, daß ein Element vom Typ Material die ersten vier der fünf aufgezählten Unter-Elemente genau einmal enthalten muß. Der Kommentar ist optional, muß also nicht mit eingegeben werden.

---

1. Eine Liste der möglichen Attribute ist enthalten in: [BLN86]

Die Kardinalität von Elementen wird mit der bekannten Signatur vorgenommen:

null oder eins	Fragezeichen
genau eins	ohne Zusatz
beliebig viele	Sternchen
mindestens eins	Pluszeichen
logisches oder	senkrechter Strich

Tabelle 1 Kardinalitäten von XML-Elementen

Innerhalb einer XML DTD werden auch die Attribute definiert, die ein Element enthalten kann. Dies geschieht durch Angabe des Schlüsselwortes !ATTLIST. Die folgende Definition spezifiziert, daß jedes Klassenelement vom Typ Material ein optionales Attribut *xmi.label* hat, das aus alphanumerischen Zeichen besteht. Der Parameter CDATA legt dabei den Bereich der zu verwendenden Zeichen fest und das Schlüsselwort #IMPLIED gibt an, daß das Attribut optional ist, also nicht mit angegeben werden muß.

<!ATTLIST Material xmi.label CDATA #IMPLIED>

Eine DTD kann in das Dokument, dessen Syntax sie definiert, eingebunden werden. Typischerweise wird sie jedoch als externe Datei angelegt, und innerhalb des Dokumentes als *Universal Ressource Identifier (URI)* referenziert:

„http://www.foo.org/Material.dtd“ oder „file:Material.dtd“

4.2.1.5 Korrektheit von XML-Dokumenten

Bei der Korrektheit von XML-Dokumenten wird zwischen drei Stufen unterschieden: *well-formedness*, *validity* und *semantic correctness*.

- Ein XML-Dokument heißt *well-formed*, falls alle Elemente korrekt als Baumstruktur aufgeführt sind, also die Start- und Ende-Tags richtig geschachtelt sind. Diese Stufe der Korrektheit ist essentiell für den Austausch von Informationen.
- *Valid* darf sich ein Dokument nennen, das sowohl well-formed ist als auch strukturkonform zu den Definitionen in der zugehörigen DTD. Ein gültiges XML-Dokument enthält daher nur die Elemente und Attribute, die in der DTD definiert sind. Gleiches gilt für den Inhalt der Elemente und die Werte der Attribute. Da die DTD aber weder im Dokument spezifiziert werden, noch zur Erstellung eines XML-Dokumentes herangezogen werden muß, ist sie zur Prüfung der „validity“ unbedingt notwendig.
- Der höchste Level der Korrektheit, die *semantic correctness* ist mit der derzeitigen Definition von XML und den DTDs leider noch nicht möglich. Nur derjenige, der mit den Dokumenten arbeitet, und das nötige Hintergrundwissen über die beschrie-

bene Domäne hat, kann den „Sinn“ der Informationen prüfen. Für das Material-Beispiel ist es z.B. nötig zu wissen, ob ein bestimmte Material auch wirklich dem angegebenen Typ entspricht (ist Durethan ein PA?).

4.2.2 Middleware

4.2.2.1 Vorteile durch den Einsatz einer Middleware

Aus der Anforderungsanalyse (siehe Kapitel 3.1) ergibt sich, daß das für den späteren Einsatz der Beispielanwendung zu verwendende Datenbanksystems eine breite Verfügbarkeit aufweisen muß. Daher ist der Einsatz eines relationalen Datenbank-Management-Systems naheliegend, da der Einsatz solcher Systeme weit verbreitet ist. Hinzu kommt, daß RDBMS für viele Betriebssysteme und Plattformen vorhanden und meistens auch preiswerter sind. Da relationale Techniken weit verbreitet und bereits seit längerer Zeit in vielen Anwendungsbereichen im Einsatz sind, existieren mittlerweile große Bestände an Daten, die in relationalen Datenbanken gespeichert sind und auf die von neu (objekt-orientiert) entwikkelten Anwendungen zugegriffen wird. Daher ist ein Brückenschlag zwischen der relationalen Speicherung der Daten und dem Zugriff von objektorientierten Anwendung nötig.

Komplexe Datenbankanwendungen, wie moderne CAD-Systeme, Knowledge-Bases oder große Geschäftsapplikationen können in einem enormen Maße davon profitieren, wenn die Datenbanksprache objektorientierte Programmierung unterstützt. So könnten komplexe Datenstrukturen, wie z.B. Zeichnungen von Bauteilgruppen mit ihren Stücklisten, in bestehende Datenbanksysteme integriert werden. Häufig müssen die Daten, die von diesen Spezial-Anwendungen genutzt werden, auch mit gewöhnlicheren Programmen, wie z. B. einer Auftragsverwaltung oder einem Projektmanagement augetauscht werden. Gerade dieser gemeinsame Zugriff auf die Daten ist aber häufig das Problem in solchen Szenarien.

Da die Analyse und anschließende Modellierung der Materialdaten mit objektorientierten Methoden vorgenommen wurde, muß nun eine Verbindung zwischen diesen beiden Welten geschaffen werden, die es ermöglicht, Datenobjekte der Anwendung im RDBMS abzulegen. Diese Aufgabe übernimmt eine Middleware, die den relationalen Teil derart kapselt, das der Anwendung ein objektorientierter Zugriff ermöglicht wird. Ein Beispiel einer solchen Middleware-Lösung ist die im Rahmen dieser Arbeit eingesetzte und im folgenden Kapitel vorgestellte Software ObjectDRIVER.

4.2.2.2 ObjectDRIVER

ObjectDRIVER ist ein Middleware-Werkzeug, mit dessen Hilfe man ein ODMG-konformes objektorientiertes DBMS (OODBMS) auf relationale DBMS (RDBMS) aufsetzen kann [LDAJ99]. Es bietet eine Abindung an die Programmiersprachen Java und C++ und verwendet als Anfragesprache die Objectoriented Query Language (OQL).

Dadurch eignet sich ObjectDRIVER im wesentlichen für zwei Arten von Anwendungen:

1. Weiterverwendung von bestehenden Datenbanken. Viele Informationssysteme verwenden Schemata, die nicht verändert werden können. Daher ist es sehr wichtig, neue, objektorientierte Datenbankanwendungen entwickeln zu können, ohne die zugrunde liegende Datenbank verändern zu müssen.

2. Erweiterung einer objektorientierten Anwendung um Persistenz. Falls die Klassenhierarchie einer Anwendung bekannt ist, kann ObjectDRIVER diese auf ein entsprechendes relationales Schema abbilden. Dadurch wird eine Transparenz bezüglich der Datenspeicherung und Rückgewinnung gegenüber der Anwendung und dem Benutzer erreicht.

Dadurch bietet dieser Ansatz mit einer Middleware im Vergleich zu einem OODBMS einige Vorteile:

1. Der Anwender kann das zu verwendende Objektmodell selbst entwerfen. Somit erreicht er eine direkte Sicht auf die Datenbank mit seinem eigenen Modell. Dies bedeutet, daß aus Modellierersicht kein „Bruch“ zwischen der relationalen und der objekt-orientierten Sichtweise entsteht.

2. Der objekt-relationale Zugriff bietet Zugang zu einer großen Datenmenge, da das relationale Modell ein weit verbreiteter Standard ist. In der Kunststofftechnik existiert eine seit mehreren Jahren gepflegte relationale Datenbank („Campus“) mit Informationen und Meßgrößen, die von den Materialherstellern und -verarbeitern ebenso genutzt wird, wie von verschiedenen Forschungseinrichtungen.

Viele Branchen setzen auf relationale Datenbanken, wohingegen in der Anwendungsentwicklung die objektorientierten Techniken längst zum Standard geworden sind. Dies führt zu dem Problem, daß sich die Metamodelle für das DBMS und die Anwendungsentwicklung unterscheiden. Genau an dieser Stelle greift ObjectDRIVER ein, und bietet durch die Möglichkeit des objektorientierten Zugriffs auf RDBMS eine adäquate Lösung des Problems, indem neue Applikationen für die Datenbank entwickelt werden können, ohne daß diese verändert werden muß.

ObjectDRIVER bietet zudem eine klare Trennung zwischen der Objekt-Ebene und der physikalischen Ebene. Auf Objektbasis wird der logische Aufbau einer Anwendung, also das (Hintergrund-) Wissen modelliert. Dieses wird durch die Kapselung des relationalen Zugriffs von der physischen Seite, also der permanenten Speicherung und Verwaltung der Daten getrennt.

4.2.2.3 Die Funktionsweise des ObjectDRIVER

Die Grundidee von ObjectDRIVER basiert auf der Idee des *Mapping*. Diese Technik entspricht im wesentlichen den Abbildungsverfahren, die auch in VARLET zum Einsatz kommen (siehe Kapitel 4.2.5.1 Das Mapping in VARLET). Beim Mapping wird eine Abbildungsstruktur definiert, die die einzelnen Klassen und Attribute den jeweiligen Relationen und ihren Attributen zuordnet. Dabei können die Attribute einer Klasse durchaus aus mehreren Tabellen stammen. In diesem Fall muß in der Klassendefinition mit angegeben werden, wie die zugehörigen Tabellen miteinander verknüpft sind (join-Operation).

Eine grobe Übersicht über den strukturellen Aufbau von ObjectDRIVER gibt Abbildung 11. Auf der linken Seite ist schematisch der Zugriff auf eine relationale Datenbank aus einer bestehenden (Legacy-) Anwendung heraus dargestellt. Auf der rechten Seite wird die Anbindung mittels ObjectDRIVER von einer objektorientierten Applikation auf dieselbe Datenbank realisiert. Dabei ist für den Anwender der relationale Zugriff völlig transparent, da er im ObjectDRIVER gekapselt ist.

Um die Eindeutigkeit der Objekte zu sichern, verwaltet OBJECTDriver eine eigene Liste mit Identitäten (ObjectIDs), um die sich der Anwender nicht zu kümmern braucht. Leider besitzt die Middleware keine Heuristiken, um eine Vererbungsstruktur aus dem OO-Schema aufzulösen. Die Auflösung muß der Anwender in der Definition der Klassenhierarchie vorgeben.

Durch die Verwendung einer generischen Sprache ist es recht einfach möglich, eine Klassenhierarchie basierend auf einer Tabellenrelation aufzubauen. Das folgende Beispiel gibt die Definition eines einfachen relationalen Schemas und des korrespondierenden objektorientierten Schemas wieder. Modelliert wird dabei das bereits in den vorherigen Abschnitten verwendete Material.

```
define schema Materialdaten {
  relationalDbms MsAccess;

  define table MATERIAL {
    NAME                String(30),
    TYP                 String(10),
    BEARBEITER          String(30),
    ERSTELLUNGSDATUM    String(10),
    KOMMENTAR           String(100)
  };
};

class Materialdaten on MATERIAL
```

Abbildung 10 Schemadefinition in OBJECTDriver (relational und objekt-orientiert)

```
  type Tuple(
    name                String on MATERIAL.NAME,
    typ                 String on MATERIAL.TYP,
    bearbeiter          String on MATERIAL.BEARBEITER,
    erstellungsdatum    String on MATERIAL.ERSTELLUNGSDATUM,
    kommentar           String on MATERIAL.KOMMENTAR
  )
end;
```

Abbildung 10 Schemadefinition in OBJECTDriver (relational und objekt-orientiert)

Die Notierung der Sprache ist in BNF[1] und enthält sowohl den relationalen Teil der Grammatik, als auch den für die korrespondierende objektrelationale Abbildung. Die komplette Definition ist in [LDAJ99], Kapitel „Databases definition“, enthalten. Eine Klasse wird dabei durch eine Menge von Zuordnungen über einer relationalen Tabelle definiert. Dazu zählen die Tabelle, auf der die Klasse basiert, die Zuordnung der Klassenattribute zu denen der Relation, eventuell nötige Verknüpfungen von Tabellen mittels einer join-operation , die Definition der Assoziationen oder die Angabe von Wertebereichen für einzelne Attribute (durch *constrained by* Bedingung).

1. BNF = Backus-Naur-Form, Beschreibungsform kontextfreier Grammatiken

Sind diese Definitionen erfolgt, wird der „relationale Teil“ des Systems gegenüber dem Anwender transparent. Die Repräsentation der Daten erfolgt wie bei der Ablage in einem ODBMS.

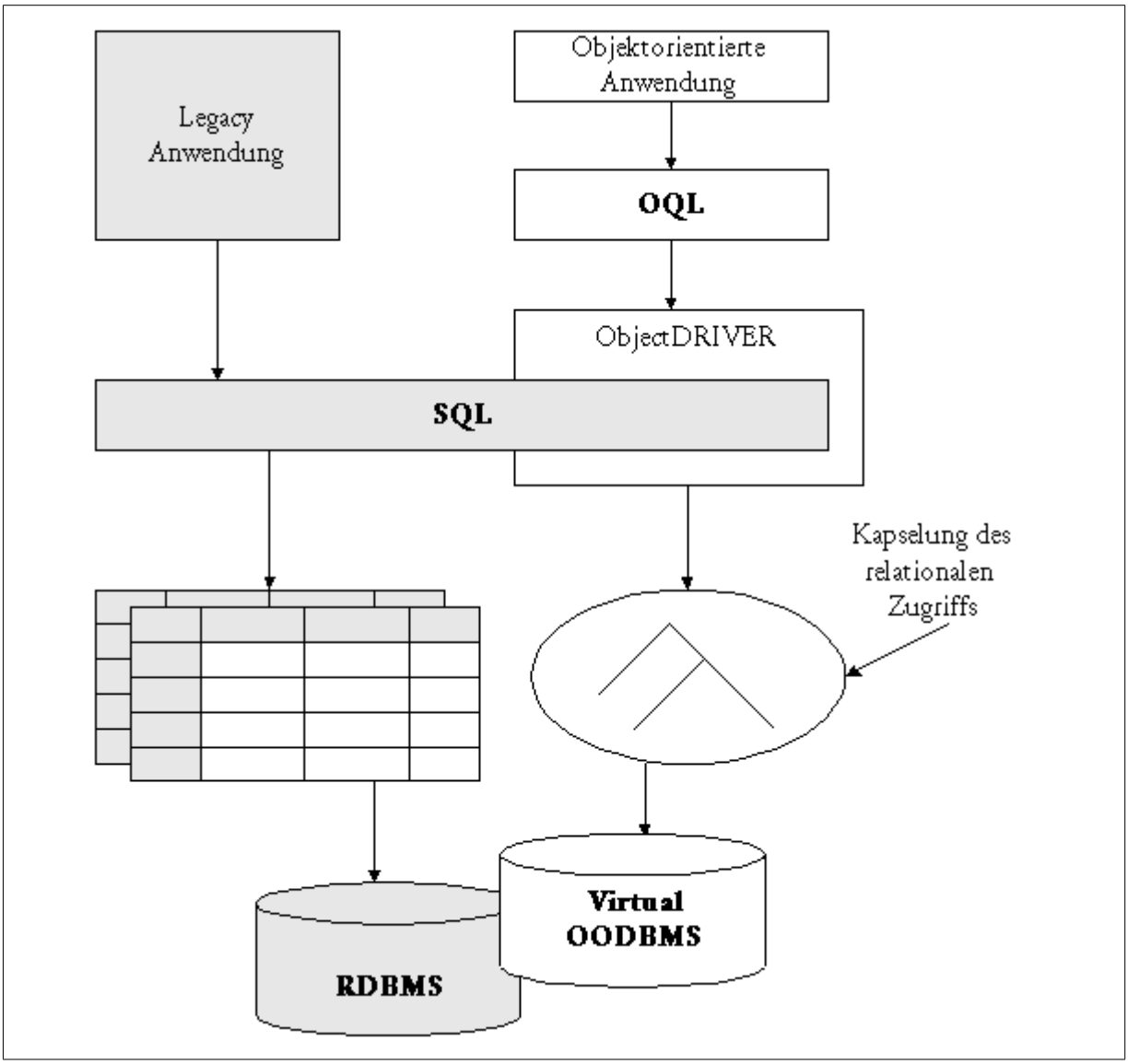


Abbildung 11 Übersicht über den Aufbau von ObjectDRIVER [LDAJ99]

Darüberhinaus bietet die Anbindung an C++ und Java die Möglichkeit, ODMG-konforme Anwendungen zu entwickeln, ohne das darunterliegende DBMS zu ändern. Die verwendete Anfragesprache ist eine Teilmenge von OQL, d.h. sie entspricht bis auf wenige Ausnahmen, deren Implementierung noch nicht oder nur unvollständig vorhanden ist, dem von der ODMG festgelegten Standard. Die Zugriffe auf die Datenbank erfolgen zur Zeit unter Windows über eine ODBC-Schnittstelle, eine Portierung der Software auf Linux ist in Arbeit. Dadurch wird der Zugriff auf eine Vielzahl von kommerziellen RDBMS gewährleistet.

4.2.3 Sichtenintegration

Zielsetzung der Sichtenintegration ist die Zusammenfassung mehrerer, unabhängig voneinander erstellter Sichten in einer Gesamtspezifikation. Im allgemeinen kann man davon ausgehen, daß sich die Sichten zum Teil überlappen, sich also an bestimmten Stellen auf die gleichen Aspekte des modellierten System beziehen. Sie sind teilweise redundant oder stehen in Beziehung zueinander. Aufgabe der Integration ist es, diese Überlappungen zu erkennen, um sie entsprechend aufarbeiten zu können. Dazu gehören die Beseitigung der Redundanzen, Vermeidung von Inkonsistenzen und Mehrdeutigkeiten und die Beibehaltung der richtigen Beziehungen.

Die Konzepte der Sichtenintegration sind denen der Schemaintegration im Bereich des Datenbankentwurfs sehr ähnlich und werden in diesem Zusammenhang erläutert [BLN86]. Bei der Schemaintegration werden die zugrundeliegenden Teilschemata in ein globales Schema integriert. Die Teilschemata können dabei verschieden Nutzersichten auf eine neu zu entwickelnde Datenbank wiederspiegeln (*view integration*), oder sie entsprechen bereits existierenden Datenbanken, die zusammengeführt werden sollen (*database integration*)[BLN86]. Im Gegensatz zur Schemaintegration, wo meist nur statische Schemata betrachtet werden, sind die folgenden Aussagen zur Sichtenintegration auch auf andere Modelle, wie z.B. Klassen- oder Interaktionsmodelle übertragbar.

Bei der objektorientierten Analyse ist ein zentraler Punkt der Sichtenintegration die Integration der Klassendiagramme. Bei der Verschmelzung zweier Klassen aus unterschiedlichen Sichten müssen sowohl die jeweiligen Zustandsdiagramme als auch die korrespondierenden Interaktionsdiagramme überprüft werden.

4.2.3.1 Bewertungskriterien für das konsolidierte Schema

Um am Ende der Integrationsschritte das daraus entstandene Gesamtschema bewerten zu können, werden in [BLN86] folgende qualitative Kriterien aufgeführt:

- *Verständlichkeit*
  Das Gesamtschema sollte für alle am Entwicklungsprozeß Beteiligten leicht verständlich und nachvollziehbar sein. Dies impliziert, daß aus den verschiedenen Möglichkeiten des Entwurfs der (qualitativ) beste ausgewählt wird.
- *Minimalität*
  Konzepte, die in mehreren Komponentenschemata auftauchen, dürfen im Gesamtschema nur noch einmal repräsentiert sein.
- *Korrektheit und Vollständigkeit*
  Das globale Schema soll alle an der Integration beteiligten Anwendungsdomänen repräsentieren. Es muß alle Konzepte, die in den Teilschemata enthalten sind, korrekt wiedergeben.

4.2.3.2 Der Integrationsprozeß

Die Sichtenintegration läßt sich nach [BLN86] in zwei Phasen unterteilen; die *Vorintegration* und eine Folge von *Integrationsschritten*.

1. In der *Vorintegrationsphase* wird festgelegt, nach welcher Strategie die Teilschemata kombiniert werden. Für die später folgende Konfliktbehandlung kann es sehr nützlich sein, jetzt schon eine Priorisierung festzulegen. Die unterschiedlichen Strategien werden in Abschnitt 4.2.3.3 erläutert.

2. In jedem *Integrationsschritt* werden dann zwei oder mehrere Sichten zu einer neuen, intermediären Sicht zusammengefaßt. Jeder Schritt läßt sich noch einmal in drei Teilaktivitäten untergliedern: *Vergleich*, *Konfliktbehandlung* und *Zusammenführung*.

2a) Während der *Vergleichsphase* sind die Korrespondenzen zwischen den zu integrierenden Sichten zu identifizieren und zu dokumentieren.

2b) Bei der *Konfliktbehandlung* wird versucht, die Schemata in Einklang zu bringen, d.h. Inkonsistenzen aufzudecken und entsprechende Konflikte zu lösen.

2c) Im letzten Schritt, der *Zusammenführung*, werden die Schemata durch Superimposition vereint. Dabei werden korrespondierende Elemente aus unterschiedlichen Sichten auf ein Element abgebildet. Korrespondenzfreie Elemente werden unverändert in die neue Sicht integriert.

4.2.3.3 Die Integrationsstrategien

Mit der Strategie wird festgelegt, wieviele Schritte benutzt werden, und in welcher Reihenfolge, um die Sichten zu integrieren. Die Integrationsstrategie läßt sich schematisch als Baum darstellen (Abbildung 12), wobei die Blätter die einzelnen Teilschemata repräsentieren, die Knoten stellen Zwischenergebnisse, sogenannte *intermediäre Sichten*, dar und die Wurzel ist die gesuchte Gesamtsicht.

Allgemein gibt es keine Festlegung für die Anzahl der in einem Schritt zu integrierenden Sichten. Man spricht dann von einer *n-ären Strategie* (Abbildung 13a). Ein Spezialfall davon ist die *One-Shot-Strategie* (Abbildung 13b), wobei alle Sichten in einem einzigen Schritt zusammengeführt werden.

Bei den *binären Integrationsstrategien* werden in jedem Schritt genau zwei Sichten vereinigt, die im nächsten Schritt mit einer Einzel- oder bereits erstellten intermediären Sicht zusammengefaßt werden. Die *Leiterstrategie* (Abbildung 13c) bzw. *balancierte Strategie* (Abbildung 13d) sind zwei Spezialfälle der binären Strategie. Im ersten Fall werden die einzelnen Sichten sequentiell (nach einer in der Vorintegrationsphase festgelegten Reihenfolge) zusammengeführt. Die balancierte Strategie hingegen fast

zunächst jeweils paarweise die einzelnen Sichten zusammen, und vereinigt dann wieder paarweise die intermediären Sichten.

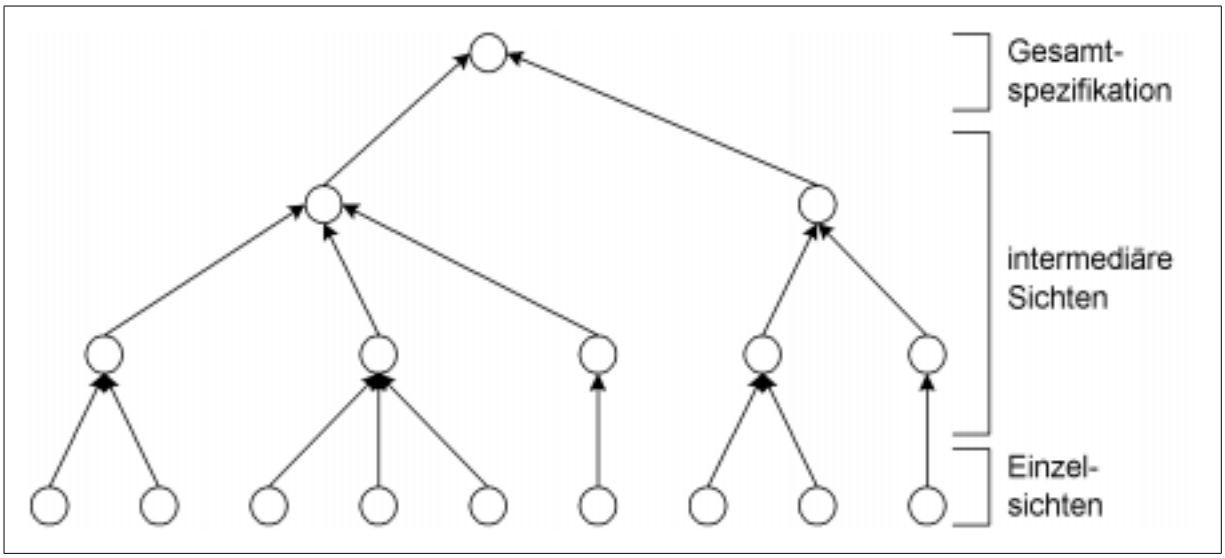
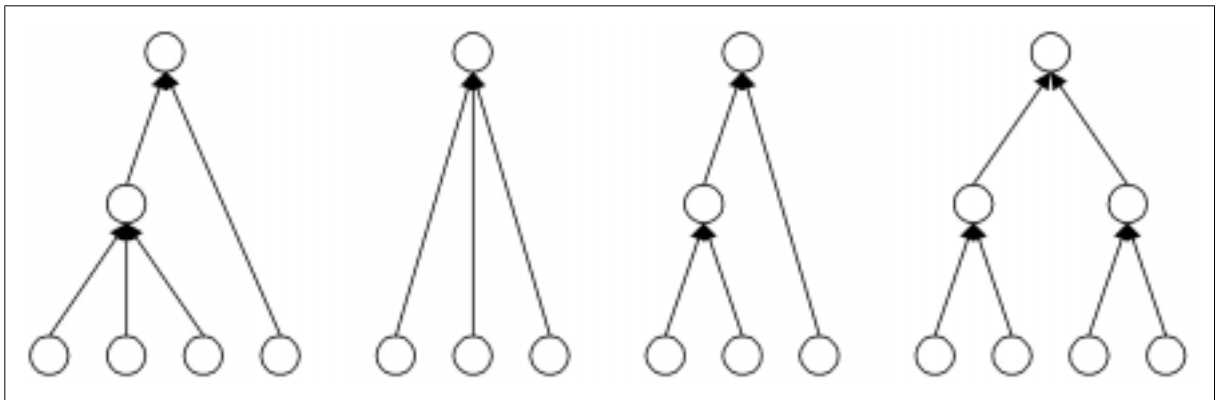


Abbildung 12 Integrationsbaum

Der Vorteil der binären Strategien ist im allgemeinen eine Vereinfachung der Aufgaben während der Vergleichs- und Konfliktbehandlungsphase. Möglich ist dies, da jeweils nur die minimale Anzahl von Sichten (zwei) gleichzeitig betrachtet werden muß, wodurch die Komplexität eines Integrationsschrittes begrenzt wird. Die Leiterstrategie ermöglicht durch die Priorisierung der Schemata eine einfache Konfliktbehandlung, die balancierte Strategie hingegen bietet die Möglichkeit der parallelen Ausführung von Integrationsschritten. Der Nachteil des binären Ansatzes liegt dabei in der erhöhten Anzahl der Schritte gegenüber der n-ären Technik.



a) n-äre Strategie	b) One-Shot	c) Leiter-Strategie	d) balancierter Baum

Abbildung 13 Integrationsstrategien

4.2.3.4 Beziehungen zwischen den Sichten

Bei der Sichtenintegration ist es wichtig, nicht nur die gemeinsamen Konzepte in das neue Schema aufzunehmen, sondern auch die Unterschiede herauszuarbeiten, die in einer semantischen Beziehung zueinander stehen. Nach [BLN86] werden diese als *inter-*

schema-properties bezeichnet. Um den Kriterien Vollständigkeit und Korrektheit gerecht zu werden, müssen auch diese in der Gesamtsicht repräsentiert sein. Verwendet man zur Darstellung Klassenmodelle, so sind diese inter-schema-properties typischerweise Generalisierungsbeziehungen oder Aggregationsbeziehungen. Aber auch beliebige Assoziationen zählen dazu.

Um die Konflikte in der Vergleichsphase zu verringern, ist es sinnvoll, bereits während der Erfassung der Einzelsichten Beziehungen zu anderen Schemata möglichst vollständig mit in die eigene Sicht zu integrieren. Werden inter-schema-properties erst während der Integrationsphase erkannt, so werden diese je nach Status des Integrators in Rücksprache mit den Sichteninhabern (moderierend) oder eigenverantwortlich (omnipotent) in das Gesamtkonzept übernommen.

4.2.3.5 Sichtenevolution

Die Lebensdauer einer Gesamtsicht ist meist zeitlich begrenzt. Durch Weiterentwicklung ändern sich die zugrunde gelegten Anforderungen, werden verfeinert oder überarbeitet. Dieser Evolutionsprozeß verläuft keineswegs monoton. Eine Verfeinerung kann im Hinzufügen von Details bestehen, aber auch das Verwerfen bereits gewonnener Fakten bedeuten. Die Anforderungsspezifikation entwickelt sich also mit den Zyklen der Anwendung weiter.

Dies steht ein wenig im Widerspruch zur der im Datenbankbereich eingesetzten Schemaintegration, da diese in der Regel eine einmalige Aktion ist. Da bei der Sichtenintegration aber wiederholt eine Anforderungsanalyse vorgenommen wird, müssen auch bereits erkannte Gemeinsamkeiten erneut überprüft werden. So können durch eine geänderte Modellierung zuvor festgelegte Äquivalenzen ihre Gültigkeit verlieren.

Ähnliches gilt für die Problematik der Konfliktbehandlung. Zuvor gelöste Inkonsistenzen können durchaus wieder auftreten, geben also nicht unbedingt eine Dauerlösung ab. Jeder Sichteninhaber kann seine Sicht unabhängig weiterentwickeln, und so die Konsistenz der Gesamtsicht zunichte machen. Somit ist eine erfolgreiche Konsistenzprüfung nur zum Zeitpunkt der Ausführung gültig.

4.2.4 Schemaevolution

Ein natürliches Phänomen von Datenbanken ist ihre Langlebigkeit. Im Gegensatz dazu ist die reale Umwelt, die in solchen Datenbanken modelliert wird, häufigen Veränderungen und Anpassungen unterworfen.

Dazu können neue Anforderungen oder Ergänzungen gehören, die beim ersten Entwurf des Modells noch nicht bekannt waren. Typische Beispiele sind die Ergänzung von geometrischen Formen in CAD-Systemen durch Fertigungsangaben für die Endmontage oder die Veränderung einer Lagerhaltung durch die zunehmende Automatisierung.

Auch der Wegfall oder Ersatz von Applikationen, die auf die Datenbank zugreifen, kann solche Änderungen bewirken.

Das im Rahmen der Diplomarbeit entwickelte Informationssystem *EvolMat* ist als Datenbankapplikation implementiert worden, um der Anforderung einer langfristigen Speicherung von Datenobjekten gerecht zu werden. Da sich im Laufe der Zeit aber die benötigten Informationen der Simulationsprogramme, die auf das Informationssystem zugreifen, ändern und neuen Forschungsergebnissen in der Kunststofftechnik angepaßt werden, muß das zugrunde liegende Schema angepaßt werden. Hier kommt als besondere Schwierigkeit hinzu, daß das Schema nicht nur an die unterschiedlichen Simulationsprogramme angepaßt werden muß, sondern diese selbst, bedingt durch teilweise langfristige Projekte, selbst einem Wandel in ihren Datenstrukturen unterworfen sind. Diese Transformation von einem alten zu einem neuen Schema bezeichnet man als *Schemaevolution* [Lang95]

Änderungen am Design der modellierten Datenstrukturen bedingen oftmals umfangreiche Umstrukturierungen. Um den so entstehenden Aufwand zu minimieren, greift man gerne auf Modellierungswerkzeuge zurück, die einem die Möglichkeit bieten, ein konzeptionelles Schema mehr oder minder automatisch auf ein logisches Schema abzubilden. Ein typischer Vertreter dieser Gattung ist das Werkzeug RONDO [RON94], welches die Schemaerstellung auf Basis von EER-Modellen und die automatische Abbildung in ein konzeptionelles relationales Schema ermöglicht.

Im Rahmen der Diplomarbeit wird dafür das am Lehrstuhl Softwaretechnik der Universität Paderborn entwickelte Reengineering-Werkzeug VARLET eingesetzt. Mit Hilfe von VARLET ist es möglich, sowohl ein relationales Schema in ein objektorientiertes zu überführen, wie auch umgekehrt. Die Überprüfung der Konsistenz zwischen den beiden Schemata übernimmt dabei VARLET.

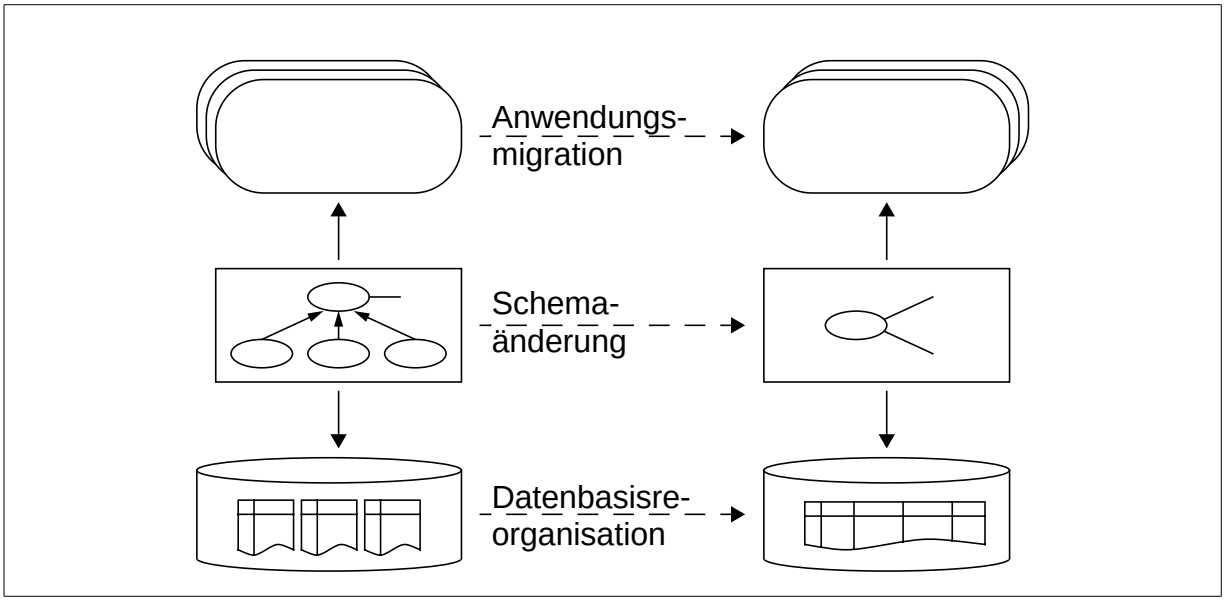


Abbildung 14 Vertikale Evolution in Objektdatenbanken [Tre95]

Unter dem Aspekt des täglichen Einsatzes des Informationssystem *EvolMat* darf die Anpassung des Datenbestandes an die neuen Modelle nicht vergessen werden. Dabei müssen nacheinander die Anwendung, das Schema und die zugrunde liegenden Daten migriert werden. Eine solche schrittweise Anpassung des Datenbanksystems bezeichnet man als *vertikale Evolution* (Abbildung 14).

Vorgabe für die Entwicklung des Informationssystems *EvolMat* war eine einfache und sichere Handhabung sowohl der Legacy-Daten, als auch die Möglichkeit flexibel auf Änderungen zu reagieren. Aus diesen Gründen ist eine manuelle Anpassung der Daten und Schemata nicht adäquat, da sie sehr zeitaufwendig und fehleranfällig ist.

4.2.5 VARLET

Der Umgang und Handel mit Informationen wird zunehmend zu einem der wichtigsten Punkte für die Wettbewerbsfähigkeit heutiger Firmen. Diese Informationen müssen möglichst schnell und von jedem Ort aus abrufbar sein. Der Einsatz aktueller Schlüsseltechniken wie das WWW, Client/Server-Systeme oder verteilter Systeme, hat großen Einfluß auf die Entwicklung von Informationssystemen. Neben neu entwickelten System ist es für viele Firmen enorm wichtig auf bereits vorhanden Daten- und Informationsbestände zugreifen zu können. Dies stellt sich jedoch oft als schwierig heraus, da solche älteren Informationssysteme oft über mehrere Generationen von Programmierern entwickelt wurden und erfahrungsgemäß nur eine schlechte Dokumentation bieten. Diese Systeme werden als Legacy Informationssysteme (IS) bezeichnet.

Die Überarbeitung und Anpassung solcher Systeme an heutige Standards wird als Reengineering bezeichnet. Dieser Prozeß hat in den letzten Jahren zunehmend an Bedeutung gewonnen. Viele Konzepte und Methoden sind entwickelt worden, um das Reengineering von Legacy IS zu vereinfachen. Die Implementierung dieser Methoden führte zur Entwicklung erster CARE[1]-Werkzeuge, um den Reengineering-Prozeß zu vereinfachen und zu automatisieren.

Die meisten der heutigen CARE-Werkzeuge arbeiten sehr stark phasenorientiert, auch Wasserfall-Modell, genannt. Dabei werden bestimmte, vorgegebene Abschnitte des Prozesses durchlaufen, ohne die Unterstützung von Iterationen der einzelnen Phasen zu bieten. Dies stellt aber gerade eine Einschränkung des Reengineering dar, da dieser Prozeß vielmehr durch eine ständige Untersuchung und Evolution geprägt ist. Neue oder geänderte Anforderungen müssen berücksichtigt werden, oder getroffene Entscheidungen aufgrund von neuen Erkenntnissen revidiert werden. Bereits getroffene Entscheidungen und durchgeführte Änderungen bei der Migration von Legacy IS gehen bei Anwendung des Wasserfall-Modells jedoch verloren. Diesen Nachteil heutiger CARE-Werkzeuge wurde in VARLET beseitigt. VARLET bietet auf der Basis von Graphersetzungsregeln [Ros97] eine inkrementelle Konsistenzüberwachung des gesamten

1. CARE = Computer Aided Reengineering

Reengineering-Prozesses. Dies bietet die Möglichkeit, die Schemaanalyse und die Entscheidungen zum Redesign in einem iterativen und interaktiven System zu vereinen.

Die Basis der Konsistenzprüfung bildet ein abstrakter Syntaxgraph, der die Beziehungen zwischen dem relationalen und dem migrierten objektorientierten Schema abbildet. Diese Technik des Mapping wird im folgenden erläutert.

4.2.5.1 Das Mapping in VARLET

Hinter der Idee der Mapping-Technik verbirgt sich ganz allgemein die Möglichkeit, die Änderungen an zwei miteinander in Beziehung stehenden Dokumenten zu verwalten, und auf das jeweils andere zu übertragen. Ein Ansatz der diese Bedingungen erfüllt sind die sogenannten *Triplegraphgrammatiken* nach [Lef95].

Die Grundidee der Triplegraphgrammatiken basiert auf der Integration zweier Dokumente mittels eines *Integrationsdokumentes*. Dieses dient der Verwaltung der Beziehungen und deren Veränderungen zwischen den beiden Dokumenten [Hol97]. Auf diesem Ansatz basierende Integrationswerkzeuge bieten zudem eine Konsistenzprüfung und Transformation von Dokumenten. In VARLET werden als Dokumente im wesentlichen die Gegenüberstellung von relationalem und objektorientiertem Schema verwaltet.

Die Integration selbst wird durch einen Satz von *Tripleregeln* beschrieben. Sie enthalten Informationen darüber, wie die Schemata ineinander überführt werden können. Dazu wird im Integrationsdokument gespeichert, welche Teile der jeweiligen Schemata miteinander in Beziehung stehen.

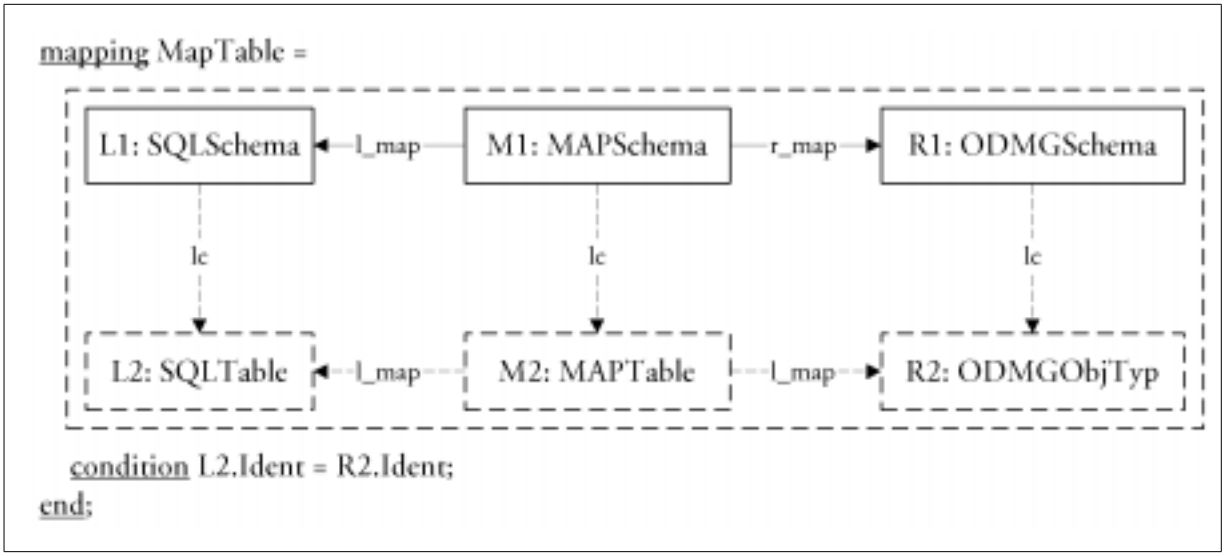


Abbildung 15 Spezifikation einer Tripleregel [Hol97]

Die Regeln werden in einer graphischen Notation (Abbildung 15) angegeben, die im wesentlichen aus zwei Teilen besteht: Der *Vorbedingung* und dem *Ausführungsteil*. Die Vorbedingung ist mit durchgezogenen Knoten und Kanten markiert. Ist diese Vorbedingung erfüllt,d.h. es existiert ein entsprechendes Beziehungsmuster im

Integrationsdokument, so kann dieses um den durch die gestrichelten Knoten und Kanten markierten Teil erweitert werden. Die folgende Abbildung beschreibt die Integration einer Tabelle mit einem Objekttyp, falls die zugehörigen Schemata entsprechend der Vorbedingung miteinander in Beziehung stehen.

Der linke Teil der Regel enthält die Regeln zur Produktion eines SQL-Dokumentes. Entsprechend sind im rechten Teil die Produktionen zur Erzeugung eines ODMG-Dokumentes beschrieben. Der mittlere Teil enthält schließlich die korrespondierenden Beziehungen zwischen den beiden Dokumenten.

Durch die Verwendung der Tripelregeln und der daraus generierten Spezifikation erfolgt die Beschreibung der Integration auf eine hohen Abstraktionsebene, aus der automatisch Spezifikationen für Graphersetzungssysteme[1] abgeleitet werden können. Mit Hilfe der Graphersetzungssysteme ist es möglich, eine Vorwärtstransformation und Rückwärtstransformation der Dokumente sowie deren Konsistenzprüfung zu realisieren. Durch die symmetrische Behandlung von Quell- und Zieldokument ergibt sich somit die Möglichkeit der bidirektionalen Transformation, also der Abbildung eines relationalen Schemas auf ein objektorientiertes Schema und umgekehrt.

Bei der Konsistenzüberprüfung muß zwischen zwei Begriffen unterschieden werden: *Konsistenz* und *Äquivalenz*. Die Abgrenzung der beiden Begriffe gegeneinander erfolgt auf Basis der *Informationskapazität* eines Schemas. Darunter versteht man die Menge aller gültigen Zustände der Komponenten des Schemas, also alle Kombinationen von Instanzen und ihren Beziehungen zueinander [Wad98].

Die Konsistenz wird gewahrt, wenn das objektorientierte Schema mittels der initialen Transformation und eventueller manueller Restrukturierungen aus dem relationalen Schema gewonnen werden kann.

Enthalten die Quell- und Zielschemakomponenten genau die gleichen Informationen, d.h. beide haben die gleiche Informationskapazität, spricht man von Äquivalenz.

---

1. Ein solches System ist PROGRES (PROgrammierte GRaphERsetzungssysteme [Zün95]), Grundlage für VARLET

5 Das Informationssystem EvolMat

5.1 Modellierung des Gesamtkonzeptes

Grundlage und Ziel des Informationssystems EvolMat sind die vom KTP entwickelten Simulationsprogramme. Im ersten Schritt werden die Programme und die von Ihnen benötigten Materialdaten analysiert. Die daraus gewonnenen Informationen dienen der Entwicklung der konzeptionellen Schemata für die einzelnen Simulationsprogramme (siehe Kapitel 5.2.2).

Bei der Entwicklung der Komponentenschemata ist mit Hinblick auf den folgenden Schritt der Schemakonsolidierung darauf geachtet worden, mögliche Konflikte im Vorfeld zu lösen. Ein wichtiger Aspekt dabei ist der Namenskonflikt. Da alle vier Simulationsprogramme aus dem Gebiet der Kunststofftechnik stammen, sind die Variablennamen in den Komponentenschemata entsprechend gewählt. Diese Nomenklatur ist in der Standardliteratur für die Kunststofftechnik verzeichnet ([Pot92], [Rao89]). Die Konsolidierung der Schemata erfolgt mit den in Kapitel 4.2.3 vorgestellten Methoden.

Das so entwickelte Gesamtschema dient als Grundlage für den Aufbau der Datenbank und der Anbindung. Zunächst wird das Schema in VARLET eingegeben und weiter bearbeitet. Später nötig werdende Änderungen des Schemas werden nur noch im Gesamtschema vorgenommen. Die Veränderungen die sich daraus ergeben werden dann mittels der Schemaevolution weiter behandelt und berücksichtigt. Mit Hilfe der in VARLET vorhandenen *TextViews*[1] können die aus dem Gesamtschema benötigten Informationen dargestellt werden. Dies ist zum einen der SQL-View, der die Informationen des relationalen Schemas enthält. Zum anderen ist ein neuer TextView generiert worden, der die ODMG-konformen Informationen über das objektorientierte Schema enthält. Diese werden im XML-Format generiert, somit wurde VARLET um eine standardisierte Schnittstelle erweitert.

Die so erzeugten Definitionen dienen als Grundlage für die Anbindung der Middleware ObjectDRIVER. Basierend auf diesen Eingaben kapselt ObjectDRIVER den relationalen Zugriff auf die Datenbank und bietet dem Programmierer über die beiden Schnittstellen (Java-API[2], C++-API) einen objektorientierten Zugriff auf das zur Verfügung gestellte virtuelle OODBMS (siehe Kapitel 4.2.2.3).

Über die entsprechende API kann nun EvolMat auf die benötigten Materialdaten zugreifen. EvolMat enthält auch die Informationen darüber, welche Daten aus dem

1. TextViews sind eine Erweiterung der Standardausgaberoutinen von PROGRES, siehe [Hol97]
2. API = Application Programming Interface

Gesamtschema die einzelnen Simulationsprogramme benötigen. Durch Generierung entsprechender Filter, also einer Sicht auf das Gesamtschema, werden die Daten den

Programmen zur Verfügung gestellt. Da die Simulationssoftware zur Zeit nur dateibasiert arbeitet, werden die entsprechenden Informationen als Textdatei exportiert.

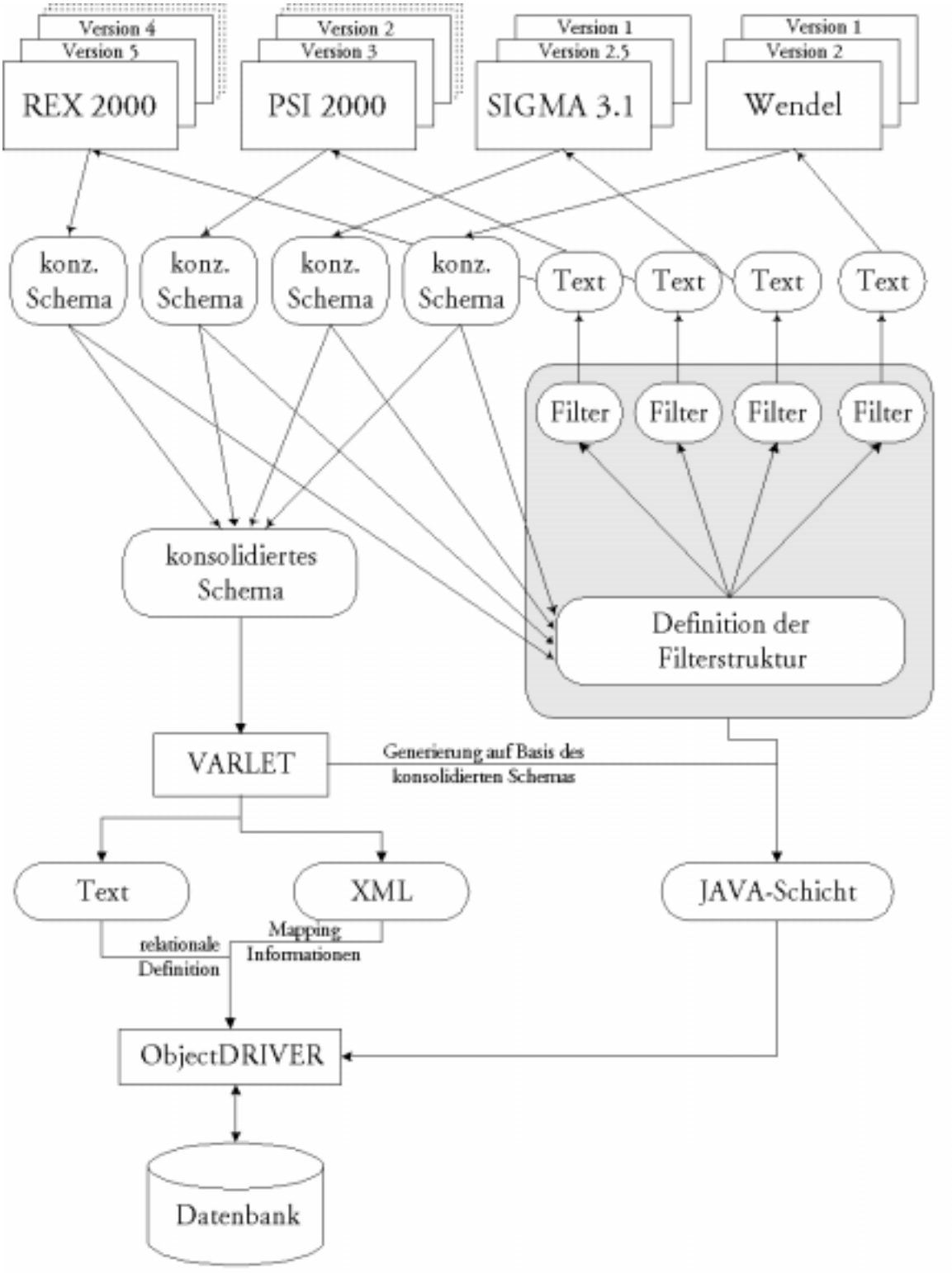


Abbildung 16 Schemazeichnung des Gesamtkonzepts

5.2 Datenbasis

Während der Anforderungsanalyse hat sich gezeigt, daß die von EvolMat zu verwaltenden Daten zuerst einmal entsprechend aufbereitet werden müssen. Bislang werden die Materialdaten von jedem der Simulationsprogramme in einem eigenen Dateiformat geschrieben und gelesen. Dadurch ist der Austausch der Materialdaten unter den Programmen nicht möglich und die Daten müssen für jedes Simulationsprogramm neu erfaßt werden. Ziel der Diplomarbeit ist es, eine gemeinsame Datenbasis für alle Programme zu schaffen. Dafür müssen die Daten zunächst analysiert und dann strukturiert werden.

5.2.1 Klassifizierung der Daten

Ein Datensatz, der die Eigenschaften des in der Simulation verwendeten Materials beschreibt, besteht aus circa 50 Einzeldaten. Da die Anwendungsdomäne aus dem Bereich der Kunststofftechnik stammt, ist es naheliegend, die Klassifizierung an fachlichen Vorgaben aus diesem Gebiet zu orientieren. Eine erste Strukturierung der Materialdaten ergibt sich durch die Gruppierung nach physikalischen Eigenschaften sowie den Daten, die einer allgemeinen Beschreibung des Materials dienen, wie z.B. der Materialname oder das Erstellungsdatum des Datensatzes. Die Begrifflichkeit ist dabei an die verschiedenen Forschungsbereiche und Lehrgebiete der Kunststofftechnik angelehnt. Diese erste Unterteilung ergibt folgende Gruppen:

Rheologiedaten	Dichtedaten	Thermodynamikdaten
Tribologiedaten	Technologiedaten	

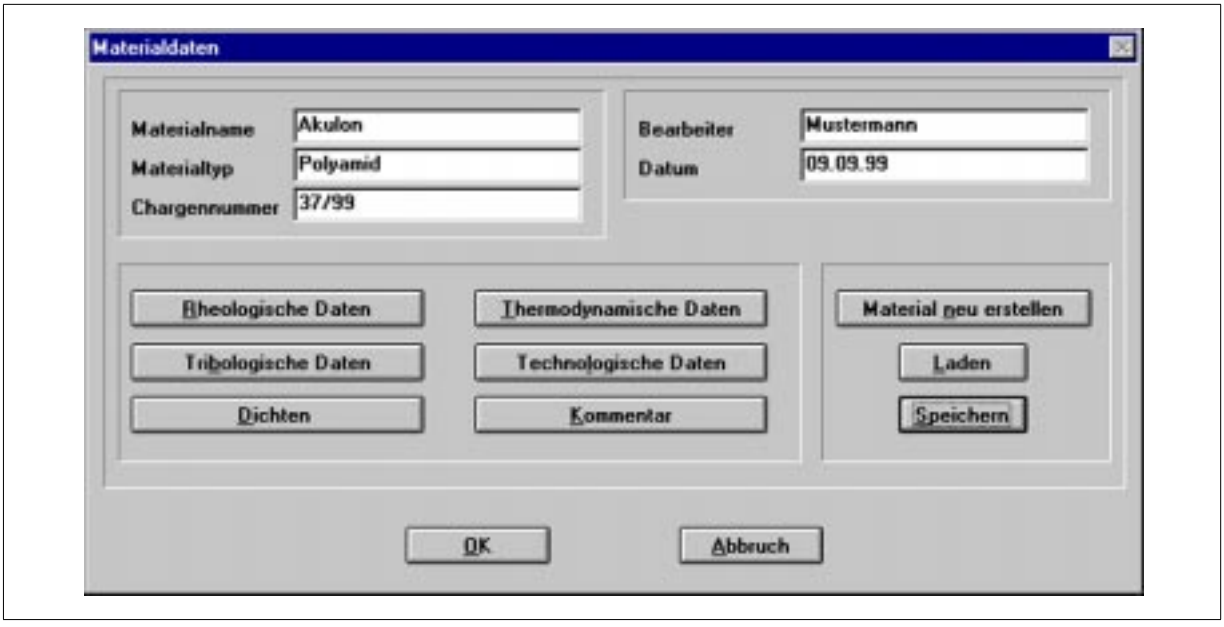


Abbildung 17 Eingabemaske für die Materialdaten in REX

Diese Unterteilung nach physikalischen Eigenschaften ist nicht nur eine strukturelle Gliederung der Materialdaten, sondern spiegelt auch im wesentlichen die Eingabedialoge der Simulationsprogramme wieder. Als Beispiel ist die Eingabemaske aus dem Simulationsprogramm REX abgebildet (Abbildung 17). Auf Basis der physikalischen Eigenschaften lassen sich die weiteren Materialdaten jeweils eindeutig zu einer der Gruppen zuordnen (Abbildung 18).

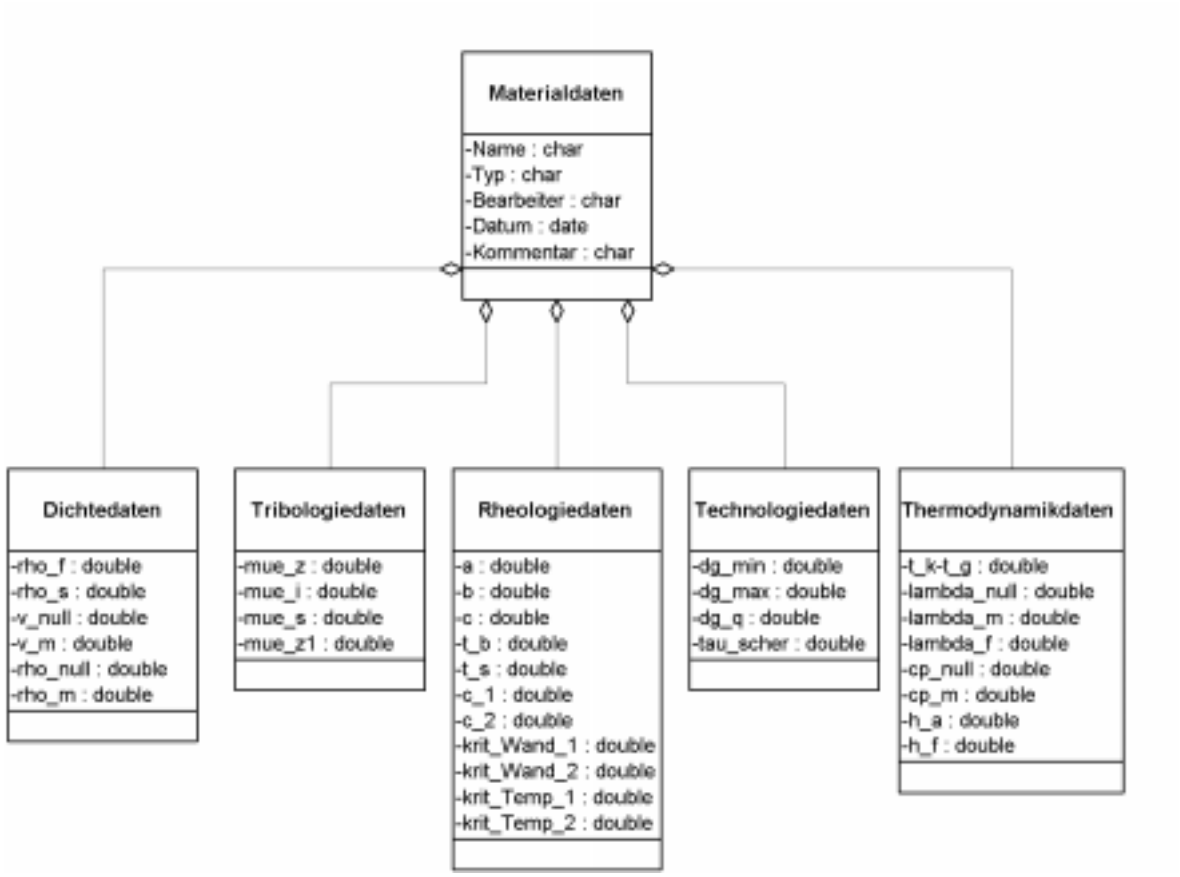


Abbildung 18 Gruppierung nach physikalischen Eigenschaften

Da diese Unterteilung noch sehr grob ist, müssen weitere Merkmale gefunden werden, die eine Verfeinerung der Struktur erlauben. Dazu ist jedoch zuerst ein wenig Kenntnis über die einzelnen Materialeigenschaften nötig. Die teilweise wenig intuitive Nomenklatur der verwendeten Variablen ist dabei an die Formelsprache der Kunststofftechnik angelehnt (siehe [Pot92], [Rao89]).

- Rheologische Materialdaten
  Sie beschreiben das Fließverhalten von Flüssigkeiten. Diese lassen sich in zwei Gruppen aufteilen: *Newton´sche* Flüssigkeiten und *nicht Newton´sche* Flüssigkeiten. Kunststoffschmelzen sind durch die sogenannte Strukturviskosität gekennzeichnet und zählen daher zu den nicht Newton´schen Fluiden. Ihre Viskosität ist nicht mehr konstant, wie bei Newton´schen Flüssigkeiten, sondern in starkem Maße von der *Schergeschwindigkeit* abhängig. Zur Berechnung des Aufschmelzvorgangs gibt es verschiedene Modelle. Alle Ansätze greifen dabei auf die materialspezifischen Kon-

stanten zurück: Die Nullviskosität a[1], die Übergangsgeschwindigkeit b, die Steigung der Funktion c und die Bezugstemperatur t_b . Die drei verwendeten Ansätze mit ihren Parametern zur Berechnung der Carreauwerte sind:

- Carreau-WLF[2] (Bezugstemperatur t_b, Standardtemperatur t_s)
- Carreau-WLF (1. und 2. Konstante des WLF-Ansatzes c_1, c_2)
- Carreau-Arrhenius (Aktivierungsenergie e, Bezugstemperatur t_b)

Mit dem Carreau-Ansatz läßt sich das kunststoffspezifische Verhalten über große Temperaturbereiche korrekt beschreiben.

Zusätzlich kann mit den Simulationsprogrammen PSI und REX ein *wandgleitendes* Material berechnet werden. Dieser Effekt tritt auf, wenn die *Schubspannung* (die Kraft, mit der das aufgeschmolzene Material an der Zylinderwand vorbeigeschoben wird) des Materials zu groß ist. Kommt ein solches Material in der Simulation zum Einsatz müssen zusätzlich zu Charakterisierung des Fließgesetzes mittels der Carreau- oder Arrhenius-Parameter zwei Wertepaare (krit_Temp, krit_Wand) eingegeben werden. Diese Werte enthalten jeweils eine Prüftemperatur und die bei dieser Prüftemperatur gemessene kritische Wandschubspannung.

- Dichtespezifische Materialdaten
  Sowohl die spezifische Dichte als auch das spezifische Volumen lassen sich mit Hilfe von *pvT-Diagrammen* (siehe Kapitel 3.1 Abbildung 8) ermitteln. Dabei wird für eine Kurvenschar, die den Druck p als Parameter hat, das spezifische Volumen v über die Temperatur T aufgetragen. Mit den aus diesen Diagrammen ermittelten Werten kann die Volumen- bzw. Dichtefunktion berechnet werden. Für beide Funktionen wird jeweils der Wert bei 0˚ C angegeben (v_0 bzw. rho_0) sowie die Steigung der zugehörigen Funktion (v_m bzw. rho_m). Zusätzlich werden hier die Dichten für den Feststoff (rho_f) bzw. die Schüttdichte (rho_s) des verwendeten Granulats angegeben. Auch hier zum Vergleich die zugehörige Eingabemaske aus den Simulationsprogrammen (Abbildung 19).
- Thermodynamische Materialdaten
  Die thermodynamischen Daten beschreiben alle Vorgänge des Wärmeausgleichs innerhab des Materials. Dabei wird auch die durch Reibung entstehende Wärme (*Dissipation*) berücksichtigt. Die Unterteilung der Eigenschaften erfolgt hierbei nach Wärmeübertragungsmechanismen. Diese sind die *Wärmeleitfähigkeit* und die *Konvektion*. Die Wärmeleitfähigkeit läßt sich mit den drei Parametern lambda_0 (Wärmeleitfähigkeit bei 0˚ C), lambda_f (Wärmeleitfähigkeit des Feststoffes) und lambda_m (Steigung der Wärmeleitfähigkeitsfunktion) beschreiben. Bei der Konvektion werden über die Angabe der Molekülstruktur die benötigten Parameter fest-

1. Vollständige Übersicht der Variablennamen siehe Anhang B: Materialdaten
2. benannt nach den Entwicklern des Modells: Wiliams, Landel und Ferry

gelegt: cp_0 (Wärmekapazität bei 0˚ C), cp_m (Steigung der Wärmekapazitätsfunktion) und h_f (Feststoffenthalpie) für amorphe Kunststoffe, zusätzlich h_a (Aufschmelzenthalpie) für teilkristalline Materialien.

- Tribologische Materialdaten
  Die tribologischen Daten geben die Reibungskoeffizienten wieder. Die Reibwerte sind abhängig von der Temperatur, dem Druck und der Gleitgeschwindigkeit. Sie sind für viele Materialien bekannt und können zum Beispiel in [VDMA84] nachgelesen werden. Zur Berechnung der Koeffizienten wird die Reibung an verschiedenen Stellen innerhalb der Maschine berücksichtigt:
  - zwischen Kunststoff und Schnecke (Schneckenreibungskoeffizient mue_s)
  - zwischen Kunststoff und Zylinder (Zylinderreibungskoeffizient mue_z)
  - zwischen Kunststoff und Kunststoff (innerer Reibungskoeffizient mue_i)
  - zwischen Kunststoff und Zylinder in Abhängigkeit von der Temperatur (temperaturabhängiger Reibungskoeffizient mue_z1)
- Technologische Daten
  Zur Berechnung des Druck-Durchsatzverhaltens einer Schneckenmaschine werden je nach Art der Feststoffförderung zusätzlich zu den tribologischen Daten auch die technologischen Daten benötigt. Dies sind die minimale (dg_min), mittlere (dg_q) und maximale (dg_max) Korngröße des Materials, sowie die Scherfestigkeit des Feststoffes (tau_scher).

Um eine feinere Gliederung der Materialeigenschaften zu erhalten, müssen geeignete Kriterien für die Aufteilung gefunden werden. Berücksichtigt man die Anwendungsdomäne lassen sich zwei Möglichkeiten zur Unterteilung der Eigenschaften finden.

1. Die Verwendung der Eigenschaften in den Simulationsprogrammen
   Hierbei lassen sich die Daten prinzipiell in zwei Gruppen unterteilen. Einige der Daten werden unverändert in die Simulationsberechnungen übernommen. Der andere Teil dient als Grundlage für die Berechnung von Zwischenergebnissen. Diese Ergebnisse fließen dann mit in die Simulation ein. Diese Gruppierung läßt sich zum Beispiel sehr schön bei den Dichtedaten nachvollziehen. Dabei gehen rho_f und rho_s als unveränderte Parameter in die Berechnung ein, während aus den beiden anderen Wertepaaren (rho_null und rho_m bzw. v_null und v_m) jeweils Funktionen zur Berechnung der spezifischen Dichte bzw. des spezifischen Volumens gebildet werden. In dieser Unterteilung spiegeln sich auch die physikalischen Modelle, die in der Simulation zum Einsatz kommen, wieder. Diese Unterteilung läßt sich allerdings nicht für alle Materialeigenschaften nachvollziehen. Innerhalb der thermodynamischen Daten gibt es einige Variablen, die sowohl direkt in die Berechnung einfließen, aber auch zur Berechnung weiterer Funktionswerte dient.

2. Notwendigkeit der Daten für die Simulationsberechnung
   Wie zuvor erwähnt, werden innerhalb der Simulation verschiedene Modelle als Grundlage der Berechnung verwendet. Für die Berechnung der Rheologiedaten sind dies zum Beispiel die drei verschiedenen Carreau-Ansätze. Zwar sollen mit dem Informationssystem alle Materialeigenschaften verwaltet werden, um eine Berechnung starten zu können sind allerdings nicht zwingend alle Angaben nötig. Es muß jedoch sichergestellt werden, daß die minimale Anzahl von Daten, die für eine Berechnung notwendig ist, zur Verfügung steht. Mit dieser Vorgabe lassen sich die Materialdaten wie folgt aufteilen:
   - Innerhalb der Rheologiedaten gibt es die drei Ansätze nach Carreau, von denen bereits einer ausreicht, um eine Simulationsberechnung starten zu können. Die ebenfalls zu den rheologischen Daten zählenden Eigenschaften für das Wandgleiten sind optional, da nicht jedes Material diese Eigenschaft besitzt.
   - Für die Dichtedaten sind die Feststoffdichte (rho_f) und die Schüttdichte (rho_s) nötig, sowie eins der Wertepaare spezifisches Volumen (v_null, v_m) bzw. spezifische Dichte (rho_null, rho_m).
   - Für die Berechnung der Thermodynamik sind die Angaben zur Wärmekapazität und Wärmeleitfähigkeit notwendig, sowie die Auswahl der Molekühlstruktur des verarbeiteten Materials. Dies wird durch die Angabe der Aufschmelzenthalpie (h_a) für amorphe Kunststoffe und zusätzlich der Feststoffenthalpie für teilkristalline Werkstoffe erreicht. Die Angabe einer Molekülstruktur ist ausreichend.
   - Die Tribologiedaten werden komplett benötigt, so daß hier keine Unterscheidung getroffen werden kann.
   - Gleiches gilt auch für die Technologiedaten, ohne deren vollständige Eingabe keine Berechnung möglich ist.

Ähnlich wie bei der ersten Strukturierung der Materialdaten nach physikalischen Eigenschaften, entspricht auch die zweite Aufteilung im wesentlichen der Benutzerführung in den Simulationsprogrammen. Beispielhaft sie hier die Maske zur Eingabe der Dichtedaten gezeigt (Abbildung 19). Im oberen Teil erfolgt die Eingabe der zwingend notwendigen Parameter, in der unteren Hälfte besteht die Auswahl zwischen den beiden Berechnungsmodellen.

Aufbauend auf dieser Unterteilung nach physikalischen und für eine Berechnung notwendigen Daten, können jetzt die Komponentenschemata für die Simulationsprogramme entwickelt werden. Dabei entsprechen die in den Schemata verwendeten Klassen der zuvor gewählten Gliederung. Die Gliederung beinhaltet drei Gruppen von Klassen:

- Die erste Gruppe enthält nur die Klasse Materialdaten, deren Attribute die allgemeine Beschreibung des Materials wiedergeben, wie den Materialnamen oder den Materialtyp.
- In der zweiten Gruppe befinden sich alle Klassen, die der Unterteilung nach physikalischen Eigenschaften entsprechen. Sie haben die Attribute, die auf jeden Fall für eine Berechnung notwendig sind, unabhängig von weiteren Berechnungsmo-

dellen.

- Die dritte Gruppe enthält schließlich die Klassen, die die verschiedenen Berechnungsmodelle wiederspiegeln. Hier sind aus jeder Gruppe der physikalischen Eigenschaften die Daten jeweils einer weiteren Klasse nötig, um eine Berechnung starten zu können.

Ordnet man diese Gruppen schichtartig an, ergibt sich die in den folgenden Diagrammen (Abbildung 20 bis Abbildung 23) gezeigte Baumstruktur. Wichtig für die Entwicklung der Diagramme ist auch die Beziehung der Klassen zueinander sowie der Typ der Klasse.

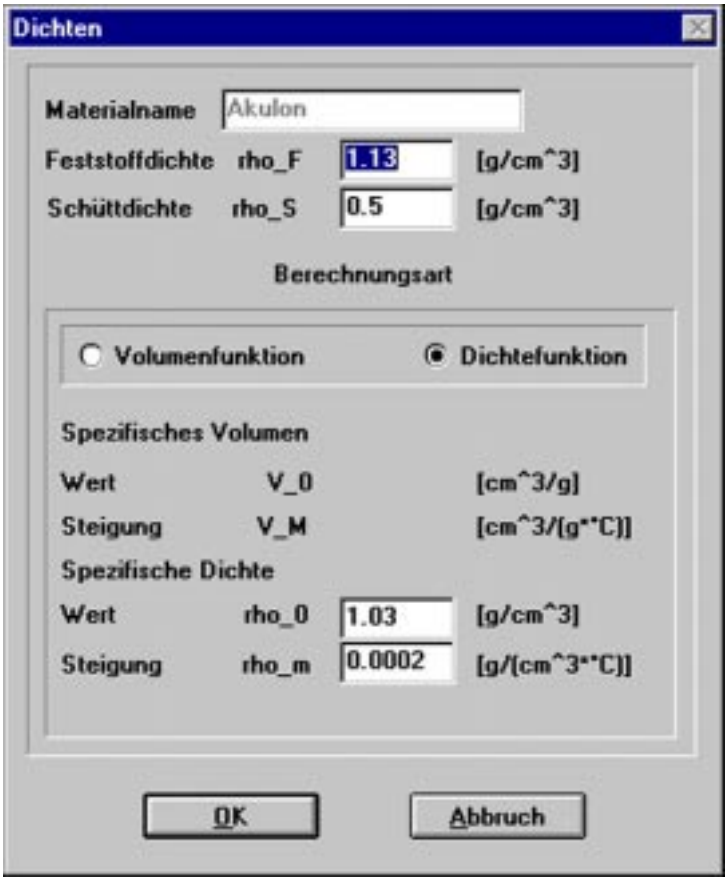


Abbildung 19 Eingabemaske für die Dichtedaten in PSI / REX

Zwischen der Hauptklasse *Materialdaten* und den fünf Klassen, die die Trennung nach physikalischen Eigenschaften repräsentieren, besteht jeweils eine Aggregationsbeziehung. Die Aggregation kann für diese Anwendungsdomäne als „besteht-aus“ Beziehung gelesen werden. Damit wird der Tatsache Rechnung getragen, daß sich die Materialeigenschaften genau aus diesen fünf Teilbereichen zusammensetzen. Alle diese Klassen müssen als abstrakte Klassen modelliert werden, sofern die in ihnen enthaltenen Daten noch nicht für eine Berechnung ausreichen. Da die Tribologiedaten und Technologiedaten vollständig benötigt werden, werden diese Klassen natürlich nicht abstrakt modelliert.

Die Klassen der dritten Gruppe werden mit Ausnahme der Klasse Wandgleiten als abgeleitete Klassen modelliert. Die Klasse Wandgleiten steht über eine Aggregation mit ihrer Oberklasse in Beziehung, da diese Daten nur optional sind. Alle Klassen dieser Gruppe sind als konkrete Klassen modelliert. Somit wird sichergestellt, daß ein Objekt einer dieser Klassen die für eine Berechnung mindestens benötigten Daten enthält.

Basierend auf diesen Vorüberlegungen können nun die Modelle der einzelnen Simulationsprogramme erstellt werden, um sie im Anschluß durch die Sichtenintegration in ein globales Konzept zu überführen. Die Diagramme werden, beginnend mit der kleinsten Anzahl an enthaltenen Materialeigenschaften, in der Reihenfolge WENDEL, SIGMA, PSI, REX entwickelt.

5.2.2 Erstellung der Komponentenschemata

Die Schemata für die einzelnen Simulationsprogramme beruhen auf der im vorherigen Abschnitt entwickelten Unterteilung der Materialdaten. Grundlage für die in den Schemata enthaltenen Materialeigenschaften sind die Daten, wie sie von den aktuellen Versionen der Simulationsprogramme verwendet werden. Als Grundlage für die zu entwickelnden Komponentenschemata dient die im vorherigen Kapitel festgelegte Gliederung der Materialeigenschaften. Da die Komponentenschemata im Anschluß zu einem globalen Schema konsolidiert werden sollen, wird schon bei der Erstellung versucht, mögliche spätere Konflikte zu vermeiden. Typische Probleme sind z.B.

Homonyme oder Synonyme. Daher wird in allen Schemata auf eine eindeutige Benennung der Variablen nach Vorgaben aus der Kunststofftechnik geachtet.

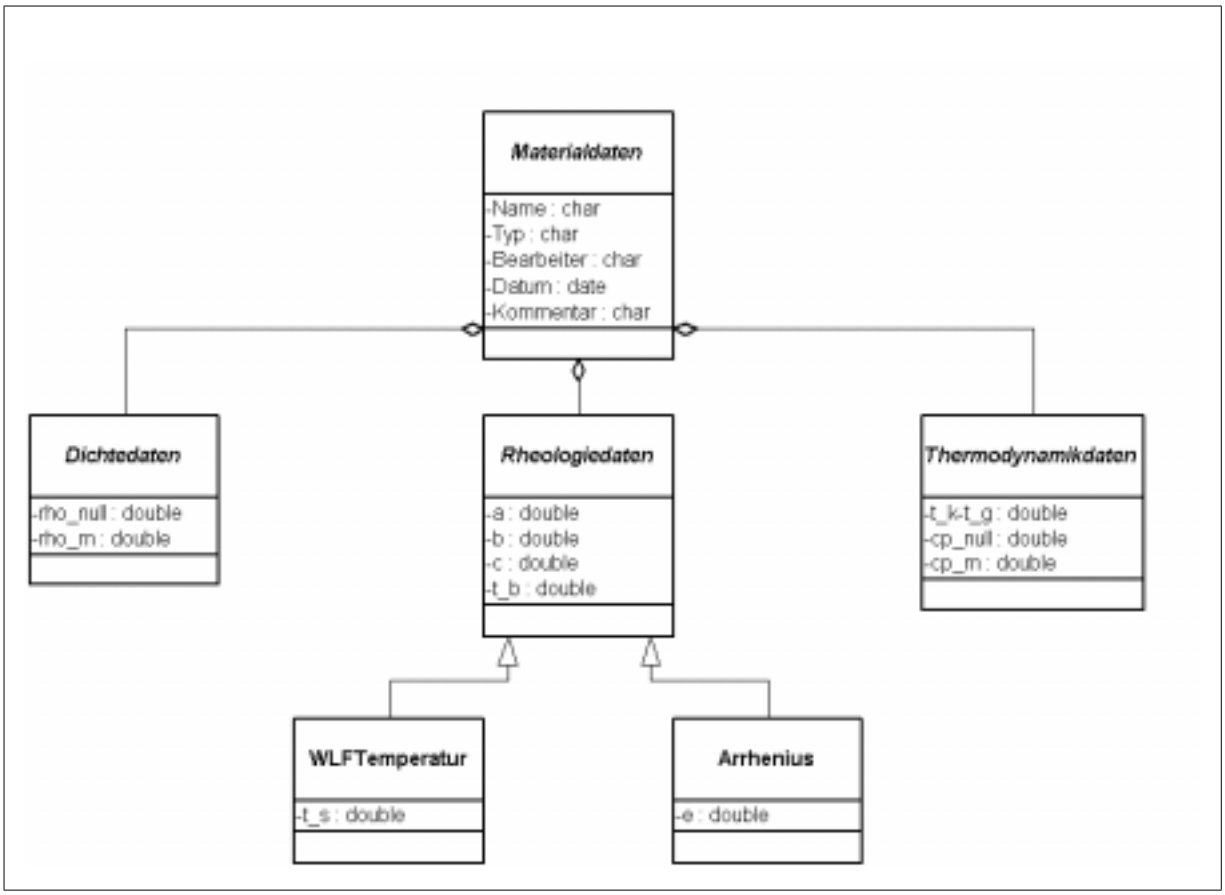


Abbildung 20 Modellierung der Materialdaten für WENDEL

Der Wendelverteiler benötigt den kleinsten Datensatz, da er nur ein Werkzeug darstellt, und keine Plastifiziereinheit. Neben den allgemeinen Materialdaten, werden zwei Dich-

teparameter, eine Teil der Rheologiedaten sowie Informationen über die Wärmekapazität (Thermodynamik) benötigt.

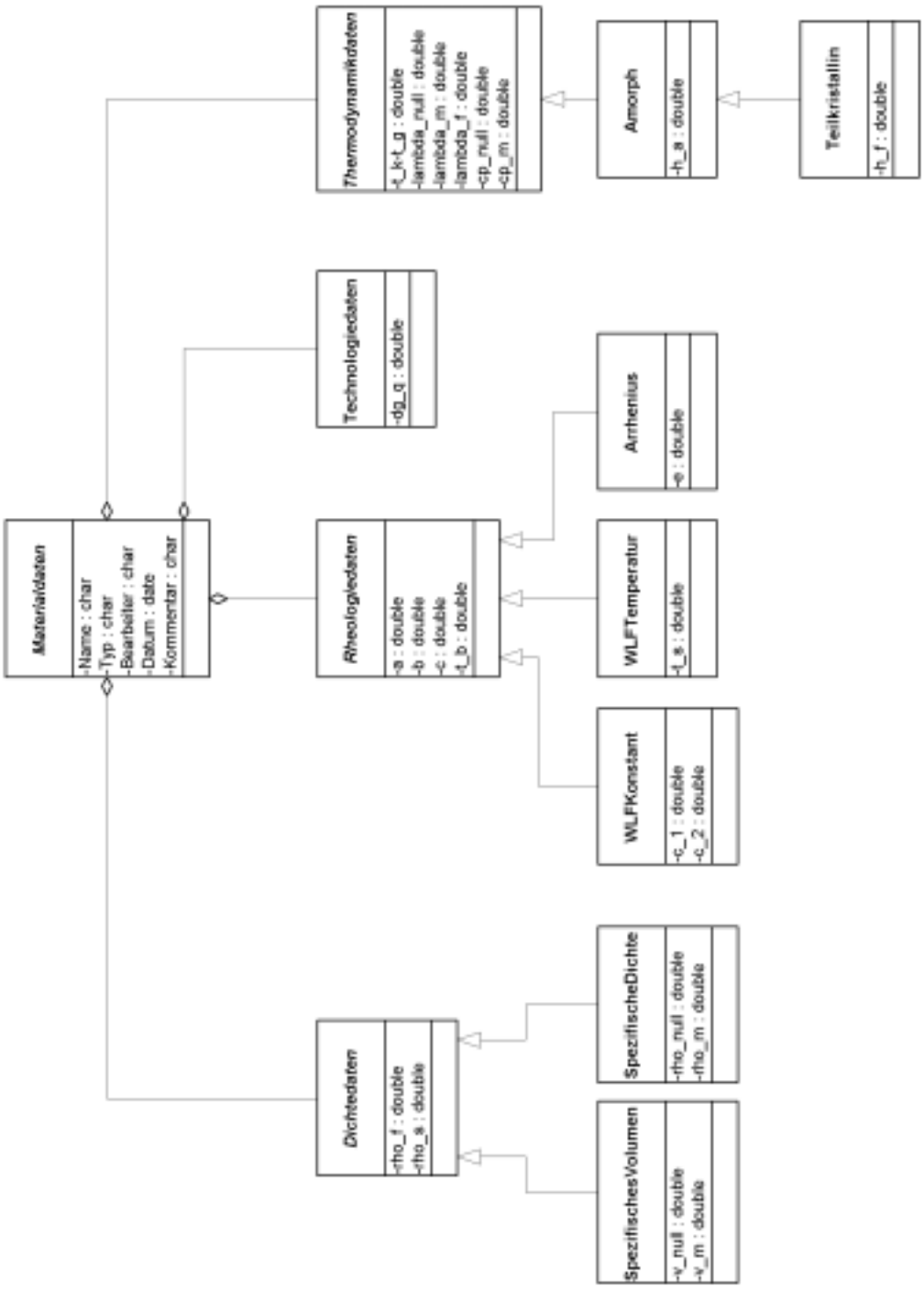


Abbildung 21 Modellierung der Materialdaten für SIGMA

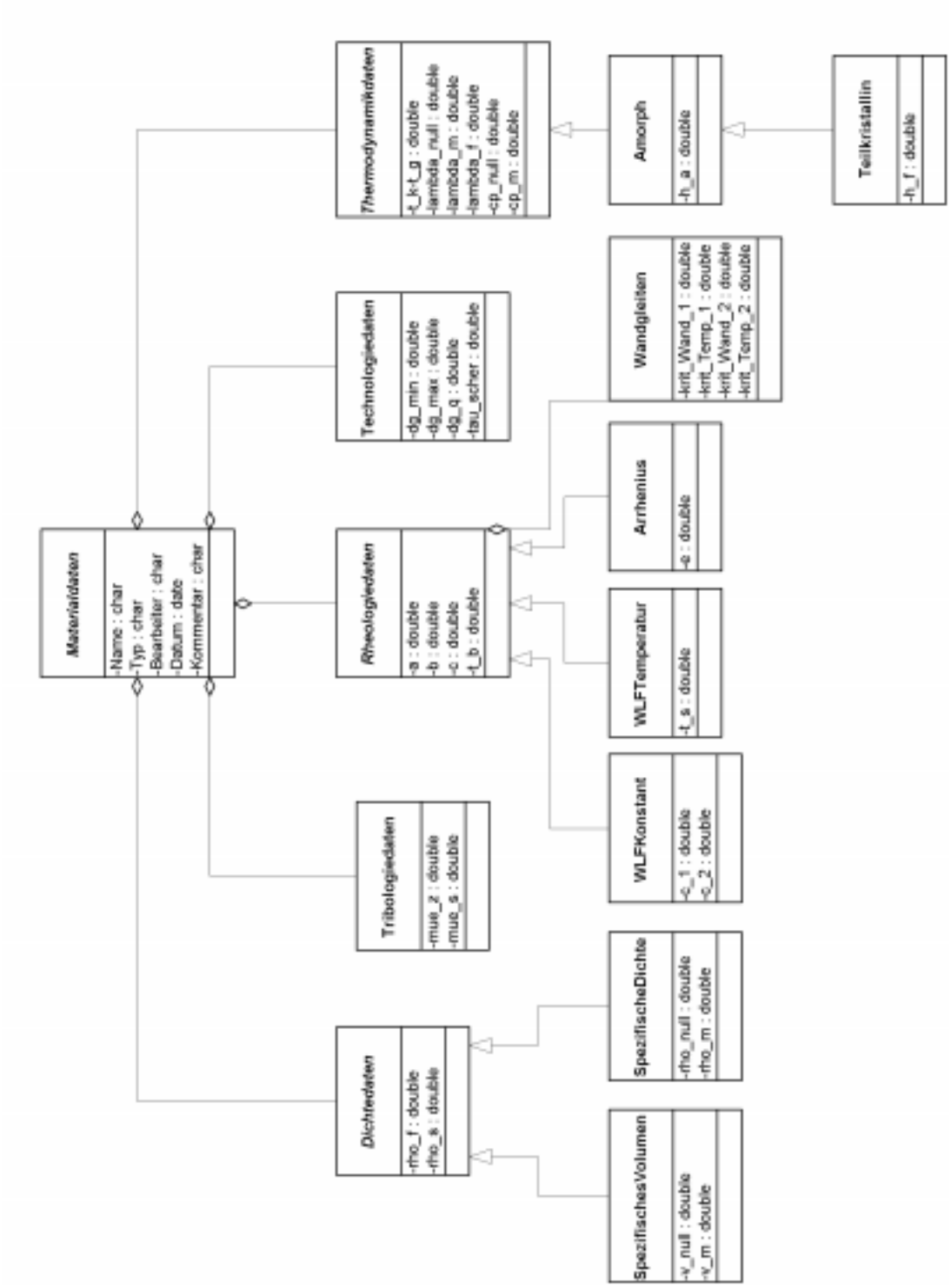


Abbildung 22 Modellierung der Materialdaten für PSI

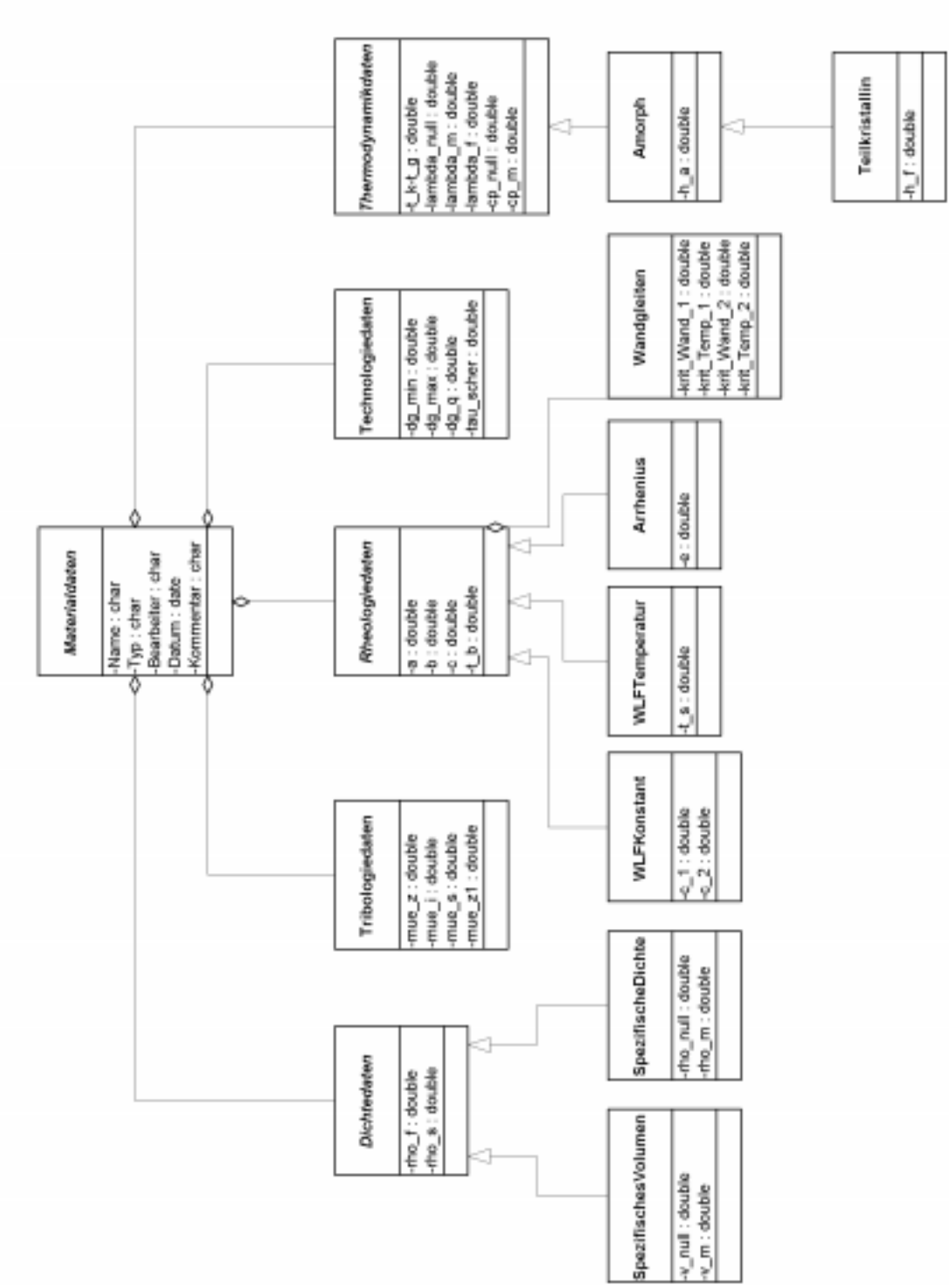


Abbildung 23 Modellierung der Materialdaten für REX

5.3 Sichtenintegration

Nachdem im vorherigen Abschnitt die Schemata für die einzelnen Simulationsprogramme erstellt wurden, müssen diese nun mit den in Kapitel 4.2.3 vorgestellten Methoden der Integration zu einem globalen Schema zusammengeführt werden. Wie bereits in den Grundlagen für die Sichtenintegration dargelegt, stehen verschieden Strategien für die Integration zur Verfügung. Wichtige Kriterien für die Auswahl der Integrationsstrategie sind die Anzahl der Integrationsschritte, bzw. die Anzahl der in einem Schritt zu kombinierenden Sichten.

5.3.1 Konsolidierung der Schemata

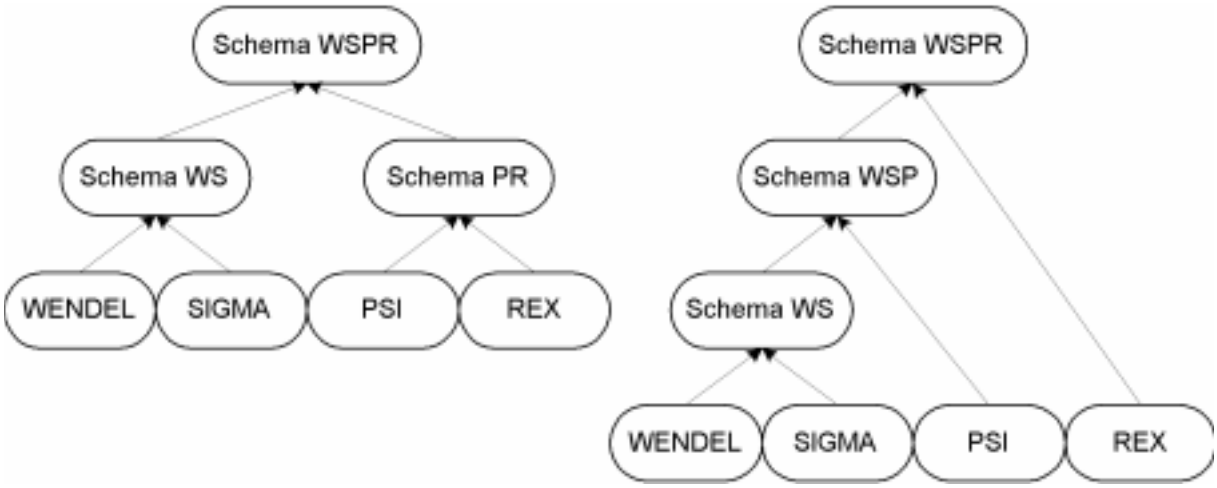
Der Prozeß der Integration wird anhand der in Kapitel 4.2.3.2 beschriebenen Schritte durchgeführt. Der Prozeß besteht aus der Vorintegrationsphase und mehreren Integrationsschritten. In der Vorintegration wird die Strategie und eine Reihenfolge, z.B. durch Priorisierung, festgelegt. Danach folgt in mehreren Einzelschritten die Zusammenführung der Teilschemata.

5.3.1.1 Vorintegrationsphase

Zunächst muß die Strategie für die Schemaintegration festgelegt werden. Die unären Ansätze bieten den Vorteil weniger Schritte, dafür müssen aber oftmals mehrere Schemata in einem Schritt konsolidiert werden. Wählt man eine binäre Strategie, sind zwar nur jeweils zwei Sichten zusammenzuführen, allerdings kann die Anzahl der Schritte in Abhängigkeit von der Anzahl der Sichten stark ansteigen.

Die Modellierung zu Anfang des Kapitels hat vier Schemata ergeben, und somit eine vergleichsweise geringe Zahl. Daher wird im folgenden eine binäre Integrationsstrategie verfolgt. Dadurch schränkt sich die Wahl auf zwei Verfahren ein: die Leiterstrategie oder den balancierten Baum. Die Anzahl der Integrationsschritte ist in diesem Fall bei beiden gleich (Abbildung 24). Betrachtet man die Schemata genauer, fällt auf, daß die Schemata, begünstigt durch die Wahl der Reihenfolge, aufeinander aufbauen. Das bedeutet, das die kleineren Schemata fast vollständig (mit nur geringen Änderungen oder Ergänzungen) im nächst größeren Schema enthalten sind. Dieser sukzessive Aufbau wird aber genau durch die Leiterstrategie widergespiegelt. Daher kommt dieses Verfahren zum Einsatz.

Die Kombination der Schemata wird in Abhängigkeit von der Anzahl der enthaltenen Materialeigenschaften gewählt. Dies ist sinnvoll, da fast alle Materialdaten der kleineren Modelle auch in den umfangreicheren Modellen enthalten sind. Daher können die größeren Modelle als eine Art Erweiterung aufgefaßt werden. Daraus ergibt sich als Reihenfolge für die Integration der Modelle: WENDEL, SIGMA, PSI, REX.



balancierter Baum Leiter-Strategie

Abbildung 24 Vergleich der binären Strategien

5.3.1.2 Integration von WENDEL und SIGMA

Beim Vergleich der beiden Schemata fällt auf, daß alle in WENDEL benötigten Materialdaten auch in SIGMA verwendet werden. Da die Namengebung in beiden Programmen auf der fest definierten Formel- und Zeichensprache der Kunststofftechnik (siehe [Rao89]) beruht, ist die sonst häufig auftretende Problematik der Synonyme bzw. Homonyme nicht gegeben.

Der einzige Unterschied liegt in der Modellierung der Dichtedaten. Im Simulationsprogramm WENDEL werden nur die Parameter rho_null und rho_m als Dichtedaten benötigt und stehen so direkt per Aggregation mit der Klasse *Materialdaten* in Beziehung. Das Schema läßt sich jedoch durch folgende Schritte so erweitern, daß es der Struktur von SIGMA entspricht.

1. Umbenennen der Klasse *Dichtedaten* in *SpezifischeDichte*
2. Einfügen einer leeren, abstrakten Klasse *Dichtedaten* zwischen *Materialdaten* und *SpezifischeDichte*
3. Ändern der Beziehungen: Aggregation zwischen *Materialdaten* und *Dichtedaten*, Spezialisierung zwischen *Dichtedaten* und *SpezifischeDichte*

Nach der Durchführung dieser Änderungen können die beiden Schemata zusammengeführt werden. Das konsolidierte Schema entspricht dann dem Schema von SIGMA (siehe Abbildung 21).

5.3.1.3 Hinzufügen des dritten Schemas: PSI

Untersucht man die beiden Schemata fällt auf, daß PSI sowohl zwei Klassen mehr (*Tribologiedaten* und *Wandgleiten*) enthält, als auch in der Klasse *Technologiedaten* weitere

Attribute. Da das intermediäre Schema aus den Komponentenschemata WENDEL und SIGMA im PSI-Modell enthalten ist, und die Unterschiede lediglich Ergänzungen darstellen, läßt sich das ganze problemlos zu einem neuen Schema, das alle drei Modelle beinhaltet, integrieren. Das Ergebnis entspricht dem in Abbildung 22 gezeigten Schema.

5.3.1.4 Konsolidierung mit dem vierten Schema: REX

Der Vergleich zwischen REX und dem bisherigen intermediären Schema liefert lediglich eine kleine Differenz. In REX enthält die Klasse *Tribologiedaten* zwei weitere Werte. Dies stellt jedoch wie im vorangegangenen Schritt keine strukturelle Änderung des Modells, sondern nur eine weitere Ergänzung dar. Somit ergibt sich als globales Schema, in dem alle vier Komponentenschemata vereint wurden, das gleiche Bild wie für das Komponentenschema REX (Abbildung 23).

5.4 Datenbank-Anbindung

Nachdem im vorangegangenen Kapitel die Komponentenschemata der einzelnen Simulationsprogramme zu einem gemeinsamen Schema konsolidiert wurden, soll in diesem Kapitel die Anbindung an ein relationales Datenbank-Management-System (RDBMS) realisiert werden.

Die Anbindung an das RDBMS wird wie in Abbildung 25 schematisch dargestellt realisiert. Zunächst wird das zuvor entwickelte Schema in VARLET eingegeben und nachbearbeitet. Aus VARLET heraus können dann die nötigen Informationen für die Middleware und die JAVA-Zugriffsschicht generiert werden. Dies sind die relationale Schemadefinition, sowie das objekt-orientierte Schema und die Beschreibung des Mappings.

5.4.1 Schemaentwurf in VARLET

VARLET ist als Reengineering-Werkzeug konzipiert worden. Der Prozeß des Reenginnering beginnt daher mit dem Einparsen oder der manuellen Eingabe eines relationalen Datenbankschemas. Da das zuvor konsolidierte Schema mit objekt-orientierten Methoden modelliert wurde, muß zunächst ein geeignetes relationales Schema entwickelt werden. Grundlage dafür ist die bei der Sichtenintegration durchgeführte Unterteilung nach physikalischen Eigenschaften (Abbildung 18). Die dort aufgezeigten Klassen werden im relationalen Schema als Tabellen modelliert, die Aggregationen werden als Fremdschlüsselbeziehungen umgesetzt. Dazu wird die Tabelle Material um fünf Attribute ergänzt. Dies sind ID_Dichte, ID_Tribologie, ID_Rheologie, ID_Thermodynamik und ID_Technologie. Die gleichen Attribute werden jeweils in die zugehörigen Tabellen als Schlüssel eingefügt.

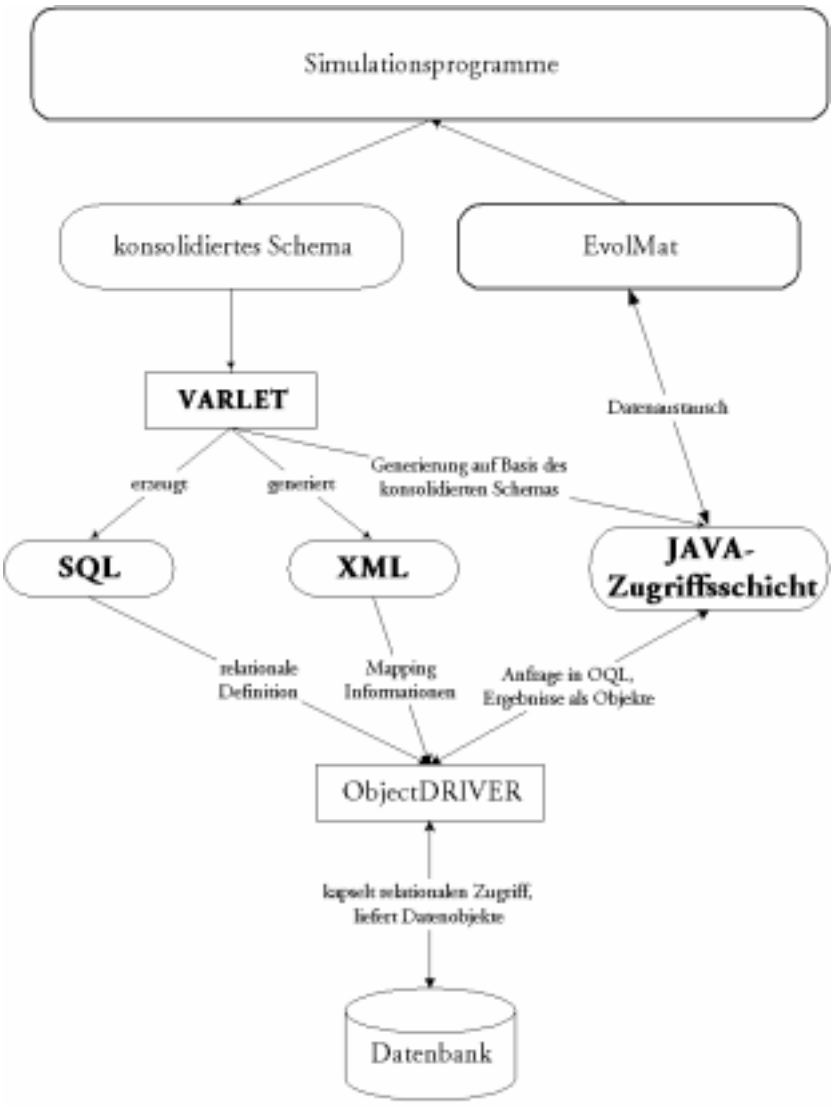


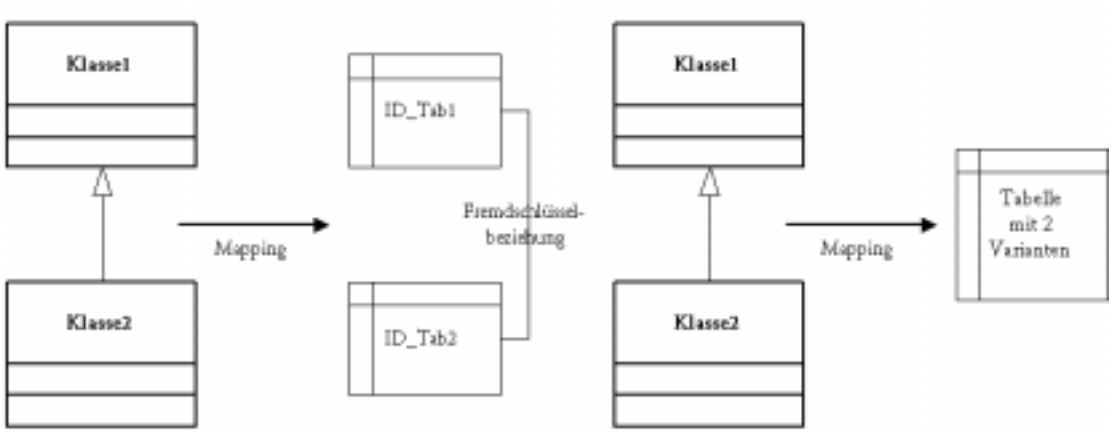
Abbildung 25 Schemazeichnung der Datenbankanbindung

Wird VARLET als Reengineering-Werkzeug eingesetzt, kann es bei der Analyse eines SQL-Schemas Varianten identifizieren. Die Erkennung der Varianten geschieht während der Analyse des Datenbestandes über Nullwerte. Dabei werden die unterschiedlichen Belegungen von Attributen mit Werten bzw. Nullwerten zu Gruppen zusammengefaßt. Jede dieser Gruppen bildet eine Variante. Bei der Migration werden die Varianten auf eine entsprechende Vererbungsstruktur abgebildet.

Dieses Mapping zwischen den Schemata ist allerdings nur in einer Richtung eindeutig, nämlich bei der Migration vom relationalen auf das objekt-orientierte Modell. Die Rückrichtung läßt je nach abzubildendem Objekttyp mehrere Interpretationsmöglichkeiten zu:

Beispiel 1: Mapping einer Vererbung

Eine Vererbungsstruktur zweier Klassen im OO-Modell kann auf zwei Arten im relationalen Modell dargestellt werden: zwei Tabellen mit einer Fremdschlüsselbeziehung (Abbildung 26a) oder eine Tabelle mit zwei Varianten (Abbildung 26b).



a) zwei Tabellen mit Fremdschlüssel	eine Tabelle mit zwei Varianten

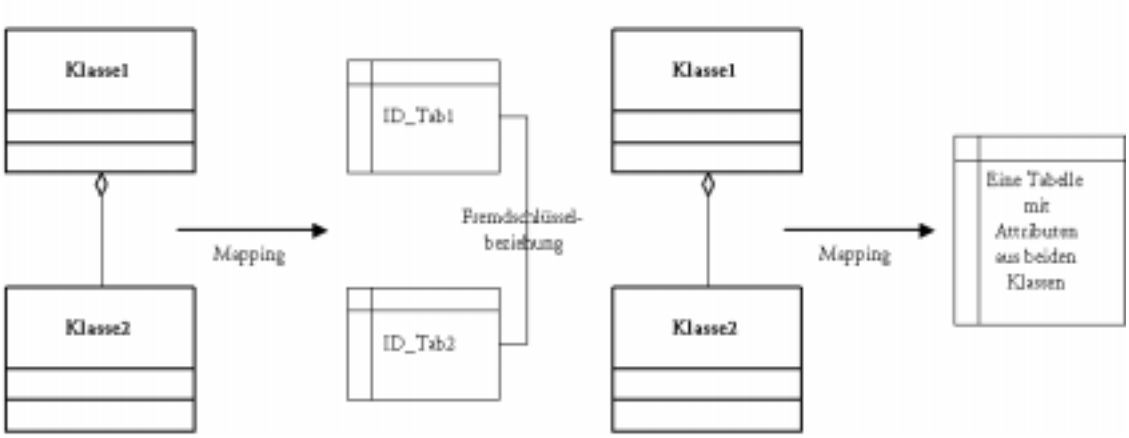
Abbildung 26 Mapping einer Vererbung im relationalen Modell

Im Fall a) würden die gemeinsamen Attribute in beiden Tabellen auftauchen, wodurch die Redundanz erhöht wird. Der Fall b) ergibt sich z.B. bei der Abbildung der Vererbungshierarchie innerhalb der Dichtedaten. Ein ähnliches Problem ergibt sich bei der Abbildung einer Aggregation:

Beispiel 2: Mapping einer Aggregation

Zwei per Aggregation verbundene Klassen lassen sich im relationalen Schema durch zwei Tabellen mit Fremdschlüsselbeziehung beschreiben (Abbildung 27a) oder die

Aggregation wird durch Zusammenführen der Attribute in einer Tabelle (Abbildung 27b) aufgelöst.



a) zwei Tabellen mit Fremdschlüssel	b) eine zusammengeführte Tabelle

Abbildung 27 Mapping einer Aggregation im relationalen Modell

Der erste Fall betrifft alle Beziehungen zwischen der Klasse Material und den fünf Unterklassen. Fall b) spiegelt sich bei der Aggregation zwischen den Klassen Rheologiedaten und Wandgleiten wieder.

Da die Transformation des objekt-orientierten Schemas in ein relationales nicht eindeutig ist, ist hier eine Interaktion des Benutzers erforderlich. Der Anwender muß das OO-Schema derart um Informationen ergänzen können, daß die Abbildungen eindeutig festgelegt sind. Die interaktive Ergänzung des objekt-orientierten Schemas mit mapping-relevanten Informationen ist zur Zeit noch nicht VARLET implementiert. Daher ist eine automatische eindeutige Übersetzung des OO-Schemas, wie es in Kapitel 5.3 entwickelt wurde, in ein zugehöriges relationales Schema mit VARLET noch nicht möglich.

Um dennoch sowohl das relationale als auch das objekt-orientierte Schema in VARLET zur Verfügung zu haben, wird das Mapping vom OO-Schema auf das relationale

Schema *per Hand* durchgeführt. Das daraus resultierende Schema läßt sich dann durch die automatische Migration in VARLET verifizieren.

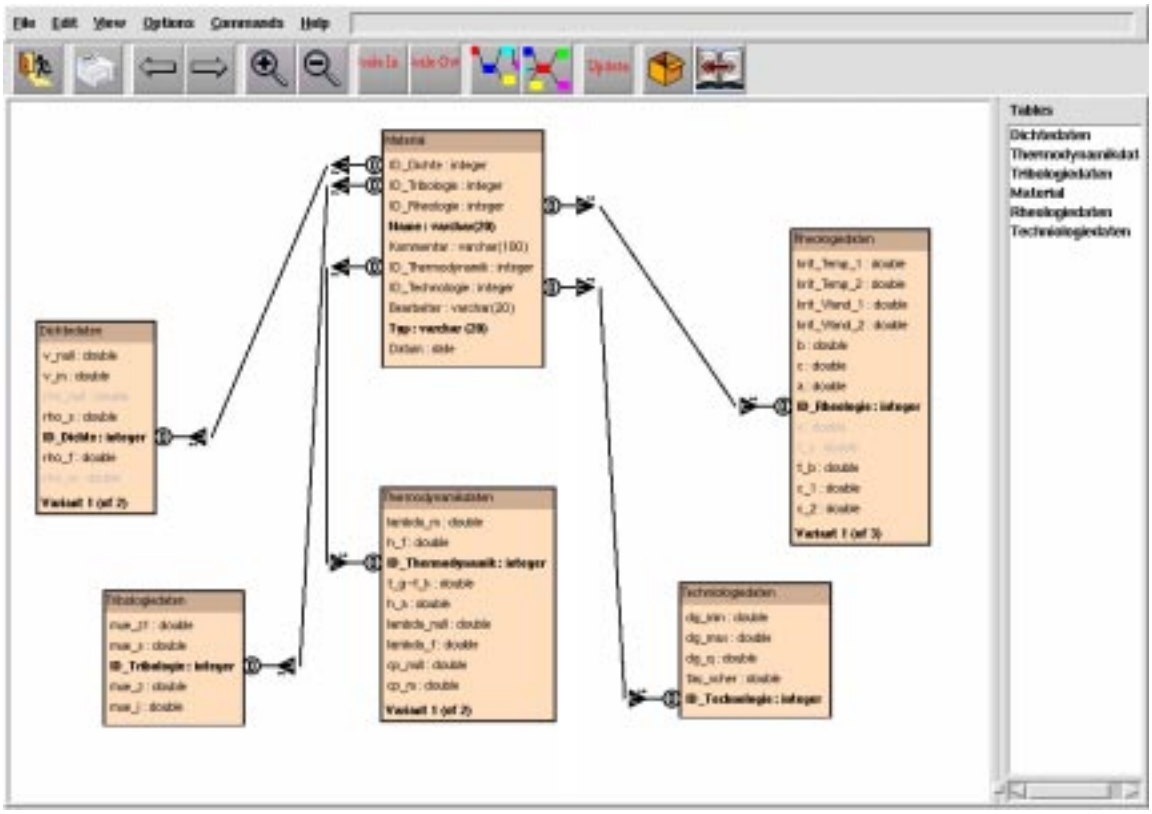


Abbildung 28 SQL-View in VARLET

Unter Berücksichtigung der in den beiden Beispielen aufgezeigten Problemfälle beim Mapping werden sechs Tabellen generiert: Material, Dichtedaten, Technologiedaten, Rheologiedaten, Tribologiedaten und Thermodynamikdaten. Die Tabelle Material enthält alle Attribute der Klasse Material und zusätzlich fünf weitere, eindeutige Attribute, die für die Fremdschlüsselbeziehung zu den anderen Tabellen dienen. Die Klasse Dichtedaten mit ihren Unterklassen wird auf eine Tabelle mit zwei Varianten abgebildet. Ähnliches gilt für die Thermodynamikdaten und die Unterklassen der Rheologiedaten. Die Aggregation zur Klasse Wandgleiten wird aufgelöst, indem die Attribute mit in die Oberklasse aufgenommen werden. Das so entwickelte relationale Schema ist in Abbildung 28 dargestellt.

Aus dem so erzeugten relationalen Schema läßt sich mit Hilfe der in VARLET integrierten TextViews [Hol97] eine Textdatei generieren, die das Schema in SQL-konformer Syntax enthält. Als zusätzliche Information ist die Angabe der Varianten enthalten. Die

SQL-Notation sieht wie folgt aus (auszugsweise für die Tabellen Material und Dichtedaten):

```
CREATE SCHEMA DiplomArbeit

CREATE TABLE Material(
(* Variant 1 of 1 *)
  Name varchar(20),
  Kommentar varchar(100),
  Bearbeiter varchar(20),
  Typ varchar (20),
  Datum date,
  ID_Thermodynamik integer,
  ID_Technologie integer,
  ID_Dichte integer,
  ID_Tribologie integer,
  ID_Rheologie integer,
  primary key ( Name, Typ ))

CREATE TABLE Dichtedaten(
(* Variant 1 of 2 *)
  rho_s double,
  rho_f double,
  v_null double,
  v_m double,
  rho_null double,
  rho_m double
  ID_Dichte integer,
  primary key ( ID_Dichte ))

ALTER TABLE Material ADD (
  FOREIGN KEY ( ID_Dichte ) REFERENCES Dichtedaten ( ID_Dichte )
)
```

Nach der Eingabe des relationalen Schemas ist es nun möglich, sich mit den Migrationsmechanismen von VARLET das objekt-orientiertes Schema generieren zu lassen und somit das relationale Modell zu überprüfen. Aus der Analyse des Schemas erzeugt VARLET das in Abbildung 29 gezeigte OO-Schema.

Dabei werden Attribute, die in allen Varianten einer Tabelle vorhanden sind, in eine abstrakte Superklasse eingefügt. Für die einzelnen Varianten werden dann jeweils Subklassen mit den zugehörigen Attributen eingefügt. Die Fremdschlüsselbeziehungen zwischen der Tabelle Material und den anderen fünf Klassen sind in Assoziationen übersetzt worden. Das so generierte Schema entspricht aber scheinbar noch nicht dem konsolidierten Schema aus dem vorherigen Kapitel. Dieser Eindruck täuscht jedoch, da sich durch Anwendung weniger Äquivalenztransformationen das gesuchte Schema herleiten läßt. Die verwendeten Transformationen sind: *GeneralizeAbstractMovePropertiesInHierarchy*, *Rename*, *SplitClass* und *Aggregate*.

Da die Generierung der Super- und Subklassen automatisch geschieht, ist die Namensgebung der Subklassen nicht besonders gut für die Lesbarkeit. Hier müssen also die neu generierten Unterklassen umbenannt werden (z.B. Dichtedaten#1 in SpezifischeDichte).

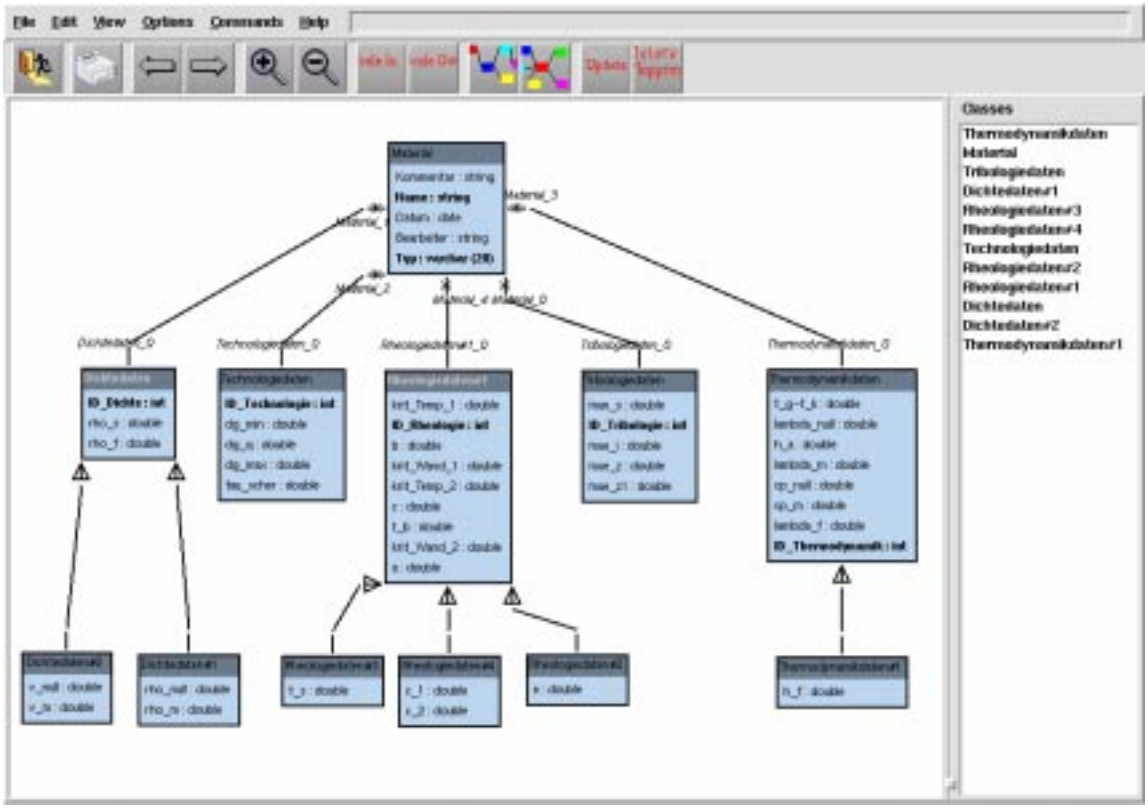


Abbildung 29 Initial migriertes Objekt-Schema in VARLET

Die Klasse Rheologiedaten enthält noch die Daten für die Wandschubspannung (krit_Wand_1, krit_Wand_2, krit_Temp_1 und krit_Temp_2). Diese sollen aber, da sie nicht für alle Rechnungen benötigt werden, als eigene Klasse aggregiert werden. Deshalb wird die Klasse gesplittet und die vier Attribute in die neue Klasse Wandgleiten verschoben

Desweiteren ist die Klasse Thermodynamikdaten nicht als abstrakte Klasse modelliert und enthält noch das Datum h_a. Um hier das Schema entsprechend den Vorgaben anzupassen, bietet VARLET die Funktion *GeneralizeAbstractMovePropertiesInHierarchy*. Dazu werden die entsprechenden Attribute (alle bis auf h_a) ausgewählt, und dann in die neu eingefügte, abstrakte Oberklasse eingefügt. Jetzt müssen nur noch die zugehörigen Klassen richtig benannt werden. Zuletzt werde noch die fünf Assoziationen der Klasse Materialdaten in Aggregationen umgewandelt und mit den entsprechenden Kardinalitäten (alle 1:n) versehen. Nach Durchführung der genannten Operationen erhält man das in Abbildung 30 gezeigte Schema. Das Schema entspricht nun genau dem objekt-orientierten Modell. Da alle Änderungsoperation am OO-Schema Äquivalenztransformationen sind, besteht immer noch eine eindeutige Abbildung vom relationalen auf das objekt-orientierte Schema und da dieses genau den Vorgaben entspricht, kann auch das relationale Modell als korrekt betrachtet werden.

Ähnlich wie beim SQL-View lassen sich auch die im objekt-orientierten Schema enthaltenen Informationen als Textdatei exportieren. Dabei wird standardmäßig eine Syntax verwendet, die konform zum ODMG-Standard [ODMG97] ist. Das Schema ist im Anhang C: Mapping-Schema für ObjectDRIVER aufgeführt.

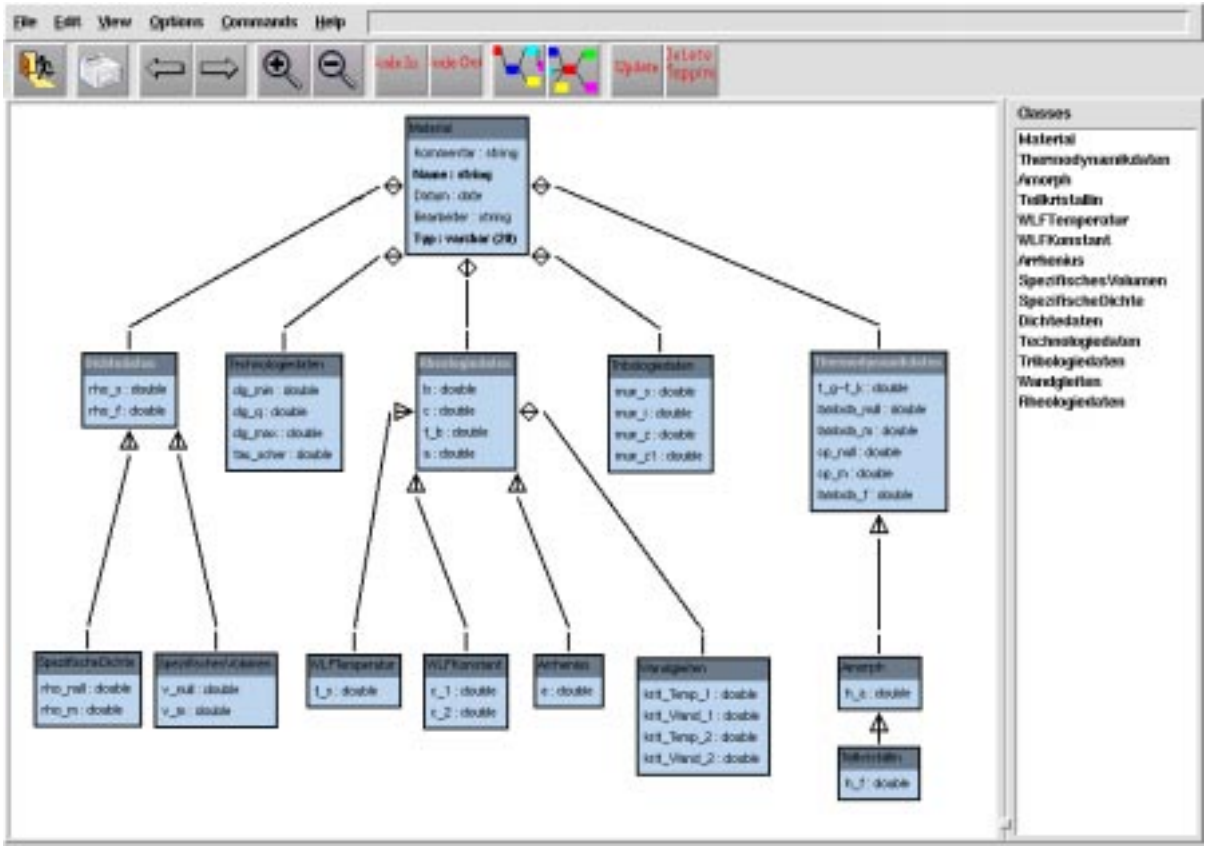


Abbildung 30 Nachbearbeitetes OO-Schema in VARLET

Zusätzlich wurde ein weiterer TextView implementiert, der den Export des OO-Schemas als XML-Dokument erlaubt.

5.4.2 Der XMI-TextView

Die Implementation des XMI-TextViews orientiert sich an den Vorgaben, die in der Arbeit von J. Holle [Hol97] gemacht wurden. Zur Implementierung wurde die Skriptsprache Tcl/Tk verwendet, um eine nahtlose Integration in das bestehende System zu ermöglichen.

Ein TextView ermöglicht die textuelle Repräsentation des aktuellen Schemas. Die Informationen für jedes Inkrement[1] werden dazu mittels eines Unparsers aus der internen abstrakten Datenstruktur gewonnen. Die Inkremente enthalten zur Darstellung das sogenannte *Presentation-Attribut*. Durch Redefinition dieses Attributes kann für jeden

1. Als Inkrement werden hier die Daten bezeichnet, die in PROGRES in einem Knoten abgelegt sind.

Inkrementtyp die Unparsing-Information angegeben werden. Das folgende Code-Fragment zeigt ein solches Unparsing-Schema. Darin wird im SQL-View der *CREATE TABLE* Block erzeugt, im OO-Schema das *interface class* und das Äquivalent in der XMI-Notation. Da in Progres nur relativ kurze Zeichenketten verarbeitet werden können, ist die sonst in Tcl/Tk durchgeführte Zeichenkettenersetzung aus Gründen der Lesbarkeit bereits eingearbeitet.

```
node_class DICompoBox is_a DIComposite
  redef_derived
  Presentation = [ ((self.Kind) = "Relation") ::
                   "CREATE TABLE " & self.DIText & "(" & ShowVariants ( self )
                   & ListFirst ( self ) & ShowPrimaryKeys ( self ) & ")"
                   | ((self.Kind) = "Column") ::
                   [ TestNext ( self ) ::
                     ListFirst ( self ) | ListFirst ( self )
                   ]
                 | ListFirst ( self )
                 ]
  OOPresentation = [ ((self.Kind) = "Class") ::
                     "interface " & self.DIText & GetInherit ( self ) & "{"
                     & ListFirst ( self )
                     & GetEdges ( self )
                     & "};"
                     | ((self.Kind) = "Attr") ::
                        ListAttr ( self ) & ";" | ListFirst ( self )
                     ]
  XMIView = [ ((self.Kind) = "Class") ::
  [use diroot : DIRoot := (self.<-item-.( instance_of LITEM ).<-lc-) : DIRoot [0:1] ::
    "<Foundation.Core.Namespace.ownedElement>
       <Foundation.Core.Class xmi.id=" & diroot.ViewName & "." & self.DIText & ">
         <Foundation.Core.ModelElement.visibility xmi.value=public/>
         <Foundation.Core.GeneralizableElement.isRoot xmi.value=true/>
         <Foundation.Core.GeneralizableElement.isLeaf xmi.value=false/>
         <Foundation.Core.GeneralizableElement.isAbstract xmi.value=false/>
         <Foundation.Core.Class.isActive xmi.value=false/>"
           & ListFirst ( self ) &
      "</Foundation.Core.Class>
     </Foundation.Core.Namespace.ownedElement>"
  end
  | ""
  ]
  | ListFirst ( self )
  ]
  end;
```

Abbildung 31 Progres-Code Ausschnitt für die TextViews

5.4.3 Anbindung des ObjectDRIVER

Als Schnittstelle zwischen dem RDBMS und dem Informationssystem EvolMat dient die Middleware ObjectDRIVER. Die benötigten Informationen über das Datenbank-

Schema sowie die Abbildung der Klassen und ihrer Attribute und Beziehungen auf das DB-Schema liest die Middleware aus zwei Textdateien ein. Die Textdateien können aus den TextViews von VARLET generiert werden.

5.4.3.1 Das relationale Schema

Die Eingabedatei für das relationale DB-Schema verwendet nahezu die gleiche Syntax wie sie in SQL gebräuchlich ist. Zu den Änderungen zählen unterschiedliche Schlüsselwörter (define statt create), andere Datentypen (z.B. string statt varchar) und Informationen über die verwendete Datenbank. Alle Änderungen lassen sich aber automatisch mittels eines Parsers durchführen, so daß die Informationen über das relationale Schema für ObjectDRIVER automatisch aus dem SQL-View von VARLET erzeugt werden können. Als Beispiel sind hier wieder wie auch zuvor beim SQL-View die beiden Tabellen Material und Dichtedaten aufgeführt:

```
define schema Materialdaten
{
   relationalDbms MsAccess;

   define table MATERIAL
      {
      Name        string(20),
      Kommentar   string(100),
      Bearbeiter  string(20),
      Typ         string (20),
      Datum       date,
      ID_Thermodynamikinteger,
      ID_Technologieinteger,
      ID_Dichte   integer,
      ID_Tribologieinteger,
      ID_Rheologieinteger,
      primary key ( Name, Typ )
   };
};
```

```
define table DICHTEDATEN
{
   rho_s       double,
   rho_f       double,
   v_null      double,
   v_m         double,
   rho_null    double,
   rho_m       double
   ID_Dichte   integer,
   primary key ( ID_Dichte )
};
```

Abbildung 32 Definition des relationalen Schemas für ObjectDRIVER

5.4.3.2 Das objekt-orientierte Mapping

Bei der Definition des objekt-orientierten Schemas werden im ObjectDRIVER zwei Informationsblöcke miteinander vermischt, d.h. in die Schemainformationen werden gleichzeitig die Mapping-Informationen mit eingebunden. Auch hier hält sich die Definition des Schemas nicht an den Standard (ODMG), da sowohl die Schlüsselwörter als auch die Syntax ein wenig anders aufgebaut ist. Die im ODMG-Schema verwendeten Schlüsselwörter wie *interface, attribute* oder *relationship* werden von ObjectDRIVER nicht verwendet. Vielmehr erinnert dessen Syntax an die Klassendefinition aus Programmiersprachen wie C++ oder Java. Dies ist insofern naheliegend, da für diese beiden Sprachen eine entsprechende API existiert.

Die Syntax einer Klasse ist wie folgt aufgebaut:

```
class Klassenname on Tabelle
bzw. class Klassenname inherit Superklasse
  type Tuple
  (
    foo      Typ on Tabelle.Attribut
  )
  join ...
end;
```

Klassenname:	Der Name der Klasse im OO-Schema
Tabelle:	Tabelle im relationalen Schema auf die die Klasse abgebildet wird
Tabelle.Attribut:	Zuordnung der Klassenattribute zu Tabellenattributen
join...:	Wird auf mehrere Tabellen zugegriffen, werden hier die nötigen Verknüpfungen (joins) angegeben.

inherit SuperklasseLiegt eine Vererbung vor, wird hiermit die Oberklasse angegeben.

Im folgenden ist die Abbildung der beiden Klassen Materialdaten und Dichtedaten auf die entsprechenden Tabellen dargestellt. Da beide Klassen nur auf jeweils eine Tabelle abgebildet werden, sind keine join-Operationen nötig.

```
class Materialdaten on MATERIAL
   type Tuple
   (
      name       String on MATERIAL.NAME
      kommentar  String on MATERIAL.KOMMENTAR,
      bearbeiter String on MATERIAL.BEARBEITER,
      typ        String on MATERIAL.TYP,
      datum      Date on MATERIAL.DATUM
      dichte     Dichtedaten on MATERIAL.ID_DICHTE
                 inverse Dichtedaten.materials
   )
end;

class Dichtedaten on DICHTEDATEN
   type Tuple
   (
      rho_s      Double on DICHTEDATEN.RHO_S,
      rho_f      Double on DICHTEDATEN.RHO_F,
      materials  Set on M<MATERIAL>
                 (
                    aMaterial Materialdaten on M.MATERIAL,
                    join DICHTEDATEN to M by (DICHTEDATEN.ID_DICHTE == M.ID_DICHTE)
                 )
```

```
class SpezifischeDichte inherit Dichtedaten
   type Tuple
   (
      rho_null   Double on DICHTEDATEN.RHO_NULL,
      rho_m      Double on DICHTEDATEN.RHO_M
   )
end;

class SpezifischesVolumen inherit Dichtedaten
   type Tuple
   (
      v_null     Double on DICHTEDATEN.V_NULL,
      v_m        Double on DICHTEDATEN.V_M,
   )
end;
```

Abbildung 33 Definition des objekt-orientierten Mappings für ObjectDRIVER

```
                inverse Materialdaten.dichte
    )
end;
```

Abbildung 33 Definition des objekt-orientierten Mappings für ObjectDRIVER

Über die Attribute `dichte` in der Klasse `Materialdaten` und `materials` in der Klasse `Dichtedaten`, wird die Aggregation zwischen beiden Klassen definiert. Die Aggregation hat die Kardinalität 1:n, d.h. zu jedem Satz von Materialdaten gibt es genau eine Gruppe von Dichtedaten, die Dichtedaten können aber in beliebig vielen Materialien vorkommen. Diese Definition der Beziehung zwischen Klassen, benutzt ObjectDRIVER um die Daten derjenigen Klassen aus der Datenbank zu lesen, die mit der aktuellen Klasse verknüpft sind.

5.4.3.3 Die Java-Zugriffsschicht

Um Objekte der Java-Schicht in der Datenbank abzulegen, oder die Daten aus der Datenbank auszulesen, müssen sie Instanzen *persistenz-fähiger* Klassen sein. Unter persistenten Klassen versteht man alle Klassen, die für die Abbildung des relationalen Datenmodells auf ein objekt-orientiertes Modell notwendig sind. Der Programmierer muß eine Klasse als persistenz-fähig kennzeichnen.

Generell gilt in ObjectDRIVER eine Klasse als persistenz-fähig, wenn sie den folgenden Anforderungen genügt:

- Sie ist abgeleitet von der Klasse `ObjectDRIVERObject`
- und sie ist in der objekt-orientierten Schemadefinition enthalten sein.

Persistenz-Paradigma:

Das in ObjectDRIVER verwendete Paradigma ist *Persistenz durch Erreichbarkeit* [LDAJ99]. Das bedeutet, daß ein transientes Objekt, das von einem persistenten Objekt referenziert wird, automatisch persistent wird, falls:

- das transiente Objekt Instanz einer persistenz-fähigen Klasse ist
- und die beiden Objekte nicht durch ein transientes Feld in Beziehung stehen.

Wird ein Objekt aus der Datenbank ausgelesen, werden nicht automatisch alle damit in Beziehung stehenden Objekte mitgelesen, da dies im worst case bedeuten könnte, die gesmate Objekt-Struktur der Datenbank einzulesen, was wiederum zu Speicherproblemen führen kann.

Stattdessen werden die referenzierten Objekte in der Client-Applikation als *proxies*[1] [LDAJ99] dargestellt. Wird auf ein proxy zugegriffen, wird das zugehörige Objekt aus der Datenbank gelesen und in den Speicher geladen.

---

1. Proxy bezeichnet hier nicht die vom WWW bekannte Zwischenspeicherung, sondern dient als Platzhalter für noch nicht geladene Objekte.

Um das durch ein proxy referenzierte Objekt nachzuladen, wird die im ObjectDRIVER konsequent verwendete Technik des Nachrichtenaustausches genutzt. Die Nachrichten-Technik ist Teil der Klasse `ObjectDRIVERObject`, und steht damit allen erbenden Klassen zur Verfügung. Die entsprechende Methode lautet `getObject()`. Obwohl eine direkte Manipulation von Datenfeldern möglich ist, ist es ratsam, alle Modifikationen von Objekten über Nachrichten durchzuführen, da so die Konsistenz gewährleistet ist. Die Verwendung von Nachrichten erfordert allerdings den konsequenten Einsatz von Zugriffsfunktionen (`get-` und `set`-Methoden). Diese sollten vor jeglicher Änderung einen `getObject()` Aufruf benutzen, um die Aktualität der Objektdaten zu gewährleisten.

Modifikationen der Objekte werden nicht, wie bei relationalen Anfragen üblich, mit SQL-Statements durchgeführt, sondern mit Hilfe der Anfragesprache OQL[1]. OQL ist eine deklarative Anfragesprache, die auf der Syntax von SQL basiert. Das Basis-Konstrukt ist ebenfalls von der Form: `SELECT ... FROM ... WHERE ....`

Im Gegensatz zu SQL ist OQL objekt-orientiert modelliert worden, d.h. Anfragen in OQL benutzen Objekte, Methoden oder objekt-orientierte Pfadausdrücke. Die Rückgabewerte solcher Anfragen sind dementsprechend keine Relationen, sondern Mengen von Objekten. Die Mengen können sortiert (`List`) oder unsortiert sein. Enthält die unsortierte Menge Duplikate, ist der Rückgabewert ein `Bag`, ansonsten ein `Set`.

Die Klasse ObjectDRIVERObject

Die Klasse `ObjectDRIVERObject` ist die Oberklasse aller Objekte der persistenten Klassen. Darin enthalten ist ein boolsches Flag `loaded`, das angibt, ob das entsprechende Objekt nur als proxy existiert, oder bereits in den Speicher geladen ist. Dementsprechend wird das Flag im Konstruktor, also bei der Erzeugung eines neuen Objektes, auf `true` gesetzt. Weiterhin existieren zwei Methoden zum Zugriff auf den boolschen Wert, `isloaded` und `setloaded`. Mit der ersten kann der Status abgefragt werden, die zweite kann den Wert auf true oder `false` setzen. Daraus ergibt sich folgender Code:

```
public class ObjectDRIVERObject{
   private boolean loaded;

   public ObjectDRIVERObject{
      loaded = true;
   }

   public final boolean isLoaded() {
      return loaded;
   }
```

1. OQL = Object Query Language, eine von der ODMG standardisierte Anfragesprache für OO-DBMS, in ObjectDRIVER wird Version 1.1 verwendet

```
    public final void setLoaded(boolean value) {
       loaded = value;
    }

    public persistentObject getObject() {
       if (!loaded)
         ObjectDriverAPI.loadObject(this);
       return this;
    }
 }
```

Die Methode `getObject` implementiert den Sicherheitsmechanismus für die proxies. Bevor auf ein Objekt zugegriffen wird, wird überprüft ob es bereits im Speicher ist, falls nicht, wird es aus der Datenbank nachgeladen.

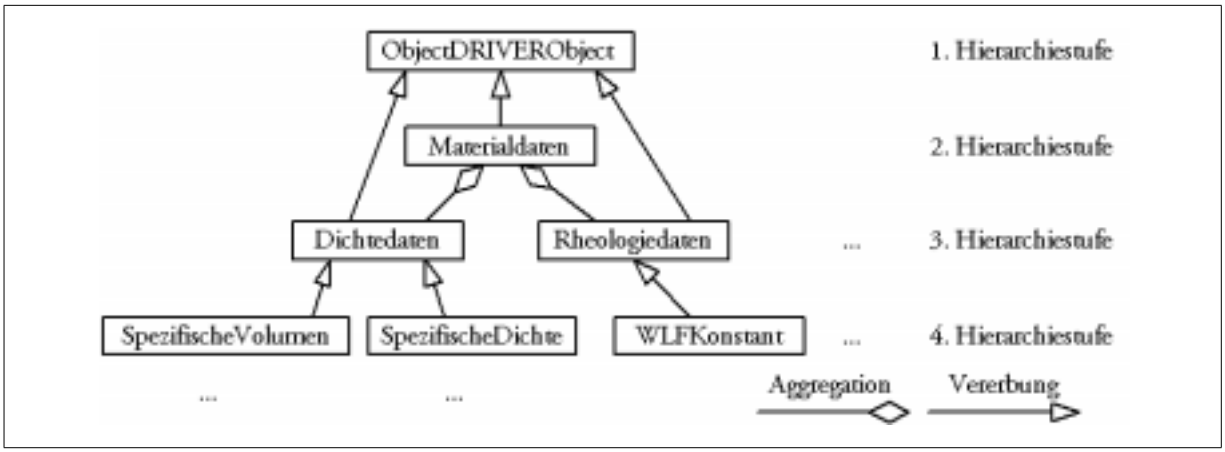


Abbildung 34 Ausschnitt der Vererbungshierarchie in EvolMat

Da die Klasse `ObjectDRIVERObject` Oberklasse aller peristenten Klassen ist, folgen jetzt die direkten Nachfolgeklassen. Da innerhalb des OO-Schemas zwischen der zweiten und dritten Hierarchieebene keine Vererbung existiert, müssen diese Klassen ebenfalls von der Klasse `ObjectDRIVERObject` abgeleitet werden. Die Klassenhierarchie für das Informationssystem EvolMat ist in Abbildung 34 dargestellt.

Die Anwendungsklassen

In diesem Abschnitt werden die Anwendungsklassen beschrieben, die dem Mapping aus Abbildung 33 genügen. Für jede Klasse des OO-Schemas wird eine Java-Klasse für die Zugriffsschicht generiert. Diese sind von der Klasse `ObjectDRIVERObject` abgleitet (direkt oder indirekt) und im OO-Schema enthalten und somit persistenz-fähig. Die Klasse `Materialdaten` ist die oberste Klasse im OO-Schema des Informationssystems, deshalb wird mir ihr angefangen. Sie erbt direkt von der Klasse `ObjectDRIVERObject`, im Klassenkopf durch das Schlüsselwort `extends` angegeben:

```
public class Materialdaten extends ObjectDRIVERObject
```

Die weiteren Klassenköpfe sehen entsprechend aus d.h. sie erben entweder direkt von der Klasse `ObjectDRIVERObject` oder eine ihrer Oberklassen tut dies, wie z.B. bei den Dichtedaten:

```
public class Dichtedaten extends ObjectDRIVERObject
public class SpezifischeDichte extends Dichtedaten
```

Konstruktoren

Jede Klasse, die direkt von `ObjectDRIVERObject` erbt, besitzt zwei Konstruktoren. Der Default-Konstruktor:

```
public Materialdaten()
{}
```

wird für die Vererbung benötigt. Wird ein Objekt der Unterklasse erzeugt, wird der Default-Konstruktor automatisch aufgerufen. Mit dem zweiten Konstruktor:

```
public Materialdaten(String _name, String _kommentar,
             String _bearbeiter, String _typ, Date _datum) {
  name=aName;
  kommentar=_kommentar;
  bearbeiter=_bearbeiter;
  typ=_typ;
  datum=_datum;
}
```

wird ein Objekt der Klasse angelegt und mit den übergebenen Werten belegt.

Die `get`- und `set`-Methoden

Mit Hilfe der `get`-Methode kann von außen auf die privaten Attribute einer Klasse zugegriffen werden. Dabei muß an dieser Stelle durch den Aufruf von `getObject()` gewährleistet werden, daß das Objekt auch wirklich existiert. Die Laderoutine für ein Objekt ist in der Klasse `ObjectDRIVERAPI` implementiert, die auch die Fehlerbehandlung (Exceptions) kontrolliert. Somit ist sichergestellt, daß die Methode nur gültige Rückgabewerte liefert. Der lesende Zugriff auf das Attribut `name` der Klasse `Materialdaten` wird wie folgt realisiert:

```
public final String getName() {
  getObject();
  return name;
}
```

Mit den `set`-Methoden erfolgt die Wertzuweisung an die Attribute einer Klasse. Da Änderungen von außen nicht erlaubt sind, sondern Modifikationen durch Nachrichten vorgenommen werden sollen, sind alle `set`-Methoden als `private` deklariert. Der Übergabeparameter enthält den neuen Wert. Entsprechend sieht die `setName`-Methode aus:

```
private void setName(String _name) {
  name = _name;
}
```

5.4.4 Generierung der Klassen für die Zugriffsschicht

In diesem Abschnitt wird die Generierung der persistenten Klassen der Zugriffsschicht beschrieben. Die erzeugten Klassen beruhen auf den Informationen aus der Schemamigration in VARLET. Die zur Erstellung nötigen Informationen können aus dem bereits vorgestellten XMI-TextView (5.4.2) extrahiert werden. Dafür sind Kenntnisse über die Struktur der Ausgabedatei nötig. Danach können, basierend auf den im XML-Dokument gespeicherten Informationen, die Klassen der Zugriffsschicht generiert werden.

5.4.4.1 Aufbau der XMI-Datei

Die aus dem XMI-TextView generierte Datei entspricht den Vorgaben für ein *well-formed* XML-Dokument. Da es auch mit der zugehörigen DTD (uml11i.dtd[1]) konform ist,

wird es als Dokument der Klasse *valid* eingestuft. Damit erfüllt das Dokument die Voraussetzungen, die für den Austausch von Informationen per XML gefordert werden (vgl. Kapitel 4.2.1.5).

Ein XML-Dokument besteht aus zwei Teilen, einer Beschreibung des Dokumentes mit Angabe der zugehörigen DTD-Datei und einer Liste der verwendeten Attribute,

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE XMI SYSTEM "uml11i.dtd" [
  <!ATTLIST ixtd
    xmi.id ID #IMPLIED
    xmi.label CDATA #IMPLIED
    xmi.uuid CDATA #IMPLIED
    xml:link CDATA #IMPLIED
    inline (true|false) #IMPLIED
    actuate (show|user) #IMPLIED
    href CDATA #IMPLIED
    role CDATA #IMPLIED
    title CDATA #IMPLIED
    show (embed|replace|new) #IMPLIED
    behavior CDATA #IMPLIED
    xmi.idref IDREF #IMPLIED
    xmi.uuidref CDATA #IMPLIED
    ixtda CDATA #IMPLIED>
]>
```

sowie dem Hauptelement für die Daten und Metadaten.

```
<XMI xmi.version="1.0" timestamp="Mon May 22 PDT 2000"></XMI>
```

Das Element <XMI> enthält die komplette Baumstruktur des Dokumentes. Es läßt sich in zwei weitere Elemente unterteilen, `<XMI.header>` und `<XMI.content>`. Das Header-Element enthält dabei lediglich beschreibende Informationen, wie das verwendete Werkzeug zur Erzeugung des Dokumentes, eventuell dessen Versionsnummer und das verwendete Metamodel.

```
<XMI.header>
  <XMI.documentation>
    <XMI.shortDescription>Varlet Model</XMI.shortDescription>
    <XMI.exporter>Varlet</XMI.exporter>
    <XMI.exporterVersion>1.0</XMI.exporterVersion>
  </XMI.documentation>
  <XMI.metamodel xmi.name="UML" xmi.version="1.1"/>
</XMI.header>
```

Die relevanten Daten sind in verschiedenen Unterelementen des Elements `<XMI.Content>` zu finden. Da eine komplette Beschreibung aller Elemente den Rahmen der Arbeit sprengen würde, sind im folgenden die wichtigsten Gruppen von Elementen entsprechend ihrer Reihenfolge im XML-Dokument aufgeführt.

---

1. Diese DTD-Datei enthält die komplette Definitionen der Modellierungssprache UML, Version 1.1

```
<Model_Management.Package xmi.id="_1.1">
   <Foundation.Core.ModelElement.name>IDL_DATATYPE</Foundation.Core.ModelElement.name>
   <Foundation.Core.ModelElement.namespace>
      <Model_Management.Model xmi.idref="_1"/>
   </Foundation.Core.ModelElement.namespace>
   <Foundation.Core.Namespace.ownedElement>
      <Foundation.Data_Types.Primitive xmi.id="_1.1.5">
         <Foundation.Core.ModelElement.name>int</Foundation.Core.ModelElement.name>
         <Foundation.Core.ModelElement.namespace>
            <Model_Management.Package xmi.idref="_1.1"/>
         </Foundation.Core.ModelElement.namespace>
      </Foundation.Data_Types.Primitive>
      <Foundation.Data_Types.Primitive xmi.id="_1.1.15">
         <Foundation.Core.ModelElement.name>void</Foundation.Core.ModelElement.name>
         <Foundation.Core.ModelElement.namespace>
            <Model_Management.Package xmi.idref="_1.1"/>
         </Foundation.Core.ModelElement.namespace>
      </Foundation.Data_Types.Primitive>
   </Foundation.Core.Namespace.ownedElement>
</Model_Management.Package xmi.id="_1.1">
```

Abbildung 35 Das XMI-Element <Model_Management.Package>

Das Element <Model_Management.Package> enthält die Abbildung der Datentypen und der Funktionswerte aus der IDL[1] auf die verwendete Programmiersprache. Der Ausschnitt zeigt die Beschreibung der Abbildung des Variablentyps `integer` und des Funktionswertes `void()`. Zusätzlich enthält das Element im Start-Tag eine Identifikation: `xmi.id="_1.1"`. Diese Numerierung wird fortlaufend vergeben, um einzelne XMI-Elemente eindeutig identifizieren zu können. Dies ist nötig, da Elemente durchaus mehrfach vorkommen können, wie z.B. eine Klassenbeschreibung.

Das nächste Element ist <Foundation.Core.Class>. Es wird für jede Klasse des Schemas einmal aufgeführt. Dieses Element enthält alle wichtigen Daten über die im Schema vorhandenen Klassen, ihre Attribute und die Beziehungen. Alle diese Informationen sind durch weitere Unterelemente strukturiert, deren Bedeutung im folgenden erklärt wird.

`<Foundation.Core.Class xmi.id="_1.2" >`	Start-Tag für die Klassenbeschreibung
`<Foundation.Core.ModelElement.name>`	enthält den Namen der Klasse
`<Foundation.Core.Classifier.associationEnd>`	gibt die Klasse an, in der die Beziehung endet
`<Foundation.Core.Classifier.feature>`	innerhalb dieses Elements werden alle Attribute aufgelistet
`<Foundation.Core.Attribute>`	enthält die komplette Beschreibung des Attributes mit Name, Sichtbarkeit, Besitzer u.a.

1. IDL = Interface Data Language, standardisierte Beschreibungssprache zur Schnittstellendefinition

`<Foundation.Core.Association>`	Start-Tag für die Beschreibung der Beziehung zwischen zwei Klassen
`<Foundation.Core.Association.connection>`	beschreibt die Art der Beziehung zwischen den Klassen, also ob es sich z.B. um eine einfache Assoziation oder eine Aggregation handelt und die Kardinalität
`<Foundation.Core.Generalization>`	zusätzliche Beschreibung der Beziehung, u.a. mit Angabe der Ober- und Unterklasse

Daraus ergibt sich für die Beschreibung einer Klasse die im folgenden Ausschnitt gezeigte XMI-Struktur. Das Beispiel zeigt die Klasse Dichtedaten aus dem Informationssystem.

```
<Foundation.Core.Class xmi.id="_1.3">
   <Foundation.Core.ModelElement.name>Dichtedaten</Foundation.Core.ModelElement.name>
   <Foundation.Core.ModelElement.namespace>
      <Model_Management.Model xmi.idref="_1"/>
   </Foundation.Core.ModelElement.namespace>
   <Foundation.Core.Classifier.associationEnd>
      <Foundation.Core.AssociationEnd xmi.idref="_1.15.2"/>
   </Foundation.Core.Classifier.associationEnd>
   <Foundation.Core.Classifier.feature>
      <Foundation.Core.Attribute xmi.id="_1.3.1">
         <Foundation.Core.ModelElement.name>rho_f</Foundation.Core.ModelElement.name>
         <Foundation.Core.ModelElement.visibility xmi.value="private"/>
         <Foundation.Core.Feature.ownerScope xmi.value="instance"/>
         <Foundation.Core.Feature.owner>
         <Foundation.Core.Class xmi.idref="_1.3"/>
         </Foundation.Core.Feature.owner>
         <Foundation.Core.StructuralFeature.type>
         <Foundation.Data_Types.Primitive xmi.idref="_1.5"/>
         </Foundation.Core.StructuralFeature.type>
      </Foundation.Core.Attribute>
      <Foundation.Core.Attribute xmi.id="_1.3.2">
         <Foundation.Core.ModelElement.name>rho_s</Foundation.Core.ModelElement.name>
         <Foundation.Core.ModelElement.visibility xmi.value="private"/>
         <Foundation.Core.Feature.ownerScope xmi.value="instance"/>
         <Foundation.Core.Feature.owner>
         <Foundation.Core.Class xmi.idref="_1.3"/>
         </Foundation.Core.Feature.owner>
         <Foundation.Core.StructuralFeature.type>
         <Foundation.Data_Types.Primitive xmi.idref="_1.5"/>
         </Foundation.Core.StructuralFeature.type>
      </Foundation.Core.Attribute>
   </Foundation.Core.Classifier.feature>
</Foundation.Core.Class>
```

Abbildung 36 XMI-Fragment für die Klasse Dichtedaten

Generierung

Für die Zugriffsschicht müssen entsprechend dem objekt-orientierten Schema die Klassen generiert werden. Für jede im OO-Schema definierte Klasse wird eine Java-Klasse erzeugt.

Das Einlesen der Eingabedatei und das Schreiben der Java-Klassen geschieht über die in Java implementierten Input- und Output-Streams. Da im XML-Dokument sämtliche Klassen und Assoziationen des Schemas abgelegt sind, braucht diese Datei nur einmal geparst zu werden. Wird das Dateiende, oder genauer das Ende-Tag `</XMI>` gelesen, ist die Einleseroutine fertig. Die eingelesenen Daten lassen sich in logische Blöcke, gekennzeichnet durch XMI-Tags, gliedern.

Der erste große Block mit relevanten Daten ist das Element `<Model_Management.Package>`. Hier werden die verwendeten Datentypen festgelegt und mit eindeutigen ID´s versehen (`xmi.id`). Über diese ID´s werden später die Datentypen für die einzelnen Attribute referenziert. Der nächste Block, die Klassendefinition, ist eingebettet in das XMI-Element `<Foundation.Core.Class>`. Die Attribute finden sich dementsprechend in dem XMI-Element `<Foundation.Core.Attribute> wieder.`

Das Schreiben der Klassen erfolgt in mehrere Teildateien, die im Anschluß konkateniert werden. Dies ist notwendig, da die Reihenfolge der XMI-Daten nicht mit der Reihenfolge des zu erzeugenden Programm-Codes übereinstimmt. Die Konkatenation erfolgt, nach dem das Ende-Tag einer Klassendefinition gelesen wurde. Zu diesem Zeitpunkt sind alle die Klasse betreffenden Informationen gelesen worden. Dies wird durch die strengen Syntax-Regeln für ein XML-Dokument der Korrektheitsstufe *valid* gewährleistet.

Für jeden Klassenblock, der im XML-Dokument gefunden wird, werden die nachfolgenden Schritte zur Code-Generierung durchlaufen. Die erste Teildatei, die geschrieben wird (`static.txt`) enthält den statischen Programmcode. Das ist Code, der unabhängig von den einzelnen Klassen ist.

Danach wird der Klassenname aus dem entsprechenden XMI-Tag (`<Foundation.Core.ModelElement.name>`) ausgelesen und in der Variablen classname gespeichert. Jetzt werden die zweite und dritte Teildatei angelegt. Die zweite Datei (`classname`[1]`.txt`) enthält den Klassenkopf sowie die Attributdeklarationen. In die dritte Datei (`classname.methods.txt`) werden die Konstruktoren, die Zugriffsmethoden und der Klassenabschluß geschrieben. Die Information, ob und von welcher Klasse geerbt wird, wird in dem Element `<Foundation.Core.AssociationEnd>` festgelegt. Zusätzlich kann die oberste Klasse der Vererbungshierarchie durch das Element `<Foundation.Core.GeneralizableElement.isRoot xmi.value="true"/>` gekennzeichnet werden. Diese und alle anderen Klassen, die nicht in der Vererbungs-

1. classname wird dabei durch den jeweiligen Klassennamen ersetzt.

hierarchie der Root-Klasse enthalten sind, müssen von der Klasse ObjectDriverAPI erben.

Nachdem der Klassenname bekannt ist, werden nun die Attribute ausgelesen. Diese sind in dem XMI-Tag `<Foundation.Core.Attribute>` dargestellt. Zu jedem Attribut werden Name, Sichtbarkeit und Gültigkeitsbereich gespeichert. Aus den Attributen werden mehrere Code-Fragmente generiert. Die Variablen in der Klasse classname, die Parameter und Zuweisungen für den zweiten Konstruktor, sowie die get- und set-Methoden. Basierend auf dem Beispiel in Abbildung 36 sehen die beiden Teildateien bisher so aus:

```
public class Dichtedaten extends ObjectDriverObject {
   private Double rho_f;
   private Double rho_s;
```

Abbildung 37 Inhalt der Datei Dichtedaten.txt

```
   public Dichtedaten() {
      super();
      inspected=false;
   }

   public Dichtedaten(Double _rho_f, Double _rho_s) {
      inspected=false;
      rho_f = _rho_f;
      rho_s = _rho_s;
   }

   public final Double getrho_f() {
      getObject();
      return rho_f;
   }

   private final void setrho_f(Double _rho_f) {
      getObject();
      rho_f = _rho_f;
   }

   public final Double getrho_s() {
      getObject();
      return rho_s;
   }

   private final void setrho_s(Double _rho_s) {
```

Abbildung 38 Inhalt der Datei Dichtedaten.methods.txt

```
        getObject();
        rho_s = _rho_s;
    }
```

Abbildung 38 Inhalt der Datei Dichtedaten.methods.txt

Auf diese Weise werden alle Attribute einer Klasse erfaßt und ihre Zugriffsmethoden generiert. Wird das Ende der Attributliste (`</Foundation.Core.Classifier.feature>`) im XML-Dokument erreicht, wird die noch fehlende schließende Klammer der Klasse an das Ende der Datei `classname.methods.txt` angefügt.

Als letztes müssen die drei Dateien zusammengefügt werden. Die Konkatenation wird in der Reihenfolge, `static.txt` + `classname.txt` + `classname.methods.txt` durchgeführt. Als Zieldatei wird die Datei `classname.java` erzeugt.

5.5 Filtergenerierung in EvolMat

Die zu Anfang der Arbeit vorgestellten Programme benötigen für die Simulationsberechnungen unterschiedliche Teile der Materialdaten. Daher soll hier eine Möglichkeit geschaffen werden, den für jedes Programm spezifischen Datensatz aus der vollständigen Beschreibung eines Materials, wie es in der Datenbank abgelegt ist, zu extrahieren. Diesen Mechanismus der Auswahl einer bestimmten Datenmenge aus einer großen Menge wird als *filtern* bezeichnet.

Vergleicht man den Filtermechanismus mit Methoden aus dem Datenbankentwurf, so entspricht dies der Definition von Sichten. Eine Sicht stellt die relevanten Daten dar, und blendet die anderen aus.

Die Informationen darüber, welche Daten eines Materials die jeweiligen Simulationsprogramme benötigen, ist bereits in den in Kapitel 5.2.2 entwickelten Komponentenschemata enthalten. Somit können die Komponentenschemata als Sichten auf das konsolidierte Schema betrachtet werden. Die Daten werden von den Programmen als Textdatei eingelesen, also brauchen die zu erstellenden Filter ihrerseits nur die Textdatei aus den gefilterten Daten zu schreiben.

5.5.1 Aufbau eines Filters

Damit der Filter die Daten aufbereiten kann, muß er diese zuerst aus der Datenbank abfragen. Dafür wird über den ObjectDRIVER eine Verbindung zur relationalen Datenbank hergestellt. Dies geschieht in der Methode `open` der Klasse `Filter`. Zuvor werden noch im Konstruktor die von ObjectDRIVER benötigten Informationen für die Schemadefinitionen und den Namen der relationalen Datenbank festgelegt.

```
public class Filter{
```

```
    private Database database;
    private String relDatabaseName;
    private String mapSchemaName;
    private String vitualOODatabase;

    public Filter(String _relDB, String _schema, String _virtualOO){
       relDatabaseName = _relDB;
       mapSchemaName = _schema;
       vitualOODatabase = _virtualOO;
    }

    private void open(String _user, String _password, String _nameDB) {
       user = _user;
       password = _password;
       nameDB = _nameDB;
       try {
          database=Database.open(nameDB, Database.openReadWrite);
          ObjectDriverAPI.connectUser(user,password);

          StartFilter();
          database.close();
       }
       catch (java.io.IOException e) {
          System.out.println("IOException caught");
       }
       catch (ODMGException e) {
          System.out.println("ODMGException caught: "+e.getMessage());
       }
       catch (ObjectDriverException e) {
          System.out.println("ObjectDriverException caught: "+e.getMessage());
       }
    }
 }
```

Zuerst wird die Verbindung zur Datenbank mit den im Konstruktor definierten Informationen aufgebaut. Dies übernimmt die Methode `open` der Klasse `Database`. Danach wird über die Methode `connectUser` der Klasse `ObjectDriverAPI` der Benutzer am Datenbanksystem angemeldet. Ist die Verbindung und Anmeldung erfolgreich, kann der Filter gestartet werden.

Als nächstes muß das gesuchte Material, welches der Benutzer im Informationssystem EvolMat ausgewählt hat, aus der Datenbank ausgelesen werden. Dies geschieht mit Hilfe einer OQL-Anfrage, die aus den vorhandenen Informationen zusammengesetzt wird. Eine solche Abfrage wird innerhalb einer Transaktion ausgeführt, so daß zuerst eine neue Transaktion initialisiert werden muß. Das Beenden einer Transaktion erfolgt durch die Methoden `abort()` oder `commit()` der Klasse `Transaction`. Da die Filter nur lesend auf die Datenbank zugreifen, kann die Transaktion mit `abort()` beendet werden.

```
 private void StartFilter() {
    try {
       Transaction transaction=new Transaction();
       Bag values;
       Enumeration elts;
       Materialdaten material;
       String query;
       OQLQuery aQuery=new OQLQuery();
```

```
      query = "select mat from mat in Materialdaten where mat.name=\""
         + matName
         + "\" and mat.typ=\""
         + matTyp
         + "\;"
      transaction.begin();
      aQuery.create(query);
      values=(Bag)aQuery.execute();
      transaction.abort();
   }
}
```

Der Rückgabewert der Anfrage ist vom Typ `bag`, also einer unsortierten Sammlung von Objekten, die möglicherweise auch Duplikate enthält. Allerdings enthält die `where`-Klausel des OQL-Statements als Bedingung genau die beiden Schüsselattribute der Relation Material, so daß genau das gesuchte Material zurückgegeben wird. Die Existenz ist gewährleistet, da die Auswahl des Materials in EvolMat auf einer zuvor an die Datenbank gestellten strukturierten Auswahl-Anfrage besteht:

```
select struct(Name: mat.name, Typ: mat.typ) from mat in Materialdaten
```

Durch die Definition der Aggregationen zwischen der Klasse Material und den Klassen Dichte-, Tribologie-, Thermodynamik-, Technologie- und Rheologiedaten in der Beschreibung für das Mapping (Abbildung 33), ist ObjectDRIVER in der Lage, die Aggregationen automatisch zu traversieren. Diese Definition bildet eine 1:n Beziehung zwischen den beteiligten Klassen ab. Auf der „:n“-Seite, also jeweils in den aggregierten Klassen, wird eine neue Menge `materials` über der Relation M<MATERIAL> definiert. Diese Relation wird gebildet als join der Relationen MATERIAL und der Relation, auf der die aktuelle Klasse basiert.

5.5.2 Die Ausgabedatei

Um die Ausgabe der gelesen Werte zu ermöglichen, werden die im vorherigen Abschnitt beschriebenen Klassen für die Zugriffsschicht erweitert.

5.6 Berücksichtigung von Schemaänderungen

Das im Rahmen der Arbeit entwickelte Informationssystem EvolMat ist kein starres und ausgereiftes System. Vielmehr ist es durch die Einflüsse und Neuerungen aus Forschung und Entwicklung einem steten Wandel unterworfen. Das Informationssystem soll die Verwaltung der Daten für die zu Beginn der Arbeit vorgestellten Simulationsprogramme übernehmen. Diese Programme beinhalten Forschungsergebnisse und Entwicklungen, die sich oft über mehrere Jahre erstrecken. Durch diese Weiterentwicklung der Simulationsprogramme, haben sich auch deren Anforderungen geändert. Durch die Integration neuer Fähigkeiten, wie z.B. neue Modelle von Schneckenelementen, werden zusätzliche

Daten über das Material benötigt. Bei der Verbesserung der Berechnungen durch neue mathematische Modelle werden nicht mehr alle Materialdaten verwendet.

Diese Problematik stellt sich bei allen Simulationsprogrammen. Durch die langjährige Entwicklungszeit in mehreren Projektabschnitten, existieren für jedes Programm ein oder mehrere Formate von Dateien mit Materialdaten. Zwar können die älteren Datensätze meistens noch eingelesen werden, der Export für die älteren Programmversionen wird aber nicht unterstütz.

Den geänderten Anforderungen, bedingt durch die Weiterentwicklung der Simulationsprogramme, muß durch eine Anpassung des Schemas Rechnung getragen. Dieser Übergang von einem alten zu einem neuen Schema wird als Schemaevolution bezeichnet. Der Prozeß der Schemaevolution entspricht genau dem Wechsel auf ein anderes Format für die Materialdaten.

5.6.1 Änderungsoperationen

In diesem Kapitel soll anhand einer Beispiel-Operation gezeigt werden, wie sich eine solche Schemaänderung auf das Informationssystem EvolMat auswirkt. Als Änderungsoperationen werden *primitive* und *komplexe* Transformationen unterschieden. Unter primitiven Transformationen sind diejenigen zu verstehen, die eindeutig einer Klasse zuzuordnen sind. Dies bedeutet aber nicht, daß die indirekten Auswirkungen im Ernstfall nicht auch das gesamte Schema betreffen können. Bei komplexen Transformationen sind gleichzeitig mehrere Klassen betroffen.

Zu den primitiven Operationen gehören alle Arten von Löschen, Hinzufügen oder Ändern von Klassen oder Attributen. Eine Klasse darf allerdings nur gelöscht werden, wenn sie keine Unterklassen besitzt. Ebenso ist die Generalisierung und Spezialisierung einer Klasse eine primitive Transformation. Zwei häufig verwendete komplexe Transformation sind das Verschieben eines Attributes in die Oberklasse und das Verschmelzen mehrerer gleicher Attribute aus den Unterklassen zu einem Attribut der Oberklasse.

5.6.2 Beispiel-Szenario

Das Simulationsprogramm SIGMA wird zur Zeit um die Fähigkeit erweitert, auch Mischungen von zwei Materialien bzw. einem Material und einem Füllstoff berechnen zu können. Die Qualität einer Mischung wird mit der Dispergiergüte (Kapitel 2.2.2 SIGMA) beschrieben. Die Dispergiergüte setzt sich aus zwei Werten zusammen: der Zugfestigkeit und der Porösität der Mischung. Beide Werte gehören in die Gruppe der Technologiedaten, werden aber nur bei der Mischung mit Füllstoffen benötigt. Daher wird eine neue Klasse `Dispergiergute` mit den beiden Parametern `zugfest` und `poroes` als Unterklasse der Technologiedaten eingeführt. Die Änderungen werden dabei nur am globalen Schema durchgeführt. Im folgenden werden die einzelnen Schritte zur

Anpassung des Informationssystems aufgezeigt. Dieser Prozeß ist noch nicht vollständig automatisiert, die Interaktion beschränkt sich aber auf wenige Stellen.

5.6.2.1 Schemaänderung in VARLET

Die Änderung des Schemas, also das Einfügen der Unterklasse Diepsergierguete, kann im OO-View von VARLET vorgenommen werden. Aufgrund der in Kapitel 5.4.1 beschriebenen Problematik der mehrdeutigen Abbildung des objekt-orientierten Schemas auf ein relationales Schema, wird auch hier dergleiche Weg beschritten. Um bei der Migration eine Unterklasse zu erhalten, wird die Relation Technologiedaten um eine Variante ergänzt. Die Variante enthält zusätzlich die beiden neuen Parameter für die Zugfestigkeit und die Porösität. Jetzt kann das relationale Schema wieder migriert werden. Dabei werden die beim ersten Entwurf durchgeführten Äquivalenztransformationen automatisch wieder angewendet. Daher braucht jetzt nur noch die neu hinzugekommene Klasse umbenannt zu werden (ebenfalls eine äquivalente Transformation) und das neue Schema ist fertig. Da ale durchgeführten Änderungen Äquivalenztransformationen sind, bleibt die eindeutige Abbildung vom relationalen auf das objekt-orientierte Schema erhalten.

5.6.2.2 Generierung der Informationsdateien und Zugriffsschicht

Nach der Anpassung des OOSchemas können aus VARLET die benötigten Informationen zur Ansteuerung des ObjectDRIVERS generiert werden. Dies ist zum einen der SQL-View, der die Beschreibung des relationalen Schemas enthält, und zum anderen der ODMGView und der XMI-View. Aus dem ODMG-View wird das neue Mapping-Schema generiert. Im Vergleich zum Original-Mapping ist dabei eine weiter Klasse aufgeführt:

```
class Dispergierguete inherit Technologiedaten
    type Tuple
    (
        zugfest     Double on DICHTEDATEN.ZUGFESTIGKEIT,
        poroes      Double on DICHTEDATEN.POROESITAET
    )
end;
```

Abbildung 39 Die neu erzeugte Klasse im ObjectDRIVER-Mapping

Aus dem XMI-View können dann wieder nach dem oben beschriebenen Verfahren die Klassen für die Zugriffsschicht generiert werden. Dabei wird die zusätzliche Datei `Dispergierguete.java` generiert.

5.6.2.3 Anpassung der Sichten und Filter

Damit die neu generierte Zugriffsschicht genutzt werden kann, müssen auch die Sichten bzw. Filter für die einzelnen Simulationsprogramme nach Bedarf neu erzeugt werden.

Die angepaßte Sicht für das Programm SIGMA ist auszugsweise in Abbildung 40 dargestellt. Die hier durchgeführte Schemaänderung hat keine Auswirkung auf die anderen Programme, so daß nur der Filter für daß Simulationsprogramm SIGMA neu generiert werden muß. Die Abhängigkeiten, welche Filter neu zu erstellen sind, müssen zurr Zeit noch vom Benutzer festgelegt werden, und die entsprechende Generierung durchgeführt werden.

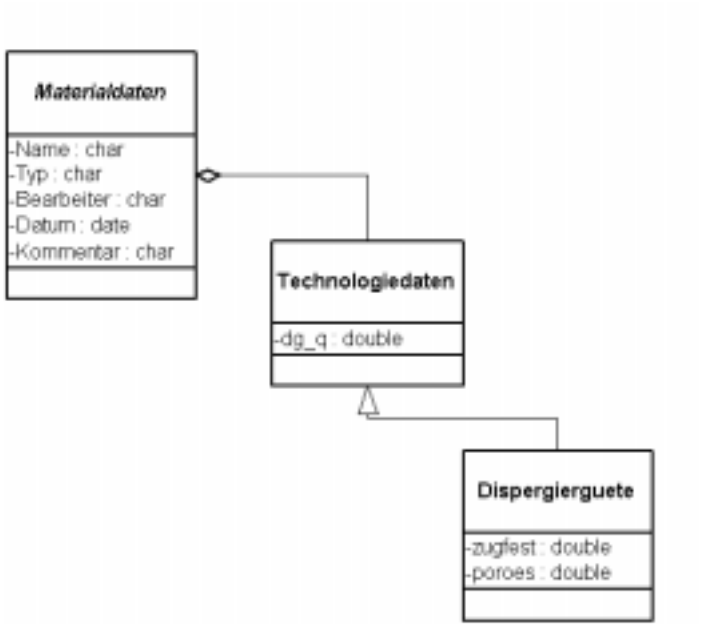


Abbildung 40 geänderte Sichtendefinition für SIGMA

6 Zusammenfassung und Ausblick

Im KTP, dem Institut für Kunststofftechnik, wird an der Forschung und Entwicklung von Schneckenmaschinen gearbeitet. Im Rahmen der Forschungsprojekte sind mehrere Simulationsprogramme zur Berechnung der Maschinen entwickelt worden. Jedes der Programme benötigt Informationen über das in der Maschine verarbeitete Material. In dieser Arbeit wurde Informationssystem entworfen, welches die Materialdaten verwalten und den Simulationsprogrammen entsprechend aufbereitet zur Verfügung stellen kann.

Dazu wurden zunächst die Simulationsprogramme auf die von ihnen benötigten Daten hin untersucht. Aus diesen Daten wurde für jedes Programm ein Modell entwickelt, das die objekt-orientierten Datenstrukturen in der UML-Notation enthält. Die dabei entstandene Klassenstruktur zeichnet sich durch eine baumartige Struktur aus. Dabei werden die Materialeigenschaften, nach physikalischen und chemischen Eigenschaften gruppiert, an die Klasse zur Modellierung der Materialdaten aggregiert, wodurch die Beschreibung eines Materials durch mehrere Gruppen von spezifischen Eigenschaften reflektiert wird.

Die Eingabe der Materialdaten erfolgt in einem grafischen, dialog-basierten System. Dabei wurde versucht, die Sequentialität der Eingabe möglichst weit zu reduzieren, trotzdem aber die vorhandenen Abhängigkeiten der Baumstruktur auszunutzen. Um Inkonsistenzen bei der Dateneingabe zu vermeiden, wurden programmintern weitere Überprüfungen vorgenommen, wie z.B. die Einschränkung der Wertebereiche von Attributen.

Um die unterschiedlichen Datensätze für die Programme in einem gemeinsamen System verwalten zu können, wurden die Modelle zusammengeführt. Dazu wurden die aus der Sichtenintegration bekannten Verfahren untersucht und die für den Anwendungsfall beste Methode, die Leiterstrategie, verwendet.

Auf Basis der so entwickelten gemeinsamen Grundlage wurde die Anbindung an ein relationales Datenbank-Management-System realisiert. Für die Abbildung zwischen der objekt-orientierten und der relationalen Sichtweise wurde eine Middleware, ObjectDRIVER, eingesetzt. Die Middleware kapselt die relationalen Zugriffe auf die Datenbank gegenüber der Anwendung, so daß ein objekt-orientierter Zugriff möglich ist. Sie bietet zudem ein Transaktionsmanagement und überwacht die Eindeutigkeit der Objekte.

Die Informationen über die Schemata und die Abbildung, die ObjectDRIVER benötigt, wurden aus VARLET erzeugt. Dazu wurde VARLET um einen weiteren TextView ergänzt, der die Daten des OO-Schemas in XMI-Notation darstellt. Dieser enthält in standardisierter Form alle Daten über die Klassen, Attribute und Beziehungen. Auch die

Umsetzung der Datentypen des Modell in die Programmiersprache ist vorhanden. Aus diesen Informationen wurden die Klassen für die Zugriffsschicht generiert.

Die Auswahl der Daten für die Simulationsprogramme erfolgt über Filter. Die Definition der Filter basiert auf den zu Beginn der Modellierung entworfenen Sichten. Darin sind genau die Daten enthalten, die das jeweilige Programm benötigt.

Da die modellierte Datenstruktur durch Forschung und neue Entwicklungen im Bereich der Kunststofftechnik einem Wandel unterliegt, muß das Informationssystem auf Änderungen des Schemas reagieren können. Änderungen werden dabei nur an dem entwickelten globalen Schema erlaubt. Mit dieser Einschränkung kann VARLET zur Unterstützung der Schemaevolution eingesetzt werden. In einem Beispiel-Szenario wurde gezeigt, wie die Schemaänderungen in VARLET propagiert werden. In einem semi-automatischen Prozeß konnten sämtliche Teile des Informationssystems, die von den Änderungen betroffen waren, neu generiert werden.

Als Weiterentwicklung des Systems ist einen größere Integration der einzelnen Schritte und weitere Automation des Generierungsprozeß wünschenswert. Dafür müssen an verschiedenen Stellen jedoch noch die Voraussetzungen geschaffen werden. Auf Seite der Anwendungsprogramme muß die Anbindung über Textdateien durch einen direkten Austausch der Objektdaten erfolgen. Dazu sind mehrere Lösung denkbar, etwa der Einsatz von vorhanden standardisierten Schnittstellen wie COM+/DCOM unter Windows oder die Anbindung mittels einer Corba-Architektur.

In VARLET fehlen noch die Möglichkeiten, ein objekt-orientiertes Modell durch Benutzerinteraktion so mit Informationen anzureichern, daß eine eindeutige Abbildung vom OO-Schema auf das relationale Modell möglich ist. Desweiteren könnten auch die Daten des relationalen Schemas nach XMI-Standard exportiert, sowie XMI-Dateien importiert werden.

Die Middleware ObjectDRIVER war zum Zeitpunkt der Diplomarbeit immer noch im Beta-Stadium. Dies betrifft vor allem die Entwicklung einer ODMG-konformen C++-Schnittstelle. Aus diesem Grund wurde für die Zugriffsschicht und die Filter Java-Code generiert, der aus EvolMat nur über der Umweg des Java-Native-Interface (JNI) ausgeführt werden kann.

Literaturverzeichnis

[Balz96] Balzert, Helmut: Lehrbuch der Software-Technik: Software-Entwicklung. Heidelberg; Berlin; Oxford: Spektrum, Akad., Verl, 1996

[BKKK87] Jay Banerjee, Won Kim, Hyoung-Joo Kim and Henry F. Korth: Semantics and Implementation of Schema Evolution in Object-Oriented Databases. SIGMOD Record (Proc. Conf. on Management of Data), vol. 16, no. 3, Mai 1987, pp. 311-322

[BLN86] Batani, C.

[Cat94] Cattel, R. (Hrsg.): The Object Database Standard: ODMG-93. Morgan Kaufmann, San Mateao, CA, 1994

[Gam95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns. Massachusetts: Addison-Wesley, 1995

[Heu95] Andreas Heuer, Gunter Saake: Datenbanken: Konzepte und Sprachen, 1. Auflage Bonn, Albany [u.a.] Internat. Thomson Publ. 1995

[Hol97] Jens Holle: Ein Generator für integrierte Werkzeuge am Beispiel der objekt-relationalen Datenbankschemaintegration. Diplomarbeit am Lehrstuhl Softwaretechnik der Universität Paderborn, Juli 1997

[HP87] HPSQL - Database Administration Guide, Hewlett Packard Inc., Juni 1987

[HTW95] Hahn W., Toenniessen F., Wittkowski A.: Eine objektorientierte Zugriffsschicht zu relationalen Datenbanken. Informatik Spektrum, Band 18, 1995, S. 143-151

[HTW95] Hahn W., Toenniessen F., Wittkowski A.: Eine objektorientierte Zugriffsschicht zu relationalen Datenbanken. Informatik Spektrum, Band 18, 1995, S. 143-151

[Info92] INFORMIX Guide to SQL Reference, Informix Software, Inc., September 1992, Part-No. 000-7193 Rev. A.

[Jah99] Jens H. Jahnke: Management of Uncertainity and Inconsistency in Database Reengineering Process. Dissertation am Lehrstuhl Softwaretechnik der Universität Paderborn, September 1999

[JW99] Jens H. Jahnke, Jörg Wadsack: Integration of Analysis and Redesign Activities in Information System Reengineering. Proceedings of CSMR´99, Amsterdam, NL. IEEE Press

[Kre97] Karsten Kretschmer: Physiaklisch-mathematische Modellbildung für die Dispergierung mineralischer Füllstoffe in einem Gleichdrall-Doppelschneckenextruder. Diplomarbeit am Lehrstuhl Kunststofftechnik der Universität Paderborn, Dezember 1997

[Lang95] Stefan M. Lang, Peter C. Lockemann: Datenbankeinsatz. Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona, Budapest: Springer, 1995

[LDAJ99] Lebastard, Demphlous, Aguiléra, Jautzy: ObjectDRIVER Reference Manual, Version 1.1 Beta:CERMICS Database Team, 1999

[Lef95] M. Lefering. Integrationswerkzeuge in einer Softwareentwicklungsumgebung. Informatik, Verlag Shaker, 1995.

[Lip95] Lippmann, Stanley B.: C++ Einführung und Leitfaden. Bonn, München: Massachusetts, Addison-Wesley, 1991

[Nic97] Nickel, Ulrich: Konzeption einer objektorientierten Bibliothek für gemischt anlog/digitale Schaltkreise. Diplomarbeit am Lehrstuhl Softwaretechnik der Universität Paderborn, Oktober 1997

[ODMG97] R.G.G. Cattell et al. Object Database Standard: ODMG 2.0, Morgan Kaufmann Publishers, Inc. 1997. ISBN 1-55860-463-4.

[OMG98] OMG: XML Metadata Interchange (XMI) Proposal to the OMG OA&DTF RFP3. OMG Document ad/98-10-05

[Pot92] Potente, H. (Hrsg.): Kunststofftechnisches Seminar. Paderborn

[Rao89] Rao, Natti S.: Formeln der Kunststofftechnik, Karl Hanser Verlag. München/Wien, 1989

[RON94] RONDO EE/R-Editor Benutzerhandbuch, Version 2.2g, PRO DV Software GmbH, Martin-Schmeißer-Weg 14, 44227 Dortmund, Juni 1994

[Ros97] Grzegorz Rosenberg, editor: Handbook of Graph Grammars and Computing by Graph Transformations. World Scientific, Singapore, 1997

[Rumb93] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen: Object-oriented modeling and design <dt.>. München; Wien: Hanser; London: Prentice-Hall Internat., 1993

[Sae95] Saechtling, Hansjürgen:Kunststoff Taschenbuch, 26. Ausgabe, Carl Hanser Verlag München/Wien, 1995

[Sch98] Schalldach, Heike: Integration von JAVA-Anwendungen mit relationalen Informationssystemen

[Schief93] Schiefer, Bernhard: Eine Umgebung zur Unterstützung von Schemaänderungen und Sichten in objektorientierten Datenbanksystemen. Dissertation Forschungszentrum Informatik, Karlsruhe, 1993

[Tre95] Tresch, Markus: Evolution in Objektdatenbanken: Anpassung und Integration bestehender Informationssysteme. Stuttgart; Leipzig: Teubner, 1995

[VDMA84] VDMA (Hrsg.): Kenndaten für die Verarbeitung thermoplastischer Kunststoffe, Band 3:Tribologie. Carl Hanser Verlag München/Wien, 1984

[Wad98] Jörg P. Wadsack: Inkrementelle Konsistenzerhaltung in der transformationsbasierten Datenbankmigration. Diplomarbeit am Lehrstuhl Softwaretechnik der Universität Paderborn, September 1998

[Zün95] A. Zündorf. PROgramierte GRaphErsetzungsSysteme (Spezifikation, Implementierung und Anwendung einer integrierten Entwicklungsumgebung), Dissertation RWRH Aachen. Deutscher Universitätsverlag (1995)

Verzeichnis der Abbildungen

Verzeichnis der Tabellen

Anhang A: Glossar

API	**A**pplication **P**rogramming **I**nterface
CAD	**C**omputer **A**ided **D**esign
CARE	**C**omputer **A**ided **Ree**ngineering
C-LAB	The **C**ooperative **C**omputing and **C**ommunication **Lab**oratory
EvolMat	**Evol**utionsorientiertes **Mat**erialverwaltungssystem
HTML	**H**yper **T**ext **M**arkup **L**anguage
KTP	Institut für **K**unststoff**t**echnik e. V. **P**aderborn
OCSI	**O**bjectoreintiert, **C**lient/**S**erver, **I**nternetbasiert
ODBC	**O**pen **D**ata**B**ase **C**onnectivity
ODMG	**O**bject **D**ata **M**anagement **G**roup
OMG	**O**bject **M**anagement **G**roup
OMT	**O**bject **M**odelling **T**echnique
OQL	**O**bect **Q**uery **L**anguage
PSI	**P**aderborner **Si**mulationsprogramm für Spritzgieß-plastifiziereinheiten
REX	**R**echnergestütze **Ex**truderauslegung
SIGMA	**Si**mulation **g**leichläufiger Doppelschnecken**ma**schinen
SQL	**S**tructered **Q**uery **L**anguage
UML	**U**nified **M**odelling **L**anguage
VARLET	**V**erified **A**nalysis and **R**eengineering of **L**egacy Database Systems Using **E**quivalence **T**ransformations
XMI	**X**ML **M**etadata **I**nterchange
XML	E**x**tensible **M**arkup **L**anguage
Wendel	Wendelverteiler, spezielles Werkzeug für Extruder

Anhang B: Materialdaten

Tabelle 2 Rheologische Materialdaten

Nullviskosität	a
Übergangsschergeschwindigkeit	b
Steigung	c
Bezugstemperatur	t_b
Standardtemperatur	t_s
1. Konstante des WLF-Anstazes	c_1
2. Konstante des WLF-Anstazes	c_2
Aktivierungsenergie	e
Wandschubspannung 1	krit_Wand_1
Wandschubspannung 2	krit_Wand_2
Massetemperatur 1	krit_Temp_1
Massetemperatur 2	krit_Temp_2

Tabelle 3 Tribologische Materialdaten

Zylinderreibungskoeffizient	mue_z
inner Reibungskoeffizient	mue_i
Schneckenreibungskoeffizient	mue_s
temperaturabhängiger Reibungskoeffizient	mue_z1

Tabelle 4 Technologische Materialdaten

mininmaler Granulatdurchmesser	dg_min
maximaler Granulatdurchmesser	dg_max
mittlerer Granulatdurchmesser	dg_q
Scherfestigkeit des Feststoffes	tau_scher

Tabelle 5 Thermodynamische Materialeigenschaften

Glasübergangstemperatur	t_g
Kristallisationstemperatur	t_k
Wärmeleitfähigkeit	lambda_0
Wärmeleitfähigkeit des Feststoffes	lambda_f
Steigung der Wärmeleitfähigkeit	lambda_m
Wärmekapazität	cp_0
Steigung der Wärmekapazität	cp_m
Aufschmelzenthalpie	h_a
Feststoffenthalpie	h_f

Tabelle 6 Dichtedaten der Materialeigenschaften

Feststoffdichte	rho_f
Schüttdichte	rho_s
Volumenfunktion	v_0
Steigung der Volumenfunktion	v_m
spezifische Dichte	rho_0
Steigung der spezifischen Dichte	rho_m

Anhang C: Eingabedateien

Relationales Schema

```
CREATE SCHEMA DiplomArbeit

CREATE TABLE Material(
(* Variant 1 of 1 *)
     Name varchar(20),
     Kommentar varchar(100),
     Bearbeiter varchar(20),
     Typ varchar (20),
     Datum date,
     ID_Thermodynamik integer,
     ID_Technologie integer,
     ID_Dichte integer,
     ID_Tribologie integer,
     ID_Rheologie integer,
     primary key ( Name,  Typ ))

CREATE TABLE Dichtedaten(
(* Variant 1 of 2 *)
     rho_s double,
     rho_f double,
     v_null double,
     v_m double,
     rho_null double,
     rho_m double
     ID_Dichte integer,
     primary key ( ID_Dichte ))

CREATE TABLE Thermodynamikdaten(
(* Variant 1 of 2 *)
     t_g-t_k double,
     lambda_f double,
     lambda_null double,
     lambda_m double,
     cp_null double,
     cp_m double,
     h_a double,
     h_f double,
     ID_Thermodynamik integer,
     primary key ( ID_Thermodynamik ))

CREATE TABLE Rheologiedaten(
(* Variant 2 of 3 *)
     a double,
     b double,
     c double,
     t_b double,
     t_s double,
     c_1 double,
     c_2 double,
     e double,
     ID_Rheologie integer,
     primary key ( ID_Rheologie ))

CREATE TABLE Techniologiedaten(
(* Variant 1 of 1 *)
     dg_min double,
     dg_max double,
     dg_q double,
     tau_scher double,
     ID_Technologie integer,
     primary key ( ID_Technologie ))

CREATE TABLE Tribologiedaten(
(* Variant 1 of 1 *)
     mue_z double,
     mue_i double,
     mue_s double,
     mue_z1 double,
     ID_Tribologie integer,
     primary key ( ID_Tribologie ))
```

```
ALTER TABLE Material ADD (
FOREIGN KEY ( ID_Dichte ) REFERENCES Dichtedaten ( ID_Dichte )
)

ALTER TABLE Material ADD (
FOREIGN KEY ( ID_Tribologie ) REFERENCES Tribologiedaten ( ID_Tribologie )
)

ALTER TABLE Material ADD (
FOREIGN KEY ( ID_Thermodynamik ) REFERENCES Thermodynamikdaten ( ID_Thermodynamik )
)

ALTER TABLE Material ADD (
FOREIGN KEY ( ID_Technologie ) REFERENCES Techniologiedaten ( ID_Technologie )
)

ALTER TABLE Material ADD (
FOREIGN KEY ( ID_Rheologie ) REFERENCES Rheologiedaten ( ID_Rheologie )
)
```

Mapping-Schema für ObjectDRIVER

```
class Materialdaten on MATERIAL
   type Tuple
   (
     name            String on MATERIAL.NAME,
     kommentar         String on MATERIAL.KOMMENTAR,
     bearbeiter         String on MATERIAL.BEARBEITER,
     typ               String on MATERIAL.TYP,
     datum            Date on MATERIAL.DATUM,
     dichte          Dichtedaten on MATERIAL.ID_DICHTE inverse Dichtedaten.materials
      rheologie              Rheologiedaten on RHEOLOGIEDATEN.ID_RHEOLOGIE inverse
Rheologiedaten.materials
      tribologie          Tribologiedaten on TRIBOLOGIEDATEN.ID_TRIBOLOGIE inverse
Tribologiedaten.materials
     technologie       Technologiedaten on TECHNOLOGIEDATEN.ID_TECHNOLOGIE inverse
Technologiedaten.materials
      thermodynamik      Thermodynamikdaten on THERMODYNAMIKDATEN.ID_THERMODYNAMIK
inverse Thermodynamikdaten.materials
   )
end;

class Dichtedaten on DICHTEDATEN
   type Tuple
   (
      rho_s        Double on DICHTEDATEN.RHO_S,
      rho_f        Double on DICHTEDATEN.RHO_F,
      materials    Set on M<MATERIAL>
                   (
                     aMaterial Materialdaten on M.MATERIAL,
                     join DICHTEDATEN to M by (DICHTEDATEN.ID_DICHTE == M.ID_DICHTE)
                   )
                   inverse Materialdaten.dichte
   )
end;

class SpezifischeDichte inherit Dichtedaten
   type Tuple
   (
      rho_null         Double on DICHTEDATEN.RHO_NULL,
      rho_m            Double on DICHTEDATEN.RHO_M
   )
end;

class SpezifischesVolumen inherit Dichtedaten
   type Tuple
   (
      v_null            Double on DICHTEDATEN.V_NULL,
      v_m               Double on DICHTEDATEN.V_M
   )
end;

class Thermodynamikdaten on THERMODYNAMIKDATEN
   type Tuple
   (
      t_g_t_k            Double on THERMODYNAMIKDATEN.T_G_T_K,
      lambda_f         Double on THERMODYNAMIKDATEN.LAMBDA_F,
      lambda_null         Double on THERMODYNAMIKDATEN.LAMBDA_NULL,
      lambda_m         Double on THERMODYNAMIKDATEN.LAMBDA_M,
      cp_null            Double on THERMODYNAMIKDATEN.CP_NULL,
      cp_m            Double on THERMODYNAMIKDATEN.CP_M,
      materials         Set on M<MATERIAL>
                      (
```

```
                        aMaterial Materialdaten on M.MATERIAL,
                                                  join  THERMODYNAMIKDATEN  to  M  by
(THERMODYNAMIKDATEN.ID_THERMODYNAMIK == M.ID_THERMODYNAMIK)
                     )
                     inverse Materialdaten.thermodynamik
   )
end;

class Amorph inherit Thermodynamikdaten
   type Tuple
   (
      h_a               Double on THERMODYNAMIKDATEN.H_A
   )

class Teilkristallin inherit Amorph
   type Tuple
   (
      h_f               Double on THERMODYNAMIKDATEN.H_F
   )

class Rheologiedaten on RHEOLOGIEDATEN
   type Tuple
   (
      a                 Double on RHEOLOGIEDATEN.A,
      b                 Double on RHEOLOGIEDATEN.B,
      c                 Double on RHEOLOGIEDATEN.C,
      t_b                 Double on RHEOLOGIEDATEN.T_B,
      t_s                 Double on RHEOLOGIEDATEN.T_S,
      c_1                 Double on RHEOLOGIEDATEN.C_1,
      c_2                 Double on RHEOLOGIEDATEN.C_2,
      e                   Double on RHEOLOGIEDATEN.E,
      krit_wand_1           Double on RHEOLOGIEDATEN.KRIT_WAND_1,
      krit_wand_2           Double on RHEOLOGIEDATEN.KRIT_WAND_2,
      krit_temp_1           Double on RHEOLOGIEDATEN.KRIT_TEMP_1,
      krit_temp_2           Double on RHEOLOGIEDATEN.KRIT_TEMP_2,
      materials         Set on M<MATERIAL>
                     (
                        aMaterial Materialdaten on M.MATERIAL,
                       join RHEOLOGIEDATEN to M by (RHEOLOGIEDATEN.ID_RHEOLOGIE ==
M.ID_RHEOLOGIE)
                     )
                     inverse Materialdaten.rheologie
   )
end;

class Technologiedaten on TECHNOLOGIEDATEN
   type Tuple
   (
      dg_min            Double on TECHNOLOGIEDATEN.DG_MIN,
      dg_max            Double on TECHNOLOGIEDATEN.DG_MAX,
      dg_q              Double on TECHNOLOGIEDATEN.DG_Q,
      tau_scher           Double on TECHNOLOGIEDATEN.TAU_SCHER,
      materials         Set on M<MATERIAL>
                     (
                        aMaterial Materialdaten on M.MATERIAL,
                     join TECHNOLOGIEDATEN to M by (TECHNOLOGIEDATEN.ID_TECHNOLOGIE
== M.ID_TECHNOLOGIE)
                     )
                     inverse Materialdaten.technologie
   )
end;

class Tribologiedaten on TRIBOLOGIEDATEN
```

```
   type Tuple
   (
      mue_z            Double on TRIBOLOGIEDATEN.MUE_Z,
      mue_i            Double on TRIBOLOGIEDATEN.MUE_I,
      mue_s            Double on TRIBOLOGIEDATEN.MUE_S,
      mue_z1            Double on TRIBOLOGIEDATEN.MUE_Z1,
      materials         Set on M<MATERIAL>
                     (
                        aMaterial Materialdaten on M.MATERIAL,
                       join TRIBOLOGIEDATEN to M by (TRIBOLOGIEDATEN.ID_TRIBOLOGIE
== M.ID_TRIBOLOGIE)
                     )
                     inverse Materialdaten.tribologie
   )
end;
```

XMI-View