Diplomarbeit

Wissensmanagement für die Entwicklung von Feststoffprozessen: Assetorientierte Analyse, Design und Implementierung eines Information-Brokers

Lars Diestelhorst

24. Juni 2004

Betreuung:

Prof. Dr. Joachim W. Schmidt
Arbeitsbereich Softwaresysteme
TECHNISCHE UNIVERSITÄT HAMBURG-HARBURG

Prof. Dr. Joachim Werther
Arbeitsbereich Verfahrenstechnik I
TECHNISCHE UNIVERSITÄT HAMBURG-HARBURG

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt zu haben. Die verwendete Literatur und sonstigen Hilfsmittel sind vollständig angegeben.

Hamburg, 24. Juni 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzfassung

Die Prozessentwicklung in der Verfahrenstechnik ist stark durch die Iteration von Synthese und anschließender Analyse des Fließschemas geprägt. Bei der Feststofftechnik hängt der Erfolg beider Schritte in einem hohen Maß von den Erfahrungen des Entwicklers ab und kann daher durch Wissensmanagement gefördert werden. Die Entwicklung eines Informations-Systems für das Wissensmanagement im Bereich des *Computer Aided Process Engineerings*(CAPE) ist daher der Zweck dieser Arbeit.

Die Komplexität der Wissensdomäne motiviert eine assetorientierte Vorgehensweise bei der Entwicklung des Systems. Diese Methodik beruht auf einer starken Trennung zwischen medialer Wahrnehmung von Entitäten und zugehörigem konzeptuellem Wissen und folgt damit den Überlegungen von Peirce und Cassirer zur Entitätenmodellierung. Zusätzlich wird die assetorientierte Entwicklung durch Dynamik und Offenheit der Systemplattform geprägt. Diese ermöglichen zusammen mit der Trennung von Inhalt und Konzept einen evolutionären Wissens- und Metawissensprozess. Bei der Durchführung der Arbeit hat sich gezeigt, dass das assetorientierte Modell einen natürlichen Erkenntnisprozess im Anwendungsgebiet des CAPE unterstützt und daher ein geeignetes Metamodell für das Wissensmanagement im CAPE ist.

Die Umsetzung der Analyseergebnisse erfolgte auf Basis des INFOASSET BROKERS sowie den Opensource-Projekten TOMCAT und AXIS der APACHE SOFTWARE FOUNDATION.

1. Einleitung

1.1. Motivation

Die Simulation von Fluidprozessen in der technischen Verfahrenstechnik ist bereits etabliert [Sch95] und ermöglicht den günstigen Vergleich von unterschiedlichen Prozessalternativen. Die Entwicklung entsprechender Simulatoren für Feststoffprozesse ist jedoch noch nicht so weit fortgeschritten. Dies hat unterschiedliche Gründe, die sich aus prinzipbedingten Unterschieden zwischen Fluidprozessen und Feststoffprozessen ergeben.

- Der Materialstrom eines Feststoffprozesses beinhaltet neben möglichen flüssigen oder gasförmigen Phasen eine oder mehrere disperse Phasen. Die Beschreibung dieser dispersen Phasen ist wesentlich komplizierter als die Beschreibung einer fluiden Phase. Das verfahrenstechnische Verhalten einer fluiden Phase ist nämlich bereits durch wenige physikalisch-chemische Eigenschaften wie z. B. Druck, Temperatur, Stoffkonzentrationen und Stoffeigenschaften festgelegt. Hingegen werden Eigenschaften eines dispersen Systems auch durch die Attribute des Partikelkollektivs bestimmt. So hängt die Fließfähigkeit von Schüttgütern in komplexerweise von der Partikelgröße, der Partikelform sowie von Porosität und Feuchtegehalt des Haufwerkes ab [Ros98].
- Für die Simulation eines Prozessschrittes können unterschiedlich detaillierte Simulationsmodelle verwendet werden. Simulationsmodelle, die auch als Prozessmodelle bezeichnet werden, reichen von so genannten *Shortcut*-Modellen bis zu empirischen Koeffizientenmodellen (siehe Abschnitt 3.5.1). Die Simulationsmodelle benötigen als Dateneingabe zunächst eine Beschreibung des Eingangsmaterialstroms, die Pflichtangaben in Abhängigkeit von der Art des Simulationsmodells enthält. Zusätzlich müssen physikalische Apparateparameter angegeben werden, die ebenfalls in die Gleichungen des Simulationsmodells einfließen. Die Werte der Materialparameter sind entweder das Resultat einer Messung oder eines vorherigen Simulationsschrittes. Die Apparateparameter ergeben sich hingegen aus der Auswahl des jeweiligen Apparats im entsprechenden Prozessschritt und seiner Dimensionierung. Die Werte dieser beiden Parametertypen lassen sich unabhängig vom ausgewählten Modell bestimmen. Zusätzlich existieren jedoch noch Modellparameter, die abhängig von der Kombination aus Apparat und Materialstrom sind und im Wesentlichen empirisch bestimmt werden.

Die Erfassung, Pflege und Bereitstellung dieser Modellparameter sind daher der Kern des Wissensmanagements im Rahmen der Simulation von Feststoffprozessen. Gegenstand dieser Arbeit sind daher Analyse, Design und prototypische Implementierung eines webbasierten Informationssystems zur Unterstützung des *Computer Aided Process Engineering* (CAPE).

Das zu entwickelnde System soll eine MICROSOFT ACCESS basierte Lösung im Arbeitsbereich Verfahrenstechnik I ersetzen. Darüber hinaus soll die Anwendung auch als Basis für

das Wissensmanagement im Bereich der Prozessentwicklung dienen. Die geringe Erfahrung mit Wissensmanagement im CAPE motiviert ein inkrementelles und evolutionäres Vorgehen. Daher wird im Rahmen der Arbeit eine assetorientierte Herangehensweise im Sinne von Cassirer und Peirce verwendet. Ein weiteres Ziel ist es, praktische Erfahrungen mit der assetorientierten Modellierung zu sammeln und auf ihre Zweckmäßigkeit zu untersuchen.

1.2. Strukturierung

Nach der Einleitung im ersten Kapitel wird im zweiten Kapitel die assetorientierte Analyse vorgestellt, die versucht dem Semantikverständnis von Peirce und Cassirer gerecht zu werden. In Kapitel 3 werden die fachlichen Konzepte des CAPE beschrieben und ein assetorientiertes Modell der Anwendungsdomäne entwickelt. Da derzeit noch kein adäquates Laufzeitsystem für die direkte Umsetzung der Ergebnisse aus der assetorientierten Analyse existiert, wird zunächst in Kapitel 4 das Implementierungsumfeld beschrieben und anschließend das Modell der assetorientierten Analyse auf die Konzepte des INFOASSET BROKERS abgebildet. Die Realisierung des in Kapitel 5 dargestelleten Entwurfs wird im anschließenden Kapitel 6 beschrieben. Im letzten Kapitel wird die Arbeit zusammengefasst und ein Ausblick auf weiterführende Projekte gegeben.

Danksagung

Die vorliegende Diplomarbeit wurde am Arbeitsbereich Softwaresysteme (STS) der Technischen Universität Hamburg-Harburg erstellt und in enger Zusammenarbeit mit den Arbeitsbereichen Prozess- und Anlagentechnik (VT3) und Verfahrenstechnik I (VT1) durchgeführt. Deshalb gilt mein besonderer Dank Herrn Prof. Dr. Joachim W. Schmidt (STS) sowie den Professoren Dr. Joachim Werther (VT1) und Dr. Günter Gruhn (VT3). Darüber hinaus verdankt die Arbeit sehr viel der engagierten Unterstützung durch wissenschaftliche Mitarbeitern dieser Arbeitsbereiche, insbesondere Dr. Ernst-Ulrich Hartge, Rainer Marrone, Matthias Pogodda, Claus Reimers, Daniel Schwier und Dr. Hans-Werner Sehring (in alphabetischer Reihenfolge) sowie der Hilfe durch Freunde und Verwandte.

2. Konzept- und inhaltsorientierte Entitätenmodellierung

Eine Grundlage der Softwaretechnik ist die Annahme, dass Teilbereiche der realen Welt durch isomorphe softwaretechnische Modelle beschrieben werden können [Eve95, Sch99]. Diese Forderung an die Softwaretechnik wurde bereits im klassischen Sinne als erkenntnisphilosophische Forderung von Ludwig Wittgenstein im „*Tractatus logico-philosophicus*“ beschrieben [Wit63]. Interessanterweise sieht Wittgenstein seine eigenen Bemühungen zur Lösung aller Probleme der Beschreibung in [Wit67] als gescheitert an.

Für spezielle Aufgaben wurden in der Softwaretechnik verschiedene erfolgreiche Metamodelle zur Entitätenmodellierung entwickelt. So haben sich im Bereich der Datenbankmodellierung *Entity-Relationship-Modelle* [Tha93] und im Bereich der Programmierung das objektorientierte Paradigma zusammen mit der *Unified Modelling Language* (UML) [Obj03, FS00] als erfolgreich erwiesen. Es lässt sich jedoch feststellen, dass es ein jedem Anspruch gerecht werdendes Universal-Metamodell zur Entitätenmodellierung nicht geben wird.

Im Folgenden wird ein Ansatz für die Entitätenmodellierung präsentiert, der sich an den Ideen von Peirce [Pei31] und Cassirer [Cas02] zur Entitätenmodellierung orientiert und an der Technischen Universität Hamburg-Harburg im Arbeitsbereich Softwaresysteme entwickelt wurde [SS03, Seh03].

2.1. Konzeptionelle und inhaltliche Erkenntnis

Die Grundidee des Ansatzes beruht auf dem Semiotikverständnis von Cassirer. Unter Semiotik wird allgemein „die Wissenschaft von Zeichenprozessen in Natur und Kultur“ verstanden [Deu04]. Innerhalb der Semiotik stellt das Semiotische Dreieck, das bereits von Aristoteles eingeführt wurde (siehe Abbildung 2.1), einen stark diskutierten Sachverhalt dar.

Nach Cassirer nimmt ein Betrachter einerseits nur solche Entitäten – konkrete oder abstrakte – zur Kenntnis, für die er ein Konzept besitzt und andererseits existiert für jedes seiner Konzepte eine Entität. Durch diese Dualität stellt sich die Frage, wie mit unbekannten Entitäten

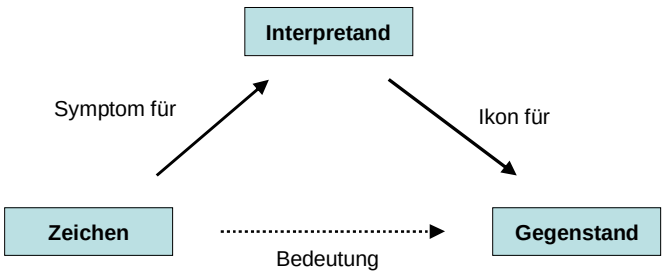


Abbildung 2.1.: Semiotisches Dreieck nach Peirce [Wik04]

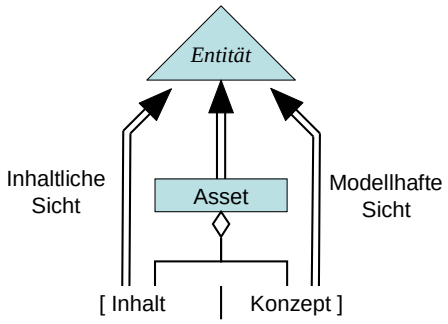


Abbildung 2.2.: Grundidee des Asset-Metamodells nach [SS03]

umgegangen wird. Entweder werden sie nicht wahrgenommen, weil kein passendes Konzept vorhanden ist, oder es wird ein neues, noch nicht sehr aussagekräftiges *ad-hoc*-Konzept gebildet, das in einem Lernprozess weiter verfeinert wird. Umgekehrt haben Menschen aber auch die Fähigkeit, auf der Basis von Vorstellungen, also Konzepten, neue Dinge zu entwerfen und zu schaffen.

Die Wahrnehmung von konkreten Entitäten basiert auf den menschlichen Sinnen. Um jedoch abstrakte Entitäten verstehen zu können, wird zusätzlich die Fähigkeit zur Kommunikation benötigt, die – abhängig vom Sinneskanal – im Wesentlichen wort- oder schriftbasiert ist. Bei konkreten Entitäten steht also die Erkenntnis und bei abstrakten Entitäten das Verständnis der jeweiligen Entität im Vordergrund.

Im Zentrum des Modells befindet sich daher einerseits eine sinnliche, mediale und andererseits eine sinnhafte, konzeptuelle Sichtweise der jeweiligen konkreten oder abstrakten Entität. In einem computerunterstützten System zum Wissensmanagement bietet die Kopplung beider Sichtweisen einen Mehrwert für den Nutzer (vgl. Abschnitt 2.1.3) und wird deshalb im vorgestellten Entitäten-Modell als *Asset*[1] bezeichnet.

Um das vorgestellte Entitätenmodell von anderen abzugrenzen, wird es im Folgenden als *Asset-Metamodell* bezeichnet. Eine graphische Darstellung der Grundidee des Asset-Metamodells ist in Abbildung 2.2 zu sehen.

2.1.1. Unterstützung medialer Aspekte einer Entität

Die Wahrnehmung von Entitäten basiert wie bereits erwähnt auf den menschlichen Sinnen. Neben Hören und Sehen sind aber auch Fühlen, Riechen und Schmecken wichtige Werkzeuge, mit denen wir unsere Umwelt wahrnehmen. Um die Atmosphäre einer alten Bibliothek zu erfassen, reicht ein einziger dieser Sinne nicht aus, sondern nur eine Komposition. Zusätzlich wird eine Bibliothek abhängig von ihren Besuchern vor dem persönlichen Hintergrund wahrgenommen. Daher wird ein Architekt beim Betrachten der Bibliothek andere Schwerpunkte setzen als ein Bibliothekar oder ein dort lernender Student.

Soll mit einem Softwaresystem dargestellt oder beschreiben werden, was eine Bibliothek ist, steht man vor einem Dilemma. Zunächst einmal ist die Wahrnehmung immer subjektiv

[1] „Vermögenswert eines Unternehmens “ [DUD01]

und vom Kontext des Betrachters abhängig. Damit sind digitale Bilder einer Bibliothek keine objektive Beschreibung, sondern sie sind immer nur eine Interpretation des Fotografen. Zusätzlich lassen sich die verschiedenen sinnlich wahrgenommenen Reize nicht vollständig digital reproduzieren. So kann der Geruch einer Bibliothek von einem gewöhnlichen Computersystem nicht wiedergegeben werden. Statt der eigenen sinnlichen Wahrnehmung der zu beschreibenden Entität können dem Nutzer des Informationssystems nur digitale, multimediale Inhalte zur Verfügung gestellt werden, die nur einen subjektiv selektierten Teil der Wirklichkeit darstellen.

Die einzelnen multimedialen Beiträge können oftmals zur inhaltlichen Beschreibung mehrerer Entitäten verwendet werden. Beispielhaft kann ein Gruppenbild als Beschreibung jeder einzelnen auf dem Foto abgebildeten Person gesehen werden. Häufig wird die Person dann durch eine Markierung hervorgehoben oder die Position auf dem Bild angegeben. Außerdem sind digitale Inhalte selbst wieder Entitäten, die sich entsprechend Cassirers Ansatz medial und konzeptuell beschreiben lassen. Diese konzeptuellen oder inhaltlichen Kontextinformationen ermöglichen dem Nutzer, die ursprüngliche Informationsquelle zu bewerten und erzeugen damit einen objektiveren Blick auf die zu Grunde liegende Entität.

Inhalte sind in vielfältigen Formaten denkbar. Neben Bildern, Geräuschen, Animationen, Diagrammen, Filmen und Texten, die sich alle direkt bzw. durch Plug-Ins vom Softwaresystem darstellen lassen, sind auch dokumentartige Inhalte, die sich nur unter Verwendung einer speziellen Software dem Anwender erschließen, als Format denkbar. Die dokumentartigen Inhalte sind insbesondere für Software-Anwendungen, die mit dem Informationssystem direkt interagieren, sinnvoll.

2.1.2. Unterstützung konzeptueller Aspekte einer Entität

Neben der Unterstützung medialer Aspekte fließen – entsprechend Cassirers Semiotik-Verständnisses – konzeptuelle Aspekte bei der Entitätenmodellierung mit ein. Das Asset-Metamodell orientiert sich bei der Modellierung der konzeptuellen Sichtweise an den Forderungen von Peirce für Entitätenmodellierung. Unter dem Begriff *Expressiveness* (Ausdrucksstärke) versteht Peirce die Konzeptualisierung einer Entität anhand dreier Merkmalskategorien [Pei31]:

- Die erste Kategorie wird auch als die Ebene der *charakteristischen Eigenschaften* bezeichnet. Sie umfasst solche Eigenschaften, die für *jede* Entität des entsprechenden Konzeptes bestimmt werden können. Sie können wie in der UML in Form von Attributen abgelegt werden. Neben einfachen sind auch komplexe Typen für die Beschreibung zulässig.

- Die zweite Kategorie von Peirce beschreibt die *Beziehungen* zu anderen Entitäten. Sie entsprechen den Assoziationen in der UML.

- Auf der Ebene der dritten Kategorie definiert Peirce eine *Systematik*, der alle Entitäten des Konzepts folgen. Die Systematik beinhaltet eine Spezifikation des zeitlichen und örtlichen Verhaltens und definiert damit das Mengenverhalten. Diese Kategorie entspricht in der UML einer Klasse. Die Systematik beinhaltet demnach auch die Typdefinitionen der ersten und zweiten Kategorie sowie eine mögliche Initialisierung mit

Werten. Die Systematik kann zusätzlich in Form von *Constraints* (Beschränkungen) formuliert werden, die zu jedem Zeitpunkt erfüllt sein müssen.

Wird beispielsweise eine Person entsprechend der einzelnen Kategorien analysiert, fallen in die erste Kategorie einfache Eigenschaften einer Person wie Vorname, Nachname, Geburtsdatum, Augenfarbe und Größe oder komplexere Eigenschaften wie die Struktur der Retina. In der zweiten Kategorie werden Beziehungen der Person zu einem möglichen Arbeitgeber, einzelnen Arbeitspaketen, ggf. einem Ehepartner und Kindern erfasst. In der dritten Kategorie wird die Systematik der beiden anderen in Raum und Zeit definiert. Es werden dabei Aussagen über die Eigenschaften und Beziehungen der Gesamtheit aller Personen gemacht, und wie sie sich sowohl räumlich als auch zeitlich verändern können. So könnte die Systematik besagen, dass eine Person nur einen Ehepartner zur gleichen Zeit haben darf, dieser sich aber ändern darf. Eine weitere Beschränkung wäre die Aussage, dass eine Person niemals jünger werden kann.

2.1.3. Wissensmanagement und das Asset-Metamodell

In diesem Abschnitt soll eine Verbindung zwischen dem Wissensmanagement und dem vorgestellten Asset-Metamodell hergestellt werden. Dabei wird für den Bereich des Wissensmanagements auf die Ergebnisse und Definitionen von [Weg02] zurückgegriffen, wobei der Fokus auf dem Begriff Wissen und nicht auf Management liegt.

Zunächst sind die Begriffe Daten, Information und Wissen voneinander abzugrenzen. Nach [Weg02] bestehen Daten aus Werten, also zum Beispiel der Zahl 54. Durch eine zusätzliche Bedeutung wird aus einem Wert eine Information, die Zahl 54 könnte also bedeuten, dass ein Bus 54 Sitzplätze hat. Aus dieser Information wird Wissen, wenn sie zu anderen Informationen in eine Beziehung gestellt werden kann. D. h. durch die intelligente Verknüpfung von Informationen entsteht Wissen. Dieses Wissen befähigt letztendlich in den nötigen Situation zum Handeln. In dem Beispiel könnte die Information, das der Bus 54 Plätze hat, verknüpft mit der Information, das eine Reisegruppe 55 Personen umfasst, zu dem Wissen führen, dass der Bus für die Reisegruppe zu klein ist. Daraus kann sich dann entsprechendes Handeln für einen Busunternehmer ergeben, nämlich die Wahl eines größeren Busses.

Eine wesentliche Aufgabe von Anwendungssystemen zur Unterstützung des Wissensmanagements ist es, den Prozess der Verknüpfung von Informationen für den Anwender zu erleichtern. Dabei sind für den Nutzer die Ablage und Recherche von Informationen neben dem Verknüpfungsprozess besonders wichtig.

Das Asset-Metamodell als Datenmodell ist dafür sehr geeignet. Informationen können sehr gut in dokumentartige oder multimediale Inhalte einer Entitätenbeschreibung abgelegt werden. Die konzeptualisierte Sichtweise auf die Entität und damit auch auf den zugeordneten Inhalt ermöglicht den Aufbau guter Recherchemöglichkeiten. Verschiedene Ansätze für die Suche nach Inhalten werden in [Weg02] vorgestellt und diskutiert.

2.2. Anforderungen an ein Asset-Laufzeitsystem

Neben den strukturellen Eigenschaften, die bei der Entitätenmodellierung erfüllt werden müssen, hat Cassirer auch Forderungen an den Erkenntnisprozess gestellt. Diese lassen sich auf das Laufzeitsystem, das das Asset-Metamodell implementiert, übertragen. Die Forderungen von Cassirer können unter dem Begriff *responsiveness* (Ansprechempfindlichkeit) zusammengefasst werden. Sie untergliedern sich in die Forderungen nach einem dynamischen und einem offenen Anwendungssystem. Diese Eigenschaften werden in den nächsten Abschnitten erläutert.

2.2.1. Offenheit und Dynamik

Bereits in den 20er-Jahren des vergangenen Jahrhunderts forderte Cassirer, dass die Inhalt-Konzept-Paare, die er als Symbole bezeichnete und die im Asset-Metamodell durch Assets repräsentiert werden, möglichst ohne Einschränkung erzeugt werden können müssen. Dies bedeutet, dass für die Wahrnehmung und das Verständnis neuer Entitäten eine vordefinierte, strikte Hierarchie von Konzepten nicht ausreichend ist. Menschen, die von Cassirer als *animal symbolicus* bezeichnet werden, sind nur dann in der Lage, Entitäten wahrzunehmen, wenn sie ihnen Konzepte zuordnen können. Dabei werden bestehende Konzepte unter Umständen angepasst (also erweitert, verfeinert, spezialisiert, eingeschränkt usw.) oder neue erzeugt.

Diese Offenheit muss von dem Anwendungssystem unterstützt werden, d. h. die für die Modellierung einer Anwendungsdomäne verwendeten Konzepte müssen frei, jedoch im Rahmen der Peirceschen Ausdruckstärke definierbar sein.

Eine konzeptuelle Offenheit wurde bereits mit dem objektorientierten Paradigma eingeführt. Bei der objektorientierten Programmierung wird nach [AC96] versucht, eine Übereinstimmung zwischen einem physikalischen System und einem Softwaresystem zu erzeugen, das dieses physikalische System simuliert.

> „The object-oriented approach to programming is based on an intuitive correspondence between a software simulation of a physical system and the physical system itself.“

Schefe stellt jedoch klar, dass Modelle in der Softwaretechnik *Beschreibungen* und keine mathematisch-naturwissenschaftlichen Modelle sind [Sch99].

> „Für die gängige Interpretation [des softwaretechnischen Modell-Begriffs] als *Abbild* [...] ist der Begriff *Beschreibung* weniger missverständlich. Damit wird der Unterschied zu mathematischen Modellen in der Naturwissenschaft deutlicher, die – im Unterschied zu den Beschreibungen der Softwaretechnik – die Möglichkeit der *Erklärung* besitzen.“

Assetorientierte Modellierung ist speziell für die Beschreibung einer Anwendungsdomäne entwickelt worden. Im konzeptuellen Teil hat sie jedoch eine hohe Ähnlichkeit mit dem objektorientierten Ansatz. Deshalb ist ein wesentlicher Vorteil der objektorientierten Entwicklung, die Wiederverwendbarkeit von Softwarekomponenten, auf die assetorientierte Entwicklung übertragbar [SS03]. Die Technologie für ein offenes Asset-Laufzeitsystem wird

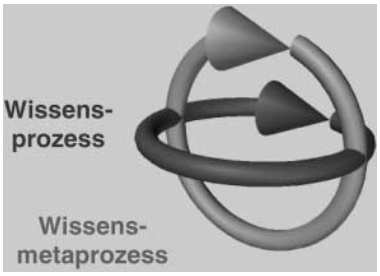


Abbildung 2.3.: Metawissensprozess und Wissensprozess nach [Sta02]

daher zu einem gewissen Teil auf den Erfahrungen des Compilerbaus für objektorientierte Sprachen aufbauen.

Neben der Offenheit stellt Cassirer die dynamische Eigenschaft des Erkenntnisprozesses heraus. Die Dynamik beinhaltet, dass zu jeder Zeit das Asset-Modell der modellierten Anwendungsdomäne erweitert oder angepasst werden können muss. Für ein konkretes Asset-Laufzeitsystem hat das zur Folge, dass die aus den Prozessmodellen der Softwaretechnik bekannten iterativen Phasen von (Re-)Analyse, (Re-)Design, (Re-)Implementierung mit einer folgenden Datenkonvertierung bei der Anpassung des Systems an den Stand des Erkenntnisprozesses entfallen. Stattdessen wird ein inkrementeller Fortschritt im Erkenntnisprozess implizit vom Laufzeitsystem unterstützt.

2.2.2. Wissensprozesse und das Asset-Laufzeitsystem

Staab fordert in [Sta02] von Informationssystemen für das „Wissensmanagement mit Ontologien und Metadaten“ im Wesentlichen die Unterstützung zweier unterschiedlicher zueinander orthogonaler Prozesse (vgl. Abbildung 2.3).

> „Die Kernidee unseres Vorschlags besteht darin, zwei aus Systemsicht zueinander orthogonale Dimensionen zu unterscheiden.
> Der *Wissensmetaprozess* umfasst alle Aspekte, die die erstmalige Erstellung der Ontologie [...] sowie ihre kontinuierliche Erweiterung und/oder Anpassung betreffen.
> Der *Wissensprozess* selbst hingegen beschreibt den Betrieb des ontologiebasierten Wissensmanagementsystems [...]. “

Dabei entspricht das Erstellen einer Ontologie der Modellierung einer Anwendungsdomäne mit Hilfe des Asset-Metamodells. Durch ein offenes und dynamisches Asset-Laufzeitsystem lässt sich außerdem die Unterstützung eines Wissensmetaprozesses realisieren.

2.3. Die Asset Sprache

Das beschriebene Asset-Metamodell wurde im Arbeitsbereich Softwaresysteme in Form einer *Asset Language* (AL) umgesetzt [SS03, Seh03], die zur konzeptuellen Modellierung und Verwaltung von Inhalten genutzt werden kann.

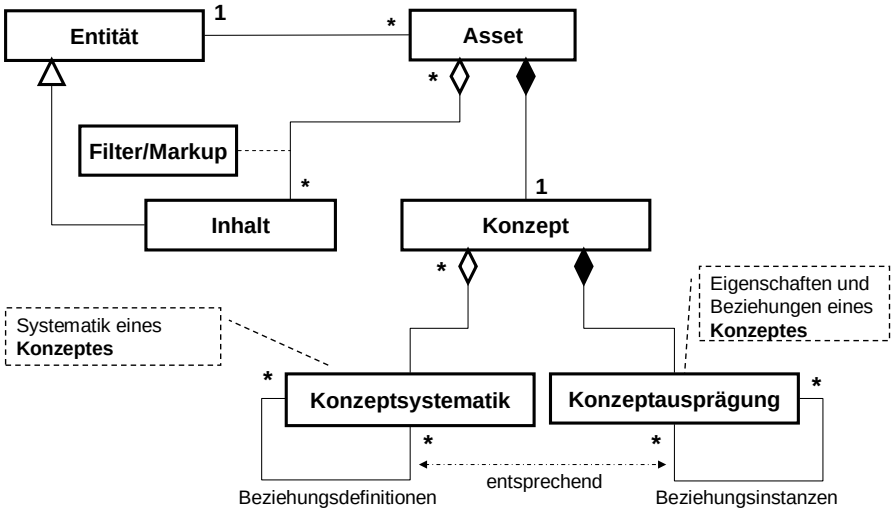


Abbildung 2.4.: Metamodell der *Asset Definition Language*

Die Sprache kann, wie es auch bei Datenbanksprachen üblich ist, in drei Teilbereiche eingeteilt werden: Definitions- (ADL), Manipulations- (AML) und Abfragesprache (AQL). Um die Interaktion mit dem Nutzer zu vereinfachen, werden Manipulationen und Abfragen über eine aus der ADL generierten graphischen Oberfläche gesteuert.

An dieser Stelle wird nochmals deutlich, dass die Forderungen Casierers nach Dynamik und Offenheit nicht durch eine Sprache selbst erfüllt werden können, sondern nur durch eine entsprechende Implementierung. Eine konzeptuelles Klassendiagramm des Asset-Metamodells ist in Abbildung 2.4 zu sehen.

`Entität` ist die zentrale Klasse des Metamodells. Unter einer Entität wird alles verstanden, wofür es einen Begriff gibt. Damit ist diese Definition allgemeiner gefasst als andere Definitionen. So wird z. B. in [DUD01] eine Entität als „Dasein im Unterschied zum Wesen eines Dinges“ definiert. Die allgemeinere Definition wurde gewählt, um neben konkreten und abstrakten Dingen auch Universale und Phänomene, wie Gefühle oder Handlungen, durch das Asset-Metamodell beschreiben zu können. An einer für Entitäten allgemein geforderten Identifizierbarkeit wird jedoch festgehalten. Diese ist entweder durch einen generierten Schlüssel oder den entsprechenden Begriff selbst gewährleistet.

Um Homonymen[2] gerecht zu werden, kann eine `Entität` durch mehrere `Assets` beschrieben werden, die jeweils eine der möglichen Bedeutungen hervorheben. Der Normalfall ist jedoch die Zuordnung eines `Assets` zu einer `Entität`.

Entitäten können einerseits medial, also inhaltlich, und andererseits konzeptionell beschrieben werden. Daher ist ein `Asset` eine Kombination von `Inhalten`, die mittels der assoziativen Klasse `Markup` zugeordnet werden, und einem zugehörigen `Konzept`. Die `Markup`-Klasse hat z. B. die Aufgabe, auf einer Landkarte eine Stadt, die durch das zugehörige `Asset` beschrieben wird, hervorzuheben. `Inhalt` ist außerdem eine von `Entität` abgeleitete Klasse. Dadurch wird eine Beschreibung des `Inhalts` in Form von `Assets` möglich.

[2] Homonyme werden umgangssprachlich auch als Teekesselchen bezeichnet.

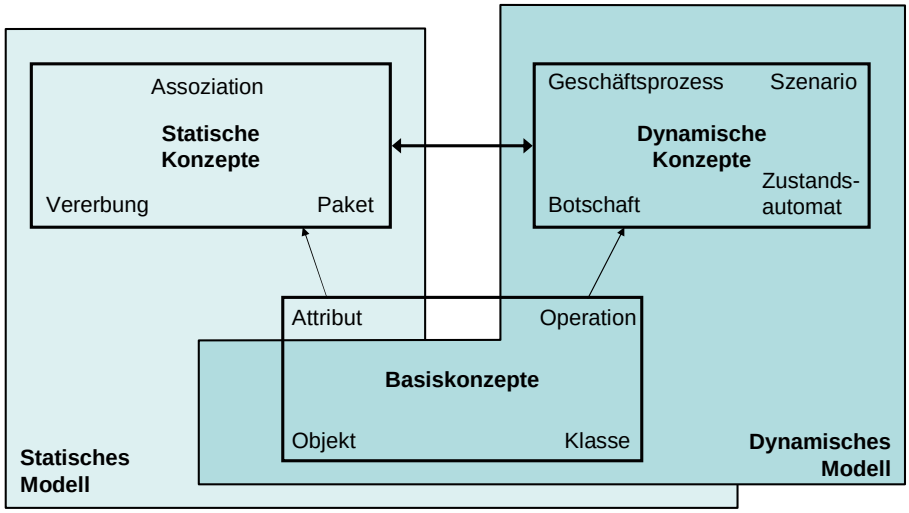


Abbildung 2.5.: Statisches und dynamisches Modell der OOA nach [Bal99]

Ein `Konzept` besteht, entsprechend der bereits erläuterten Merkmalskategorien von Peirce, einerseits aus einer `Konzeptsystematik`, die für alle Konzepte der jeweiligen Art gilt, und andererseits aus Eigenschaften und Beziehungen, die in einer `Konzeptausprägung` modelliert werden. Dabei werden mögliche Beziehungen der Konzeptausprägungen durch Assoziationen der jeweiligen Konzeptsystematiken definiert.

2.4. Assetorientierte Analyse

Die Analyse ist ein integraler Bestandteil in fast allen Prozessmodellen der Software-Technik [Bal98]. Je nach verwendetem Programmier-Paradigma und Anwendungsproblem werden unterschiedliche Methoden bei der Analyse eingesetzt. Mittlerweile hat sich in vielen Bereichen die objektorientierte Analyse (OOA) gegenüber der strukturierten Analyse (SA) durchgesetzt [Bal01]. Durch die Einführung des Asset-Metamodells ist auch eine Anpassung der Vorgehensweise bei der Analyse notwendig. Da das Asset-Metamodell – zumindest auf der konzeptuellen Seite (vgl. Abbildung 2.2) – auf objektorientierten Ansätzen beruht, wurde die assetorientierte Analyse (AOA) auf Basis der OOA entwickelt. Da die UML – vergleichbar mit der AL – selbst keine Methode vorgibt, sondern nur die Möglichkeit der Modell-Beschreibung, existieren unterschiedliche Vorgehensweisen bei der OOA, die beispielsweise innerhalb des *Rational Unified Process* (RUP) [Kru96, Kru98] oder des *Business-Oriented Software Engineering Process* (BOEP) [Oes98] definiert werden. Als Grundlage für die hier beschriebene Methode der AOA wurde der OOA-Prozess von [Bal99] verwendet, wie er in [Bal01] beschrieben wird.

Danach ist das Ergebnis der OOA eine Spezifikation des Software-Produktes. Diese Spezifikation besteht aus Konzepten der Objektorientierung und lässt sich in ein dynamisches und ein statisches Modell gliedern (vgl. Abbildung 2.5). Dies wird im Makroprozess der OOA durch die einzelnen Aufgabenbereiche berücksichtigt:

1. Analyse im Großen

2. sechs Schritte zum statischen/strukturellen Modell

3. drei Schritte zum dynamischen Modell

Die Analyse im Großen umfasst Aufgaben, die nicht spezifisch für eine objektorientierte Entwicklung sind:

1. Erstellung und Analyse von Anwendungsfällen[3]

2. Bildung von Paketen.

Beide Unterschritte sollten mit Einschränkungen in ein Vorgehensmodell für die AOA übernommen werden. Die Anwendungsfälle beschränken sich bei der AOA im Wesentlichen auf die Ablage und das Suchen von Informationen bzw. Inhalten unterschiedlicher Art. Als Akteure sind sowohl menschliche Nutzer als auch Software-Komponenten denkbar. Die Bildung von Paketen ist bei größeren Projekten mit mehreren Mitarbeitern sinnvoll, um die Bearbeitung durch Teams zu erleichtern. Obwohl die Paketbildung von der AL nicht unterstützt wird, soll auch dieser Schritt übernommen werden, da er die Gliederung der Aufgaben erleichtert.

Die nächsten beiden Schritte der OOA haben das Ziel, das statische/strukturelle und dynamische Modell zu spezifizieren. Da sich beide Schritte beeinflussen, werden sie i. Allg. parallel durchgeführt.

Das Ziel der AOA ist jedoch nur die strukturelle Modellierung der Wissensdomäne. Daraus folgt, dass auf die Schritte zur Erstellung eines dynamischen Modells verzichtet werden kann. Stattdessen ist es sinnvoll, bei der strukturellen Modellierung zwischen medialen und konzeptuellen Anteilen zu unterscheiden. Nach der Analyse im Großen sollte bei der AOA zunächst mit den Schritten zum medialen Modell begonnen werden. Diese Schritte sehen wie folgt aus:

1. Inhalte identifizieren
   Die in der Analyse im Großen erstellten Anwendungsfälle dienen als Basis für die Indentifikation von Inhalten. In diesem Schritt werden außerdem – sofern vorhanden – Muster-Instanzen der Inhalte gesammelt. Zusätzlich ist eine kurze Beschreibung der Inhalte sinnvoll. Diese Beschreibung sollte dem Vokabular der Wissensdomäne entsprechen und gemeinsam mit den Wissensträgern erarbeitet werden.

2. Inhalte spezifizieren
   Anhand der Muster-Instanzen und der Beschreibung wird eine Spezifikation der inneren Struktur festgelegt. Mögliche Werkzeuge dafür sind z. B. Grammatiken, Dokument-Typ-Definitionen oder XML-Schema-Definitionen.

Nach der medialen Modellierung wird mit der konzeptuellen Modellierung begonnen. Tatsächlich laufen jedoch beide Prozesse parallel ab und beeinflussen sich gegenseitig. Die Schritte zum konzeptuellen Modell orientieren sich an den Schritten zum statischen Modell aus [Bal01] und beinhalten folgende Aufgaben:

[3] In [Bal01] wird der Begriff Geschäftsprozess verwendet.

1. Konzepte identifizieren
   Konzepte werden wie die Klassen in der OOA identifiziert. Als Grundlage dafür dienen die gefundenen Anwendungsfälle des Makroprozesses. Neben der Zuordnung von identifizierten Inhalten zu dem Konzept ist auch eine kurze Beschreibung sinnvoll.
2. Eigenschaften identifizieren
   Es werden entsprechend der ersten Merkmalskategorie von Peirce alle charkateristischen Eigenschaften des Konzeptes identifiziert. Das schließt insbesondere Eigenschaften mit ein, die für Suchanfragen gute Diversifikationsmerkmale sind. Eine Spezifikation der Datentypen erfolgt zu einem späteren Zeitpunkt.
3. Beziehungen identifizieren
   In diesem Schritt werden entsprechend der zweiten Merkmalskategorie von Peirce die Beziehungen zwischen Konzepten identifiziert. Eine genauere Spezifikation bezüglich Navigierbarkeit und Kardinalitäten erfolgt zu einem späteren Zeitpunkt.
4. Vererbungsstruktur identifizieren
   Aufgrund der identifizierten Eigenschaften und Beziehungen lässt sich eine Vererbungsstruktur aufbauen.
5. Eigenschaften spezifizieren
   Für die ermittelten Eigenschaften werden Datentypen, Wertebereiche und Initialwerte festgelegt.
6. Beziehungen vervollständigen
   Im vorletzten Schritt werden Multiplizitäten und Navigierbarkeit der Beziehungen ergänzt.
7. *Constraints* festlegen
   Es werden entsprechend der dritten Merkmalskategorie von Peirce allgemeingültige Beschränkungen für das Konzept festgelegt.

Insgesamt besteht der Makroprozess der AOA also aus elf Schritten, die in folgende Aufgabenbereiche eingeteilt werden:

1. Analyse im Großen (zwei Schritte)
2. zwei Schritte zum medialen Modell
3. sieben Schritte zum konzeptuellen Modell

Das hier vorgestellte Vorgehensmodell der AOA ist als ein Beispiel zu verstehen. Jedoch sollte bei der Entwicklung anderer Methoden ein anwendungsfallorientiertes Vorgehen im Mittelpunkt beibehalten werden.

3. Verfahrenstechnische Grundlagen

In diesem Kapitel werden Grundlagen und Konzepte aus dem Bereich der Verfahrenstechnik vorgestellt. Dafür wird die in Kapitel 2 beschriebene Methode der assetorientierten Analyse verwendet (vgl. Abschnitt 2.4). Das Ergebnis dieser Analyse ist eine Beschreibung der Anwendungsdomäne. Die vollständige konzeptuelle Modellierung in ADL sowie eine Beschreibung der zugehörigen Inhalte sind in Anhang A und B zu finden.

Die wesentliche Aufgabe der Verfahrenstechnik ist die Entwicklung von Prozessen (Verfahren). Daher wird zunächst beschrieben was ein Prozess ist und welche prinzipiellen Aufgaben es bei der Prozessentwicklung gibt.

Da das Informationssystem als Werkzeug bei der Entwicklung von Feststoffprozessen eingesetzt werden soll, werden die relevanten Unterschiede zwischen Feststoff- und Fluidprozessen herausgestellt und die sich daraus ergebenden Konsequenzen für die Anwendung erläutert.

Sehr hilfreich bei der Erstellung dieses Kapitels waren [Ros98, Mar03] und [Rie03, Teil 1].

3.1. Prozess und Prozessentwicklung

Unter einem verfahrenstechnischen Prozess versteht man die Verkettung von physikalischen, chemischen, biologischen und informationstechnischen Operationen, um die Eigenschaften eines Stoffes gezielt zu verändern. Ein Prozess, der den Stoff entsprechend den gegebenen Voraussetzungen modifiziert, besitzt die *gewünschte Funktion*. Operationen werden durch technische Einrichtungen wie Apparate und Maschinen realisiert. In der Regel existieren für jede Operation unterschiedliche Realisierungsmöglichkeiten. Eine verfahrenstechnische Anlage ist dementsprechend die Verkettung technischer Einrichtungen und bildet „die Hülle des Prozesses“ [Mar03].

Die Aufgabe der Prozessentwicklung ist also die Bestimmung der einzelnen Operationen, ihre Verkettung zu einem Prozess und die Festlegung der technischen Realisierung jeder Operation. In der Regel werden Prozesse und Anlagen durch Fließbilder dargestellt. Daher kann bei einer Ergebnis-orientierten Sichtweise die Prozessentwicklung als die Generierung eines Fließbildes verstanden werden.

Mathematisch beschrieben handelt es sich bei einem Fließbild um einen gerichteten Graphen, der mindestens eine Quelle und eine Senke enthält. Die Knoten des Graphen stellen die verschiedenen Prozess-Schritte (Operationen/Apparate) dar und Materialströme zwischen den Prozess-Schritten werden durch Kanten repräsentiert. Häufig hat ein Prozess-Schritt mehrere eingehende und/oder ausgehende Materialströme.

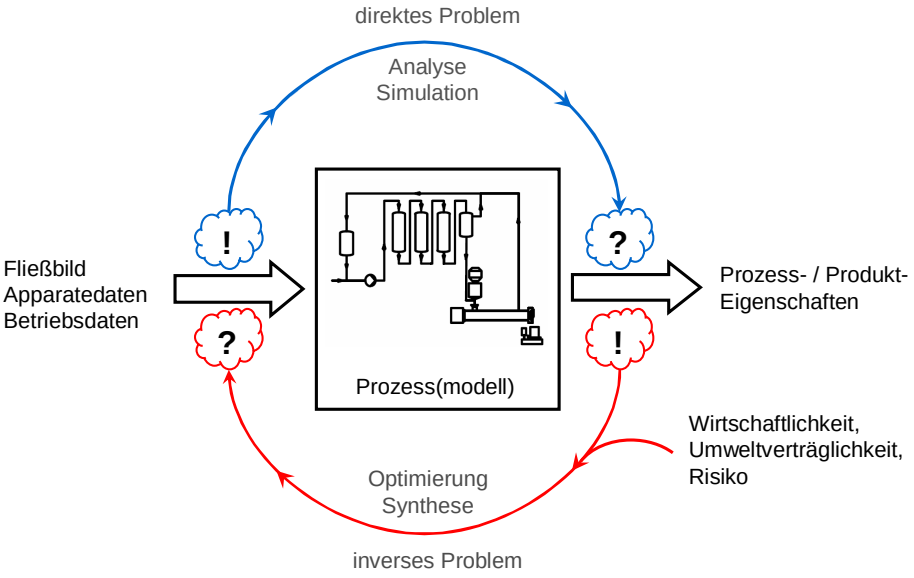


Abbildung 3.1.: Analyse und Synthese bei der Prozessentwicklung nach [Mar03]

3.1.1. Synthese und Analyse in der Prozessentwicklung

Der Entwurf eines Prozesses bzw. die Generierung eines Fließbildes ist nicht eindeutig. Es sind viele Alternativen möglich, die zwar prinzipiell das Gleiche erreichen, aber sich dennoch hinsichtlich eines Gütemaßes wie Umweltverträglichkeit, Sicherheitsrisiko und Wirtschaftlichkeit unterscheiden. Daher muss nach jedem Synthese-Schritt der Entwurf in einem Analyse-Schritt bewertet werden. Die Prozessentwicklung besteht daher immer aus Synthese- und Analyse-Schritten, die abwechselnd hintereinander ausgeführt werden, bis der Entwurf mit der festgelegten Spezifikation übereinstimmt. Sowohl bei der Analyse als auch bei der Synthese werden zunehmend mathematische Modelle verwendet, um den Prozess entweder zu simulieren oder zu optimieren. Die Simulation geht von einem Prozess-Entwurf aus und berechnet mit den Apparate- und Betriebsdaten aus den Zulaufströmen, die sich ergebenden Ablaufströme. Der Analyse-Schritt wird auch als direktes Problem bezeichnet, da das Ergebnis direkt durch Rechnen bestimmt werden kann.

Hingegen sind beim inversen Problem, der Synthese, die gewünschten Produkte nach Mengen und Qualität vorgegeben und der dazugehörige Prozess wird bestimmt. Dieser Schritt ist wesentlich schwieriger als der andere, da er nicht eindeutig ist und somit kreative Fähigkeiten vom entwerfenden Ingenieur verlangt. Neben dem Entwurf des Fließbildes zählt auch die Optimierung bzw. die Auslegung von Apparateeigenschaften eines festgelegten Fließbildes zu den Aufgaben der Prozess-Synthese. Eine graphische Darstellung dieser alternierenden Schritte ist in Abbildung 3.1 zu sehen.

Offensichtlich muss ein Informationssystem hauptsächlich die Prozess-Synthese unterstützen, d. h. es müssen Entscheidungshilfen beim Entwurf von Anlagen bereitgestellt werden. Strukturelle Entwurfsentscheidungen beruhen neben den Ergebnissen der Analyse zu einem großen Teil auf Erfahrungen aus früheren Fällen. Diese Erfahrungen werden ggf. in Form von heuristischen Regeln weitergegeben, verbleiben aber häufig in den Köpfen der Experten.

An dieser Stelle wird deutlich, dass das Informationssystem Aufgaben des Wissensmanagements erfüllen muss. Dazu gehören die Verknüpfung und Sammlung von Informationen sowie entsprechende Möglichkeiten zur Recherche (vgl. Abschnitt 2.1.3).

Neben der Prozess-Synthese bedarf aber auch die Prozess-Analyse in der *Feststoff*-Verfahrenstechnik der Unterstützung durch ein Informationssystem. Der Grund dafür liegt darin, dass die Simulationsmodelle für die Operationen der Feststoffprozesse nicht wie bei den Fluidprozessen rein analytisch sind, sondern auch empirische Komponenten enthalten (vgl. Abschnitt 3.5.1). Daher steht im Mittelpunkt des entwickelten Prototypen die Unterstützung der Analyse. Es wurde darauf geachtet, dass eine Erweiterung um eine Synthese-Unterstützung zu einem späteren Zeitpunkt möglich ist.

3.1.2. Anlagen als Komponenten-Systeme

Anlagen können wie Software-Systeme im Sinne eines Systemkonzeptes betrachtet werden. Eine Anlage ist also ein komplexes System, das aus unterschiedlichen, miteinander verknüpften Teilsystemen besteht. Diese stellen wiederum eigenständige Systeme dar. Charakterisierende Strukturmerkmale des Systemkonzeptes sind die Aggregation und Dekomposition. Während die Aggregation (oder Abstraktion) feingranulare Teilsysteme zu grobgranularen Systemen zusammenfasst, spaltet die Dekomposition (oder Detaillierung) grobgranulare Systeme in feingranulare Teilsysteme auf.

Diese Sichtweise wird beim Entwicklungsprozess ausgenutzt, um effizient nach Prozessen mit der gewünschten Funktion zu suchen. Durch die Analyse grobgranularer Systeme werden Grundlegende Prozess-Alternativen bewertet. Die dafür benötigten Daten sind wenig umfangreich und leicht zu beschaffen. Beispielhaft kann die Entscheidung, ob eine Anlage im Batch-Betrieb oder im kontinuierlichen Betrieb arbeiten soll, relativ einfach anhand des zu erwartenden Umsatzes und des Produktzykluses bestimmt werden. Durch dieses Vorgehen ergibt sich ein schnelles Ausschließen von Prozess-Alternativen (vermutlich) niedriger Güte. Dieses Ausschlussverfahren wird mit den in Betracht kommenden Prozessen auf detaillierteren Ebenen solange wiederholt, bis das System aus elementaren Teilsystemen, also letztendlich aus Grundoperationen, besteht.

Elementare Systeme können durch eine Maschine oder einen Apparat realisiert werden. Der Entwicklungsprozess verläuft jedoch nicht rein linear vom Groben zum Feinen, sondern enthält Rückschritte bei der Suche nach einer guten Lösung.

Das Systemkonzept lässt sich auch in den Fließbildern der Norm [DIN 28004][1] wiedererkennen. Das Grundfließbild ist die gröbste Darstellung eines Prozesses. Die nächste feinere Stufe wird durch Verfahrensfließbilder dargestellt. Der technischen Umsetzung bereits sehr nahe sind Rohrleitungs- und Instrumentenfließbilder (vgl. Abbildung 3.2).

3.2. Feststoffprozesse

Feststoffprozesse sind verfahrenstechnische Prozesse, die Feststoffe gezielt verändern. Feststoffprozesse werden der *mechanischen* Verfahrenstechnik zugeordnet, da die Funktion der

[1] Die DIN 28004 ist mittlerweile veraltet und durch die [EN ISO 10628] ersetzt worden.

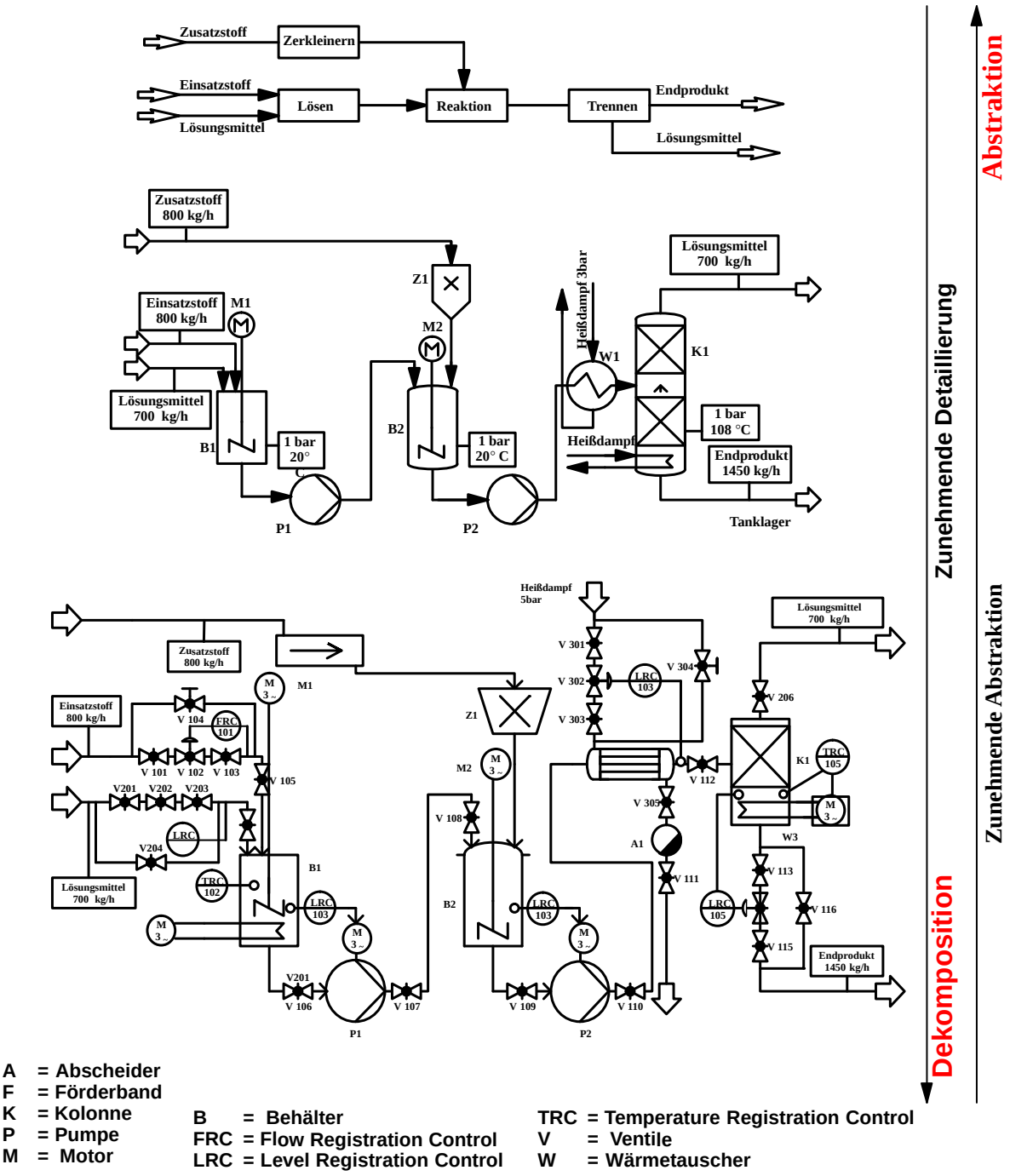


Abbildung 3.2.: Hierarchische Prozess- und Anlagendarstellung von Fließbildern der [DIN 28004] (von oben nach unten): Grundfließbild, Verfahrensfließbild und Rohrleitungs- und Instrumentenfließbild. Die Bilder stammen aus [Mar03].

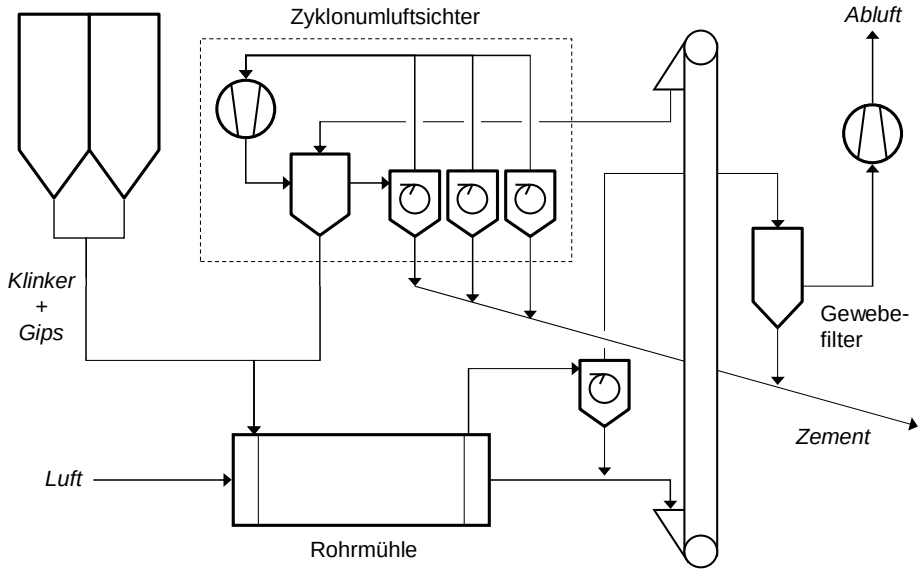


Abbildung 3.3.: Fließbild einer Mahlanlage nach [Ros98]

verwendeten Apparate auf meist mechanischen Prinzipien beruht.

Typische Feststoffprozesse findet man z. B. bei der Aufbereitung von Rohkies oder der Zementproduktion. Die in diesem Kapitel verwendeten Muster-Instanzen stammen daher zum größten Teil aus diesen Anwendungsbereichen und wurden zwei Fallstudien aus [Ros98] entnommen. Als Beispiel für einen Feststoffprozess ist in Abbildung 3.3 das Verfahrensfließbild einer Mahlanlage für die Herstellung von Zement zu sehen, das im Folgenden erläutert wird.

Die Hauptkomponenten der Mahlanlage sind eine Luftstrom-Rohrmühle, ein Zyklonumluftsichter, ein Abscheidezyklon und ein Gewebefilter. Die Edukte Klinker und Gips gelangen zusammen mit dem Rückführstrom aus der Klassierung (Zyklonumluftsichter) in die Rohrmühle. Unter Klassierung wird das Trennen von Material anhand der Partikelgröße verstanden.

Neben dem Sieben ist Sichten eine spezielle Form des Klassierens. Beim Sichten werden kleine, leichte Partikel durch einen Luftstrom ausgetragen. Dieses Prinzip ist die Grundlage für einen Zyklon und wird z. B. auch bei der Trennung von Spreu und Weizen angewandt.

Im nächsten Schritt wird das Gips-Klinker-Gemisch von einer Rohrmühle zerkleinert. Aufgrund der gewünschten sehr kleinen Partikelgröße des Endproduktes und der Härte des Materials wird ein Großteil der aufgewandten Energie in Wärme umgesetzt. Da bei einer Überhitzung des Gipses jedoch Kristallwasser austritt und damit die gewünschte Anwendungseigenschaft des Gipses als Abbindeverzögerer verloren geht, wird das Material in der Mühle durch einen Luftstrom gekühlt.

Die Kühlung hat den ungewollten Nebeneffekt, dass ein Teil des Materials vom Luftstrom mit ausgetragen wird. Aus Gründen der Luftreinhaltung und auch aus wirtschaftlichen Gründen muss daher das Material aus dem Luftstrom zurückgewonnen werden. Dies geschieht in einem zweistufigen Prozess. Zunächst werden grobe Anteile durch einen Zyklon,

und dann die restlichen, sehr feinen Partikel durch einen Gewebefilter vom Luftstrom getrennt. Das grobe Material des Abscheidezyklons wird zusammen mit dem Mahlgut der Rohrmühle in den Zyklonumluftsichter geführt. Der Zyklonumluftsichter besteht wiederum aus einzelnen Teilsystemen, auf die hier nicht eingegangen werden soll. Da das durch den Gewebefilter gewonnene Material bereits den Produktanforderungen entspricht, wird es dem Ablaufstrom der Anlage zugeführt. Der Zyklonumluftsichter trennt den bereits fertigen Zement von den noch zu groben Material-Anteilen ab und führt entsprechend die feine Fraktion dem Endprodukt und die grobe Fraktion der Rohrmühle zu. Der beschriebene Prozess zeichnet sich durch eine innere Rückführung im Zyklonumluftsichter und eine äußere Rückführung aus.

3.2.1. Simulation von Feststoffprozessen

Um bereits vor der Konstruktion Aussagen über Kosten, Leistungsfähigkeit sowie über die Eigenschaften des Produktes machen zu können, werden verfahrenstechnische Prozesse simuliert. Dabei werden prinzipiell die beiden folgenden Verfahren zur Simulation unterschieden.

Gleichungsorientierte Simulation

Bei der gleichungsorientierten Simulation wird der gesamte Prozess durch ein System von Differentialgleichungen modelliert. Das Ergebnis der Simulation ist die meist numerische Lösung der Differentialgleichungen. Dieses Verfahren wird insbesondere bei der Simulation von Fluidprozessen angewendet und ist in dem genannten Anwendungsbereich schon etliche Jahre kommerziell verfügbar (z. B. von der Firma ASPENTECH). Ein Vorteil dieser Simulationstechnik ist die Eigenschaft, dass Aussagen über die Dynamik des Systems gemacht werden können. Dies schließt insbesondere mit ein, ob ein Gleichgewichtszustand des Systems erreicht werden kann oder ob es instabil ist. Problematisch bei diesem Verfahren ist jedoch die Aufstellung des Gleichungssystems sowie der zur Lösung benötigte Rechenaufwand. Bei Fluidprozessen wurde dies in den Griff bekommen, da zum einen die einzelnen Reaktionsschritte unabhängig vom Materialstrom durch Gleichungen modelliert werden können und zum anderen die Eigenschaften des Materialstroms durch wenige physikalische Parameter und Stoffkonzentrationen beschrieben werden können. So gilt für Gase, bei nicht zu hohem Druck, die thermische Zustandsgleichung:

$$p \cdot m \cdot V = R \cdot T$$

Chemische Reaktionen können durch Reaktionsgleichungen und physikalische bzw. mechanische Eigenschaften durch wenige Materialkonstanten (z. B. Oberflächenspannungskoeffizient bei Flüssigkeiten) beschrieben werden .

Blockorientierte Simulation

Die blockorientierte Simulation oder auch Fießschemasimulation wird bei sehr komplexen Fluidprozessen oder bei Feststoffprozessen angewendet. Häufig werden bei der Fließschemasimulation die Ströme des Gleichgewichtszustandes ermittelt. Durch die zusätzliche Berücksichtigung von Zeitintervallen während der Simulation lassen sich jedoch auch Aussagen über dynamische Eigenschaften des Systems machen. Ausgehend von einem Fließschema-Diagramm wird, beginnend bei der Material-Quelle dem Materialfluss folgend, für jeden Apparat der Ablaufstrom berechnet. Dabei ist der Ablaufstrom eines Apparats gleichzeitig der Zulaufstrom eines anderen Apparats oder einer Material-Senke. Selbstverständlich sind je Apparat mehrere Zulauf- und Ablaufströme sowie mehrere Material-Quellen und -Senken im Prozessaufbau möglich. Aufgrund von Rückführungen innerhalb des Prozessaufbaus (vgl. Abbildung 3.4) müssen Berechnungen der Materialströme so lange durchgeführt werden, bis entweder ein Gleichgewichtszustand oder eine vorher angegebene maximale Anzahl von Iterationen erreicht wird. Neben der Berechnung der Materialströme sind natürlich auch die zu erwartenden Kosten (z. B. Energie, Wasser) des Prozesses interessant. Die Kosten können aus dem ermittelten Gleichgewichtszustand berechnet werden.

3.3. Materialstrom

Wie bereits im vorherigen Abschnitt deutlich geworden ist, ist es wesentlich aufwändiger, einen Feststoff zu beschreiben als ein Fluid. Fluide bestehen aus Fraktionen *gleicher* Teilchen, d. h. eine Fraktion wird durch einige unterschiedliche Moleküle exemplarisch vollständig repräsentiert. Feststoffe hingegen bestehen aus Fraktionen *ähnlicher* Teilchen und daher reicht es nicht aus, nur einen Partikel je Material-Fraktion zu beschreiben. Offensichtlich ist jedoch die Beschreibung jedes einzelnen Partikels nicht möglich.

Stattdessen wird das Partikelsystem in einer Weise beschrieben, die sowohl der Ähnlichkeit als auch den Unterschieden von Partikeln gerecht wird. Synonym zu dem Begriff Partikelsystem werden häufig auch die Begriffe disperses System und Partikelkollektiv benutzt.

Ein disperses System besteht aus Partikeln, die sich gegenseitig, verursacht durch Fluss- und Kraftfelder, beeinflussen können. Die einzelnen Partikel können durch verschiedene Eigenschaften voneinander abgegrenzt werden. Das wohl am häufigsten benutzte Klassifikationsmerkmal ist dabei die Partikelgröße. Die minimale Partikelgröße liegt im Bereich der Partikeltechnologie zwischen 10 nm (Aerosole, Kondensationsprodukte und Fällungsprodukte) und 100 nm (minimale Partikelgröße durch mechanische Zerkleinerung).

Neben den dispersen Phasen enthält ein Materialstrom häufig auch kontinuierliche Phasen. Prinzipiell sind bis auf gasförmig–gasförmig alle möglichen Kombinationen der klassischen Aggregatszustände (fest, flüssig, gasförmig) bei der kontinuierlichen und dispersen Phase möglich. Alltägliche Beispiele für Kombinationen ohne feste disperse Phase sind zum Beispiel Nebel (gasförmige kontinuierliche Phase und flüssige disperse Phase) und Schaum (flüssige kontinuierliche Phase und gasförmige disperse Phase). Im Bereich der Partikeltechnik werden im Wesentlichen Materialströme untersucht, die eine oder mehrere feste disperse Phasen und, falls vorhanden, fluide (flüssige oder gasförmige) kontinuierliche Phasen enthalten. Diese Kombinationen sind in der Tabelle 3.1 hervorgehoben.

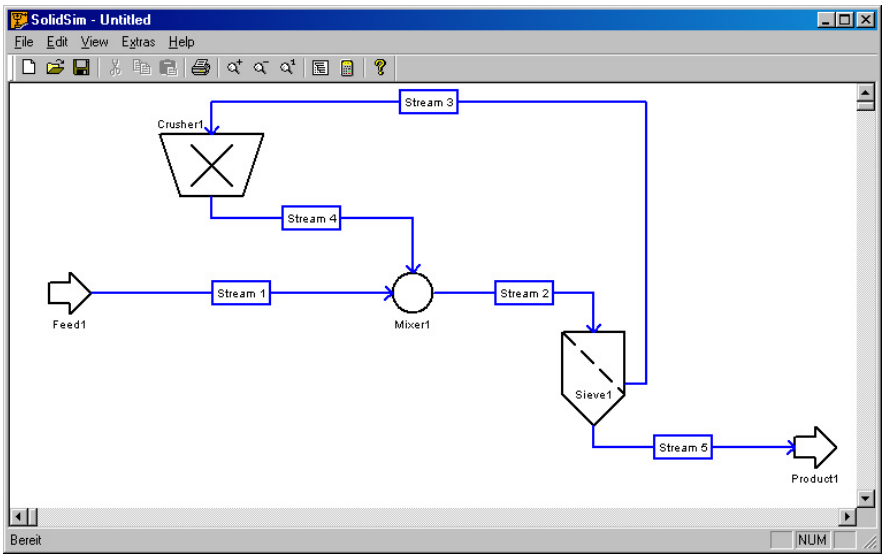


Abbildung 3.4.: Simulation eines Prozesses zur Zerkleinerung von Kies: Die Abbildung zeigt einen *Screenshot* der Simulations-Software SOLIDSIM, die im Arbeitsbereich VT1 der Technischen Universität Hamburg-Harburg entwickelt wird. Dargestellt ist ein Prozess für die Zerkleinerung von Kies. Der Kies wird durch einen Brecher (*crusher*) zerkleinert. Anschließend wird der zu grobe Anteil vom gebrochenen Kies durch ein Sieb (*sieve*) abgetrennt und wieder in den Brecher zurückgeführt. Dieses Verfahren gewährleistet einen geringen Überkornanteil im fertigen Produkt. Als Über-/Unterkornanteil bezeichnet man den Anteil der Partikel, die entsprechend der Spezifikation eine zu große/kleine Partikelgröße haben.

Tabelle 3.1.: Aggregatszustände von kontinuierlicher und disperser Phase: Die Tabelle zeigt die möglichen Kombinationen der Aggregatszustände einer dispersen und einer kontinuierlichen Phase. Für die Simulation von Feststoffprozessen sind die Kombinationen mit hervorgehobenen Zellen von Bedeutung, da dort die kontinuierliche Phase Fluideigenschaften hat. Für die Kombination von fester kontinuierlicher und flüssiger disperser Phase existiert kein allgemeiner Begriff und eine Kombination von gasförmiger disperser und gasförmiger kontinuierlicher Phase existiert nicht.

kontinuierliche Phase / **disperse Phase**	**fest**	**flüssig**	**gasförmig**
fest	granulare Struktur	Suspensionen Schlamm	Aerosole Staub Rauch
flüssig		Emulsionen	flüssige Aerosole Nebel
gasförmig	fester Schaum / durchlüfteter Kunststoff	Blasen Schaum	---

3.3.1. Partikelmerkmale

Das Verhalten von Partikeln innerhalb eines Prozesses hängt im Wesentlichen von der Partikelgröße, der Partikelform und der Oberflächenstruktur ab. Für die Simulation von Feststoffprozessen sind daher angemessene, d. h. leicht zu handhabende und dennoch aussagekräftige Beschreibungen dieser Eigenschaften nötig. Es können geometrische und physikalische Eigenschaften der Partikel unterschieden werden.

Geometrische Partikelmerkmale

Alle Größen- oder Formmerkmale eines Partikels sind geometrische Merkmale. Typische Größenmerkmale sind:

- Volumen eines Partikels
- Oberfläche eines Partikels
- Oberfläche der Projektion eines Partikels
- Längen unterschiedlich definierter Strecken (z. B. Durchmesser oder spezielle Sehnen nicht sphärischer Partikel)

Größenmerkmale können relativ einfach durch einen Zahlenwert angegeben werden. Im Gegensatz dazu ist eine Beschreibung der Partikelform nicht ohne weiteres möglich. Gegen eine sprachliche Beschreibung der Form durch Begriffe wie z. B. „rund“ und „länglich“

spricht die offensichtliche Subjektivität und Ungenauigkeit. Jedoch sind Methoden wie die Beschreibung der Form durch eine Fourier-Analyse oder fraktale Geometrie zu aufwändig. Daher wird die Partikelform im Normalfall nicht exakt, sondern durch einen Formfaktor beschrieben, der die Abweichung von einer sphärischen Form wiedergibt.

Physikalische Partikelmerkmale

Physikalische Eigenschaften beschreiben das Verhalten von Partikeln. Einige dieser Eigenschaften sind:

- Sedimentations-Geschwindigkeit
- Diffusions-Geschwindigkeit
- elektrische Ladung
- Lichtstreuung

Die physikalischen Partikeleigenschaften sind von den geometrischen Partikeleigenschaften sowie anderen Parametern, häufig materialspezifischen Eigenschaften, wie z. B. Dichte, Elastizität und Brechungsindex, abhängig.

3.3.2. Beschreibung eines Materialstroms

Für die Beschreibung eines Materialstroms gelten die gleichen Forderungen wie für die Beschreibung eines einzelnen Partikels. Neben einer hohen Aussagekraft wird eine gute Handhabe sowohl bei der Messung der Eigenschaften als auch bei der späteren Simulation gefordert. Die Beschreibung der physikalischen und geometrischen Eigenschaften eines jeden Partikels ist zwar sehr exakt, jedoch faktisch messtechnisch und datenverarbeitungstechnisch nicht durchführbar. Allerdings ist die Angabe von einfachen Mittelwerten oder von Wertebereichen der verschiedenen Eigenschaften für das gesamte Partikelkollektiv nicht exakt genug. Daher werden ähnliche Partikel des Materialstroms in einzelne Zellen zusammengefasst und durch ihre jeweiligen Wertebereiche oder Mittelwerte in den unterschiedlichen Eigenschafts-Dimensionen beschrieben.

Da Partikel durch multidimensionale Daten beschrieben werden, ist eine grafische Darstellung schwierig. Daher werden häufig nur die Dimensionen visualisiert, die einen speziellen, vom Ersteller der Grafik ausgewählten Zusammenhang verdeutlichen. Abbildung 3.5 zeigt die Verteilung der Partikelgröße und den jeweiligen Massenanteil organischer Substanzen eines typischen Rohkieses aus Westdeutschland.

Neben verteilten Eigenschaften, für die Beschreibung der dispersen Phase, besitzt ein Materialstrom aber auch kontinuierliche Eigenschaften. Diese können für den Gesamtstrom oder für disperse und fluide Teilströme gelten. Zusätzlich werden implizit extensive und intensive Eigenschaften unterschieden. Als intensive Stoffgrößen werden verfahrenstechnische Größen verstanden, die unabhängig von der Stoffmenge sind, wie z. B. die Temperatur. Dies bedeutet nicht zwangsläufig, dass die Temperatur gleichverteilt ist. Unter extensiven Stoffgrößen versteht man dementsprechend Größen, die abhängig von der Stoffmenge sind, wie z. B. die Masse oder das Volumen. Bei gedachter Teilung eines (homogenen) Brottteiges hat

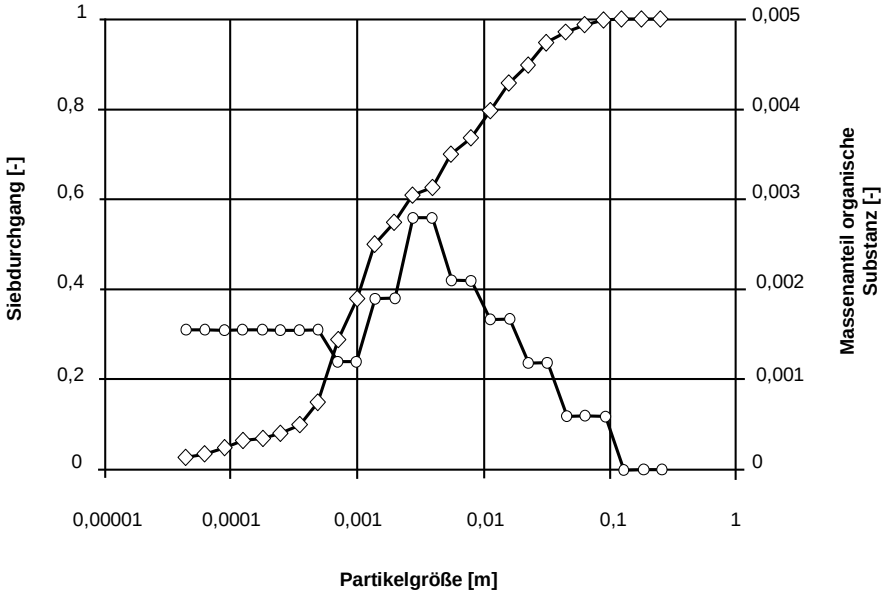


Abbildung 3.5.: Westdeutscher Rohkies: Partikelgrößenverteilung und organische Anteile, nach [Ros98].

sich die Temperatur einer Hälfte gegenüber der Gesamtmenge nicht verändert, jedoch sind Masse und Volumen nur noch halb so groß.

Fraktionsbaum

Die verteilten und von einander abhängigen Eigenschaften der dispersen Phase lassen sich durch einen gerichteten Baum darstellen [GRWT95]. Motiviert ist diese Darstellung durch die experimentelle und iterative Ermittlung der Eigenschaften. Im Experiment wird zunächst die statistische Verteilung einer Eigenschaft des Materials ermittelt (z. B. die Partikelgröße). Genau genommen wird nicht die statistische Verteilung, sondern ein Histogramm ermittelt. D. h. es wird der Wertebereich der verteilten Größe in Klassen diskretisiert (z. B. Intervalle) und der prozentuale Anteil der einzelnen Klassen am Materialstrom bestimmt. Das Ergebnis des Vorgangs ist neben der Bestimmung der einzelnen Anteile auch eine den Klassen entsprechende Aufteilung der Materialprobe. Diese Fraktionen werden nun iterativ anhand der Verteilung anderer Eigenschaften in weitere Unterfraktionen zerlegt. Die sich daraus ergebende Struktur kann man als einen Baum auffassen, bei dem jeder Kind-Knoten einen fraktionellen Anteil am Vater-Knoten repräsentiert. Solch ein Baum ist in Abbildung 3.6 zu sehen.

Hyperwürfel

Alternativ zu einem Fraktionsbaum kann die disperse Phase durch einen OLAP-Hyperwürfel dargestellt werden (vgl. Abbildung 3.7). Die Dimensionen des Würfels entsprechen der Anzahl der beschriebenen Eigenschaften. Jede Zelle des Hyperwürfels symbolisiert dabei einen Anteil der dispersen Phase mit festgelegten Eigenschaften. Aus den Koordinaten der

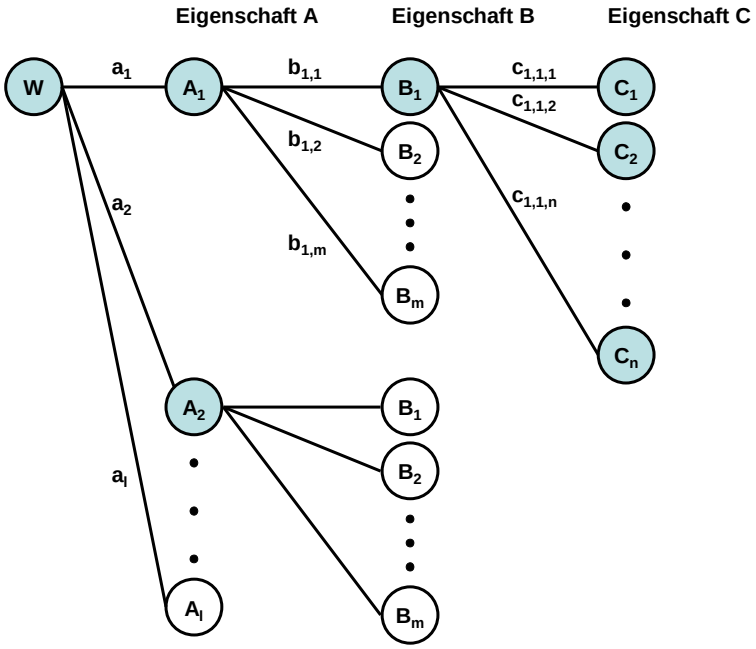


Abbildung 3.6.: Beispiel eines Fraktionsbaums: Die Abbildung verdeutlicht die allgemeine Struktur eines Fraktionsbaums. Die Wurzel W des Baums repräsentiert die Gesamtheit der dispersen Phase. Diese ist in die Klassen $A_1, \ldots, A_l$ fraktioniert und die Kanten sind mit dem prozentualen Anteil $a_1, \ldots, a_l$ der jeweiligen Klasse beschriftet. Für die anderen Knoten des Baums gilt das Entsprechende.

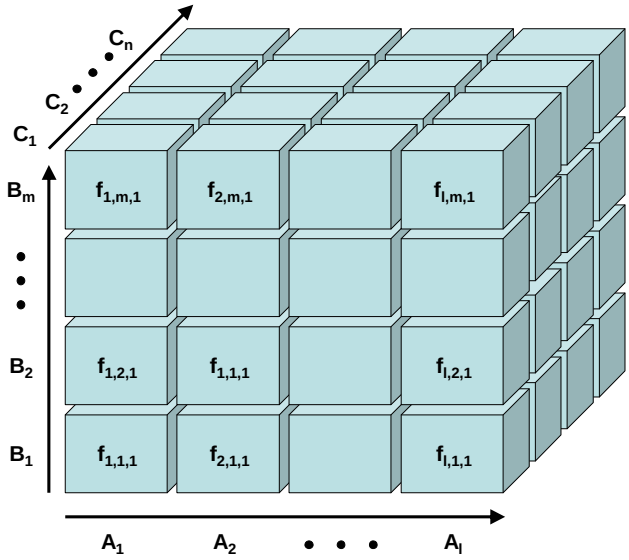


Abbildung 3.7.: Modellierung der dispersen Phase durch einen Hyperwürfel: Dieser Hyperwürfel entspricht dem Fraktionsbaum aus Abbildung 3.6. Die Indizes der einzelnen Zellen entsprechen den Klassen der einzelnen Eigenschafts-Dimensionen. Die $f_{g,h,i}$ in den einzelnen Zellen geben den fraktionellen Anteil der Zelle am Gesamtstrom der dispersen Phase an.

Zelle lassen sich die Eigenschaften in den unterschiedlichen Dimensionen ablesen. Der prozentuale Anteil an der dispersen Phase, der in den Zellen als Kennzahl abgelegt ist, kann aus dem ursprünglichen Fraktionsbaum ermittelt werden. Dabei gilt: $f_{g,h,i} = a_g \cdot b_{g,h} \cdot c_{g,h,i}$. Diese Darstellung der dispersen Phase wird innerhalb der Simulationssoftware SOLIDSIM verwendet. Die Würfel-Darstellung bietet außerdem durch die bekannten OLAP-Operationen (*slice*, *dice*, *drill down*, *roll up*, usw.) die Möglichkeit, analytische Fragestellungen zu beantworten.

Um die Äquivalenz beider Modelle zu verdeutlichen, wird in Abbildung 3.8 ein Muster-Material durch einen Fraktionsbaum und einen gleichwertigen Würfel dargestellt. Partikel des Materials können sich in der Form (rund oder eckig) und der Farbe (hell oder dunkel) unterschieden. Dementsprechend hat der Würfel zwei Dimensionen, die jeweils zwei unterschiedliche Werte umfassen und der Fraktionsbaum hat vier Blätter (zwei mal zwei). Die Berechnung der einzelnen Anteile erfolgte nach den Beschreibungen der vorherigen Abschnitte.

3.3.3. Mediales und konzeptuelles Modell eines Materialstroms

Materialströme mit dispersen Phasen (multidimensionale Daten) werden durch recht komplexe Strukturen modelliert. Diese Strukturen erschließen sich in der Regel nicht durch einen menschlichen Nutzer, sondern bekommen erst durch eine Spezialanwendung, wie eine Simulations- oder eine Statistik-Software, einen Nutzwert für den Anwender.

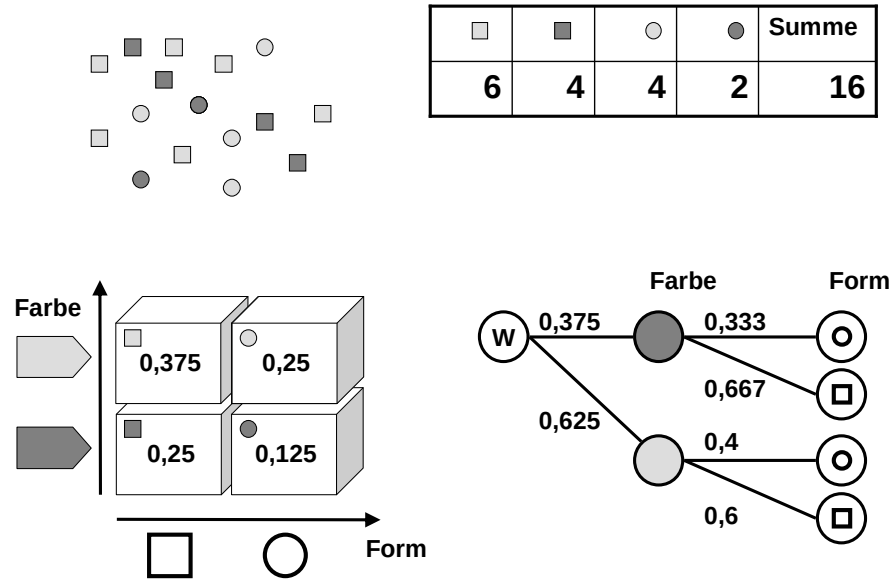


Abbildung 3.8.: Fraktionsbaum und Hyperwürfel eines Muster-Materials

Daher wurde die gesamte Beschreibung eines Materialstroms auf ein XML-Schema abgebildet (vgl. Anhang B). Auf konzeptueller Ebene wurden nur die Eigenschaften `Name` und `Beschreibung` realisiert, um eine textbasierte Recherche zu ermöglichen (vgl. Anhang A, Zeilen 34 - 45). Umfassendere Recherche-Möglichkeiten, die z. B. statistische Verteilungen einzelner Eigenschaften berücksichtigen, wurden wegen des zu erwartenden hohen Zeitaufwandes und des schwierigen Fachkonzeptes nicht realisiert. Durch eine offene und dynamische Systemarchitektur wird jedoch eine spätere Ergänzung dieser Suchfunktionen unterstützt.

3.4. Apparate

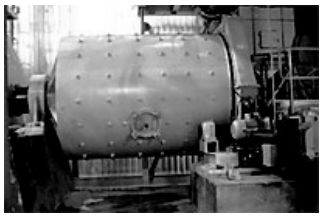
Apparate sind verfahrenstechnische Maschinen, die Materialströme verändern, vereinigen oder trennen. Sie können einerseits anhand der *Technik der Verfahrens* eingeordnet werden also biologische, chemische oder mechanische Verfahrenstechnik und andererseits anhand des verarbeiteten Materials also Feststoffe (i. Allg. disperse Systeme) und Gase bzw. Flüssigkeiten (i. Allg. fluide Systeme). Im Rahmen der Diplomarbeit stehen insbesondere Apparate der mechanischen Verfahrenstechnik für Feststoffprozesse im Mittelpunkt. Es darf jedoch nicht vernachlässigt werden, dass Feststoffprozesse neben dispersen auch fluide Phasen enthalten. Einen kleinen Überblick über Apparate der mechanischen Verfahrenstechnik bietet Abbildung 3.9. Den einzelnen Apparaten können elementare Grundoperationen zugeordnet werden:

- Taumelsiebmaschine – Trennen
- Recyclingbrecher – Zerkleinern
- Staubabscheider – Trennen

Taumelsiebmaschine [Fa. Allgaier]

Recyclingbrecher [Fa. Beyer]

Staubabscheider [Fa. Stela]



Kugelmühle [Fa. FHZ]

Abbildung 3.9.: Muster-Apparate der mechanischen Verfahrenstechnik

- Kugelmühle – Zerkleinern

Eine Übersicht über die verschiedenen Operationen in der mechanischen Verfahrenstechnik bietet das *Rumpfsche Ordnungsschema der Partikeltechnik* (vgl. Abbildung 3.10).

Das über Apparate vermittelte Wissen (im Studium, in Fachbüchern) ist Konzeptwissen. D. h. es wird nicht die Funktionsweise eines konkreten, real existierenden Apparats beschrieben oder vermittelt, sondern die eines generalisierten Apparats. Ein Apparat wird neben dem Funktionsprinzip, das durch mathematische Modelle beschrieben werden kann, durch seine Eigenschaften charakterisiert. Diese Eigenschaften müssen nicht unbedingt Einstellungen, sondern können auch funktionsbeeinflussende Konstruktionsgrößen sein. Aus Ingenieurssicht kann also zwischen Konstruktions- und Betriebsparametern unterschieden werden. Konstruktionsparameter werden während des Entwurfs festgelegt und heißen daher auch Entwurfsparameter. Nach der Konstruktion des Apparats können sie im Gegensatz zu den Betriebsparametern nicht mehr verändert werden. Aus Informatikersicht ist eine Trennung beider Parametertypen nicht notwendig, da beide Parameterarten i. Allg. zum Simulationszeitpunkt veränderbar sind.

3.5. Modelle

Während des Entwicklungsprozesses werden verschiedene mathematische Modelle eingesetzt. Dimensionierungsmodelle dienen während des Synthese-Schrittes dazu, aus erwarteten Prozessdaten wie dem Materialfluss die Konstruktionsgrößen des Apparats grob festzulegen. Während der weiteren Analyse- und Synthese-Schritte werden die festgelegten Größen verfeinert. Um die Güte eines Prozesses hinsichtlich der Energiekosten zu bewerten, werden Energiemodelle eingesetzt. Im Analyse-Schritt werden in erster Linie Simulationsmodelle für die Berechnung der Prozess- und Produkteigenschaften verwendet.

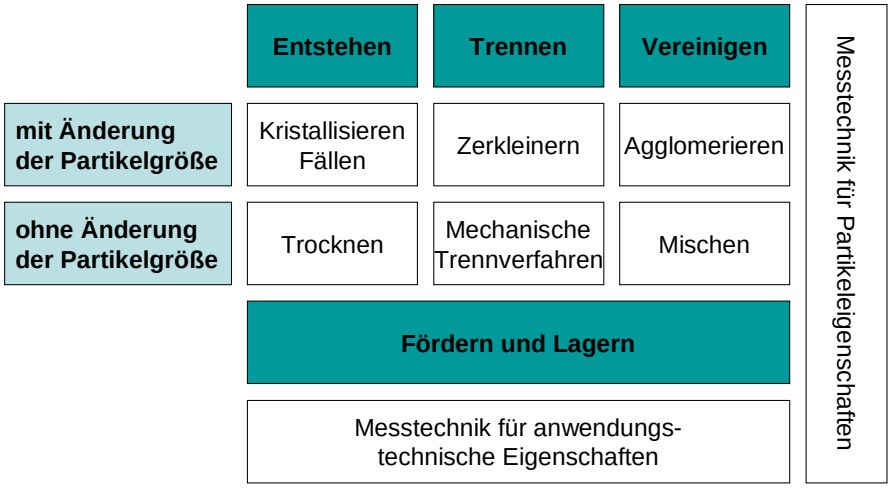


Abbildung 3.10.: Ordnungsschema der Feststoffprozesse

Die Gemeinsamkeit aller Modelle ist, dass sie auf mathematischen Formeln beruhen. Mit Hilfe der Formeln werden auf Basis der Eingabewerte die Ausgabewerte bestimmt. Textuelle Modellbeschreibungen müssen daher mathematische Formeln bzw. Rechenvorschriften, z. B. durch die MathML [Wor03a], unterstützen. Die Besonderheiten der einzelnen Modellarten werden in den folgenden Abschnitten erläutert. Die Modelle eines Kegelbrechers aus [Ros98] sind in Anhang C zu finden.

3.5.1. Simulationsmodelle

Die besonderen Eigenschaften der Feststoffe erschweren die mathematische Modellierung der einzelnen Apparateklassen. Daher existiert für nahezu jede Grundoperation Ansätze mit unterschiedlicher Modellierungstiefe. Sie reichen von einfachen *Shortcut*-Modellen zu rigoroseren Ansätzen. Eine Übersicht ist in Abbildung 3.11 zu sehen. Zunächst können die Modelle anhand des Detaillierungsgrades, in dem die den Dispersitätszustand kennzeichnenden Feststoffverteilungen erfasst und berücksichtigt werden, in *Shortcut*-Modelle und Fraktionsbilanz-Modelle aufgeteilt werden. *Shortcut*-Modelle beschreiben statt der differenziert betrachteten Änderung der Fraktionsanteile lediglich die Änderung eines oder mehrerer Verteilungsrepräsentanten in einem Prozess-Schritt. Mit dem berechneten Verteilungsrepräsentanten muss im Weiteren eine vollständige Verteilung der dispersen Phase erzeugt werden. Im einfachsten Fall wird die Annahme getroffen, dass sich die Kontur der neuen Verteilung nicht von der alten unterscheidet, d. h. mit der Veränderung des Repräsentanten wird die Verteilung lediglich um diese Differenz verschoben.

Die rigorosen Modelle umfassen hingegen die detaillierte Beschreibung der Änderung vollständiger Verteilungen mit Hilfe von Fraktionsbilanz-Modellen[2]. Obwohl sich die Bilanzgleichungssysteme für die einzelnen Modellgruppen allgemein formulieren lassen, sind für

[2]im Zusammenhang mit Zerkleinerungs- und Agglomerationsprozessen auch als Populationsbilanz-Modelle bezeichnet

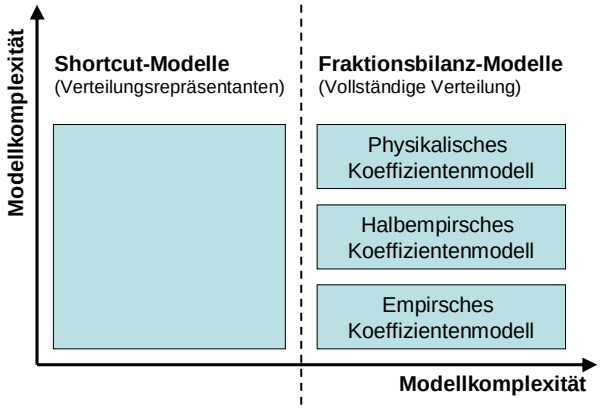


Abbildung 3.11.: Mehrstufige Modellhierarchie

eine sinnvolle Nutzung im Rahmen der Fließschemasimulation entsprechende apparatespezifische Koeffizientenmodelle erforderlich. Koeffizientenmodell ermöglichen eine Berechnung der in das Fraktionsbilanzmodell eingehenden Koeffizienten für einen konkreten Anwendungsfall. Dabei werden nur Dispersitätseigenschaften berücksichtigt, die für den eigentlichen mechanischen Vorgang eine besondere Relevanz besitzen. Beispielhaft ist die Korngröße (Partikeldurchmesser) für das Sieben eine solche Eigenschaft. Allgemein gilt, dass der Modellierungsgrad eines Modells umso höher ist, je stärker einflussnehmende Mikroprozesse bei der Modellierung der Makroprozesse berücksichtigt werden. Dadurch nimmt in der Regel nicht nur die Kompliziertheit des mathematischen Aufbaus zu, sondern auch die Anzahl der eingehenden, häufig unbekannten Parameter.

Die in die Modelle von Feststoffen eingehenden Parameter sind dabei nur in Sonderfällen reine Stoffgrößen. Dies liegt im Wesentlichen daran, dass bei Feststoffprozessen nicht wie bei molekular-dispersen Fluiden (disperses System auf der Ebene der Moleküle) auf die Ebene gleichartiger, homogener Moleküle zurückgeschlossen werden kann. Häufig sind diese Parameter für eine Prozessklasse wertmäßig abhängig vom Apparatetyp, der Bauklasse, der jeweiligen Abmessung und der Stoffart. Es müssen daher bei der Spezifizierung der Modellgleichungen von Feststoffprozessen vielseitige apparative und stoffliche Aspekte berücksichtigt werden. Die Koeffizientenmodelle können in drei Gruppen unterschiedlicher Modellierungstiefe eingeteilt werden:

- *Empirische Koeffizientenmodelle* stellen einen funktionalen Zusammenhang zwischen Koeffizienten und den prozessbestimmenden Dispersitätsgrößen her. Die eingehenden Parameter werden für konkrete Anwendungsfälle durch eine Parameteranpassung empirisch ermittelt. Das bedeutet, dass die Parameter strenggenommen nur für den konkreten Anwendungsfall Gültigkeit besitzen und daher bei einer Extrapolation der Daten auf abweichende Betriebsbedingungen überprüft werden müssen.

- *Halbempirische Koeffizientenmodelle* berücksichtigen zusätzlich physikalische Ansätze und beinhalten im Wesentlichen die Bereitstellung von funktionalen Zusammenhängen zur Variation der wesentlichen Betriebs- und Entwurfsparameter eines Apparats. Ausgehend von einem Referenzfall werden empirische Beziehungen für ein *Scale-Up*

bzw. *Scale-Down* der Koeffizienten formuliert. Dabei erfordern die zusätzlichen empirischen Parameter weitere Anpassungen.

- *Physikalische Koeffizientenmodelle* stellen den Grenzfall dar, der eintritt, wenn es gelingt, eine maßstabsunabhängige Beschreibung der relevanten Apparateeinflüsse unter Verwendung physikalischer Zusammenhänge vorzunehmen. Bei den hier eingehenden Modellparametern handelt es sich um rein stoffabhängige Größen.

3.5.2. Energiemodelle

Um über die Wirtschaftlichkeit eines Entwurfes Aussagen machen zu können, werden Energiemodelle für die einzelnen Apparate benötigt. Die Energiemodelle stellen also den Energieaufwand in Bezug zu der stofflichen Veränderung dar. Leider kann bei Feststoffprozessen der benötigte Energieaufwand nicht funktional aus den Zustandsgrößen der Prozessströme ermittelt werden, wie es bei fluiden Prozessen der Fall ist. Problematisch ist insbesondere, dass sich häufig beim Einsatz von mechanischer Energie keine eindeutige Aufteilung des Energieaufwandes in Transportanteile, Verluste und Anteile für die Stoffumwandlung vornehmen lässt. Die Berechnung der für den Apparat erforderlichen Antriebsleistung erfolgt dann entweder über empirische Ansätze oder mit Hilfe mechanischer Modelle basierend auf den Entwurfsgrößen und Betriebseinstellungen des Apparats und dem jeweiligen Materialdurchsatz.

3.5.3. Dimensionierungsmodelle

Um überhaupt eine Fließschemasimulation durchführen zu können, müssen die Entwurfs- und Betriebsparameter festgelegt werden. Eine Orientierung bei der Festlegung dieser Parameter bieten Dimensionierungsmodelle, die die Entwurfs- und Betriebsparameter anhand der Stromdaten des Eingangs- und Ausgangsstroms bestimmen. Prinzipiell können dafür die bereits vorgestellten Simulationsmodelle verwendet werden. Voraussetzung dafür ist allerdings, dass ein Zusammenhang zwischen Stoffwandlung und apparativen Größen gegeben ist. Die numerische Lösung des Problems erfolgt, indem die bisher abhängigen als unabhängige Größen definiert werden und umgekehrt.

In der praktischen Umsetzung sind jedoch hinsichtlich der Verwendung von rigorosen Simulationsmodellen verschiedene Einschränkungen zu treffen:

1. Es liegen zum Zeitpunkt des Prozessentwurfs in der Regel keine Informationen über die Modellparameter vor, die nicht Betriebs- oder Auslegungsgrößen sind.
2. Da die Anzahl der fraktionellen Bilanzen in der Regel nicht mit der Zahl der zu bestimmenden, unbekannten Apparateparameter übereinstimmt, erhält man im Ergebnis ein überbestimmtes Modellgleichungssystem, für das es keine analytische Lösung gibt.

Daher werden für die Dimensionierung *Shortcut*-Modelle verwendet. Die Spezifizierung des Prozess-Schrittes erfolgt dabei über die Angabe eines charakteristischen Parameters, beispielsweise einer Trenngrenze oder eines Zerkleinerungsverhältnisses, und der Durchsatzforderung, die sich aus dem vorherigen Prozess-Schritt ergibt. Die endgültige Festlegung der Parameter orientiert sich jedoch an der Verfügbarkeit diskreter Baureihen.

3.6. Simulationskontext

Oftmals werden bei der Simulation von Feststoffprozessen empirische oder halbempirische Simulationsmodelle verwendet (vgl. Abschnitt 3.5.1). Die für die Simulation zu verwendenden empirischen Parameter hängen sowohl von dem Materialstrom als auch von der apparativen Ausrüstung ab. Diese Kombination von Apparat, Material und zugehörigem Simulationsmodell wird als *Simulationskontext* bezeichnet.

Die Ermittlung gültiger Parameter für einen Simulationskontext ist aufwändig. Zunächst müssen die Fraktionsbäume der realen Zu- und Ablaufmaterialströme bestimmt werden. Die zu bestimmenden Eigenschaften werden durch das Prozessmodell vorgegeben. Danach werden die Parameter durch Optimierung bestimmt. Genau genommen gelten die ermittelten Parameter nur für diese Konstellation aus Material, Apparat und Modell und müssen auf Gültigkeit bei ähnlichen Materialien bzw. anderen Apparateeigenschaften überprüft werden.

Daher beinhaltet ein Simulationskontext neben den apparate- und stoffspezifischen Modellparametern auch Verweise auf die realen Zu- und Ablaufströme, sofern sie bestimmt wurden. Die realen Daten ermöglichen eine Bewertung des Modells und können ggf. Hinweise auf Verbesserungsmöglichkeiten liefern.

3.7. Zusammenfassung der assetorientierten Analyse

3.7.1. Problemstellung

Die Aufgabe des Informationssystems ist die Unterstützung des Wissensmanagements für das CAPE. Dabei müssen unterschiedliche Perspektiven berücksichtigt werden. Zum einen muss das System eine prozess-orientierte Sicht unterstützen, um den Phasen des Entwicklungsprozesses gerecht zu werden. Während der Durchführung des Prozesses fallen jedoch auch Daten an. Daher muss zum anderen auch eine produktorientierte Sicht unterstützt werden. Die anfallenden/benutzen Daten liegen in der Regel als Dokumente vor, können aber auch in Datenbanken enthalten sein. Beispielhaft sind Dateien mit allgemeiner technischer Dokumentation, Beschreibungen von Materialströmen, Apparaten und Simulationsmodellen sowie Stoffdatenbanken zu nennen. Der Strukturierungsgrad reicht also von unstrukturiert, über semistrukturiert bis hin zu strukturiert. Diese Vielfalt hat Auswirkungen auf die Wahl der Entwicklungs-Plattform (vgl. Abschnitte 4.1 und 4.2).

Neben der Problemstruktur, die sich aus der Aufgabenstellung ergibt, wird die Wahl der Entwicklungs-Plattform durch die Schnittstellen-Anforderungen an das System geprägt. Für die Unterstützung des CAPE wird außer einer grafischen Oberfläche für Experten auch eine Schnittstelle für Software-Anwendungen, wie z. B. Simulations-Software, benötigt.

Weitere zu berücksichtigende Aspekte bei der Wahl der Plattform ergeben sich aus allgemeinen Anforderungen, die sich durch das Wissensmanagement an das System ergeben. Eine ausführliche Analyse diesbezüglich befindet sich in [Weg02].

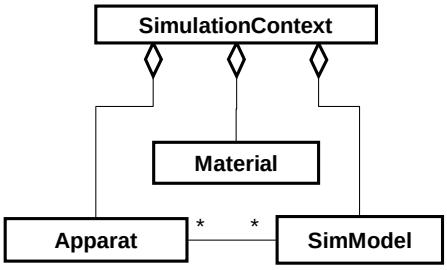


Abbildung 3.12.: Beziehungen des Asset-Modells der Anwendungsdomäne als konzeptuelles UML-Klassendiagramm

3.7.2. Struktur

Eine zentrale Bedeutung für die Simulation von Feststoffprozessen haben die Verfahrensfließbilder. Jeder Knoten des Graphen steht für einen Simulationskontext, der durch einen Apparat, ein Simulationsmodell und einen Zulaufstrom definiert ist. Der Simulationskontext legt die apparate- und materialspezifischen Parameter für die Simulation eines Prozess-Schrittes fest. Die Gültigkeit/Qualität der Parameter lässt sich anhand von realen Stromdaten, die dem Simulationskontext zugeordnet sind, überprüfen. Außerdem können einem Apparat verschiedene Simulationsmodelle zugeordnet werden, die diesen mathematisch beschreiben. Diese Zuordnung ist unabhängig von einem Simulationskontext. Die Assoziationen zwischen den einzelnen Entitäten sind in Abbildung 3.12 dargestellt.

3.7.3. Anwendungsfälle

1. **Definition von Apparate-, Material- und Modellparametern:**
   Der Anwendungsfall beinhaltet das Anlegen, Löschen und Bearbeiten von (Formal-) Parametern für Apparate, Materialien oder Modelle.
2. **Verwaltung von Apparaten:**
   Der Anwendungsfall beinhaltet das Anlegen, Löschen und Bearbeiten von Apparaten. Die Bearbeitung von Apparaten umfasst auch das Zuweisen von Apparateparametern sowie von passenden Prozessmodellen. Die Zuweisung von Parametern kann auch durch eine Daten-Extraktion aus der medialen Beschreibung des Apparats erfolgen.
3. **Verwaltung von Prozessmodellen:**
   Der Anwendungsfall beinhaltet das Anlegen, Löschen und Bearbeiten von Prozessmodellen. Die Bearbeitung von Prozessmodellen umfasst auch das Zuweisen von Modellparametern sowie von zugehörigen Apparaten. Die Zuweisung von Parametern kann auch durch eine Daten-Extraktion aus der medialen Beschreibung des Prozessmodells erfolgen.
4. **Verwaltung von Materialien (Materialströmen):**
   Der Anwendungsfall beinhaltet das Anlegen, Löschen und Bearbeiten von Materialien. Die Bearbeitung von Materialien umfasst auch das Zuweisen von Materialparametern. Die Zuweisung von Parametern kann auch durch eine Daten-Extraktion aus der medialen Beschreibung des Materials erfolgen.

5. **Verwaltung von Simulationskontexten:**
   Der Anwendungsfall beinhaltet das Anlegen, Bearbeiten und Löschen von Simulationskontexten. Die Erzeugung eines Simulationskontextes erfolgt implizit durch die Zuweisung eines Materials zu einer Apparat-Material-Kombination.
6. **Nutzerverwaltung:**
   Die Nutzerverwaltung beinhaltet das Anlegen, Bearbeiten und Löschen von Nutzern des Systems.
7. **Anmelden und Abmelden eines Nutzers**

Die hier beschriebenen Anwendungsfälle müssen sowohl durch eine grafische Benutzerschnittstelle als auch durch eine Service-Schnittstelle, für den Zugriff durch eine Simulations-Software, abgedeckt werden. Ausnahmen davon sind die Nutzerverwaltung und die Verwaltung der verschiedenen Parameter. Diese Aufgaben sollen nur über eine HTML-basierte Schnittstelle unterstützt werden.

4. Das Implementierungs-Umfeld

4.1. Plattform-Wahl

Für die Implementierung des Software-Systems wurde der INFOASSET BROKER als Softwareplattform verwendet. Der INFOASSET BROKER ist „ein integrales Portalsystem für das Wissensmanagement“ [inf01] und bietet daher bereits eine gute Unterstützung des dokumentenorientierten Wissensmanagements.

Für die Realisierung der Software-Schnittstelle standen unterschiedliche Middleware-Technologien zur Auswahl. Obwohl der CAPE-OPEN-Standard [CO-04] für die Entwicklung von Simulations-Software CORBA [Obj04] vorsieht, ist die Entscheidung für das leichtgewichtige SOAP [Wor03b, Wor03c, Wor03d] gefallen. Der Grund dafür ist, dass bei der vorgesehenen Service-Schnittstelle der Austausch und die Wartung von Informationen im Vordergrund stehen und nicht die Inter-Objekt-Kommunikation im eigentlichen Sinne. Für die Realisierung der SOAP-basierten Schnittstelle wurde APACHE AXIS ausgewählt, da im Arbeitsbereich bisher gute Erfahrungen mit Axis gemacht wurden und es zudem kostenfrei verfügbar ist.

Um beide Technologien zu integrieren, wird eine Servlet-Engine verwendet. Dies hat auch den Zweck, ggf. eine WebDAV-Schnittstelle leicht zur Verfügung stellen zu können. Die Wahl der Servlet-Engine fiel auf die JSP- und Servlet-Referenz-Implementierung APACHE TOMCAT. Die einzelnen Komponenten des Implementierungs-Umfeldes werden in den folgenden Abschnitten beschrieben.

4.2. Die Architektur des INFOASSET BROKERS

Der INFOASSET BROKER ist eine Standardsoftware für den Aufbau von personalisierbaren Content-Portalen im Internet und Enterprise-Knowledge-Portalen im Intranet. Im Folgenden wird der Begriff *Broker* synonym zu INFOASSET BROKER benutzt.

Die Architektur des Software-Systems lässt sich je nach Betrachtungsweise in unterschiedliche Schichten gliedern. Bei einer groben Sichtweise handelt es sich beim Broker um eine 3-Schichten-Web-Architektur, wie sie z. B. in [NMMZ00] beschrieben wird. Die Schichten der Web-Architektur greifen jeweils auf einen eigenen (virtuellen) Adressraum zu und können anhand dieses Kriteriums voneinander unterschieden werden. In der Regel kommunizieren die Schichten über Netzwerkprotokolle miteinander.

Die Schichten der Web-Architektur lassen sich in weitere Schichten unterteilen. Diese feineren Schichten grenzen sich anhand ihrer Funktion voneinander ab und liegen als einzelne Module und/oder Pakete vor.

Da die einzelnen Schichten festgelegte Schnittstellen besitzen und nur mit den benachbarten Schichten kommunizieren, lassen sie sich relativ einfach durch andere Implementierungen austauschen. Dadurch ist das System flexibel und kann schnell an neue Anforderungen angepasst werden. Zusätzlich erleichtert die Schichtenarchitektur die Wiederverwendung einzelner Komponenten.

Die einzelnen Schichten der Architektur, sowie deren Unterschichten, sind in Abbildung 4.1 zu sehen und werden im Folgenden kurz vorgestellt. Eine detaillierte Übersicht über den INFOASSET BROKER befindet sich in der Dissertation von Holm Wegner [Weg02], zahlreichen Studien- und Diplomarbeiten [HS03], [Det03], [Koc02], [Jac02], [Büc02], [Leh01], [Die01] und dem INFOASSET Whitepaper [inf01].

1. **Web-Client:**

   a) **Präsentation:**
   Diese Schicht bietet dem Endbenutzer eine grafische Oberfläche, um mit dem System interagieren zu können. Im Wesentlichen werden dafür Web-Browser und mobile Endgeräte eingesetzt. Es ist aber auch über andere Netzwerk-Clients möglich, mit dem Broker zu kommunizieren (z. B. mit Mail- und FTP-Clients).

2. **Web-Server:**

   a) **Kommunikation:**
   In dieser Schicht wird die Kommunikation des Portal-Servers mit der clientseitigen Präsentationsschicht gesteuert. Dabei werden die jeweiligen Protokolle auf die generische Interaktionsschicht abgebildet. Beispiele dafür sind Web-Server, die mittels HTTP kommunizieren, oder FTP-Server, die Dateien mittels

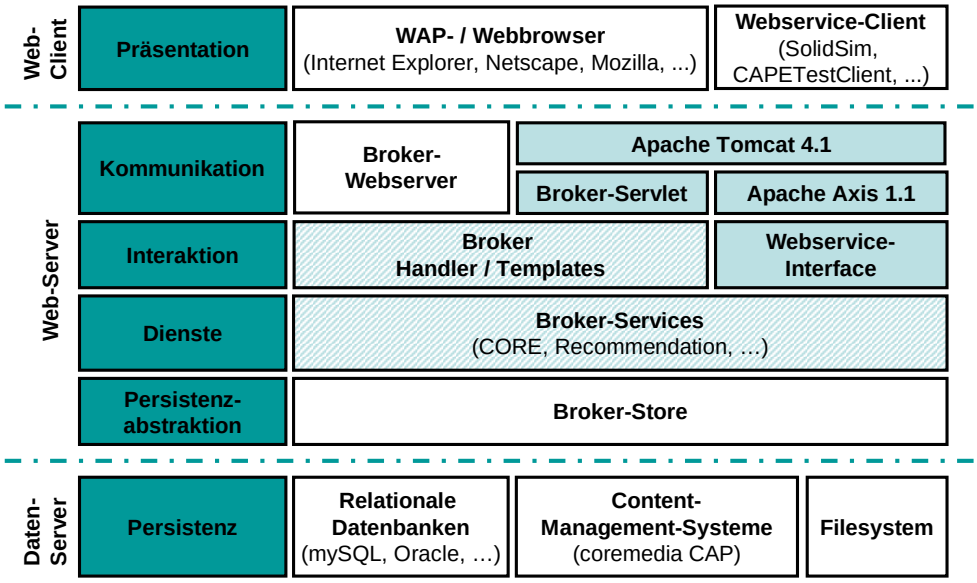


Abbildung 4.1.: Schichtenarchitektur des INFOASSET BROKERS: Die Grafik zeigt die einzelnen Ebenen der 3-Schichten-Webarchitektur nach [NMMZ00] sowie eine Unterteilung in verschiedene Funktionsschichten. Zusätzlich sind die realisierenden Softwarekomponenten abgebildet

FTP Dateien anbieten. Im Rahmen dieser Diplomarbeit wurde die Web-Server-Implementierung des Brokers durch ein *Servlet* ersetzt, das Anfragen auf die Interaktionsschicht abbildet und in der APACHE TOMCAT 4.1 Servlet-Engine läuft (vgl. Abschnitt 4.5).

b) **Interaktion:**
Diese Schicht ist für die clientspezifische Dokumentenerzeugung zuständig. So werden für Webbrowser Dateien im HTML-Format erzeugt und für WAP-Browser entsprechend im WML-Format. Die Generierung dieser Formate erfolgt dabei über formatunabhängige Regeln (*Handler*) und formatspezifische Vorlagen (*Templates*).

Die prinzipielle Funktionsweise eines `Handlers` ist dabei recht einfach. Zunächst wird eine Instanz der Klasse `Template` erzeugt. Diese wird mit einer Vorlagendatei (HTML- oder WML-Datei) initialisiert. Bei der Initialisierung wird die Datei auf durch $-Zeichen hervorgehobene Platzhalter untersucht. Die Platzhalter sind Referenzen auf in der `Handler`-Klasse in Form von `Substitution`-Klassen definierte Ersetzungen. Anschließend wird der Ersetzungsvorgang des `Template`-Objekts gestartet (vgl. Abschnitt 6.2.3).

Neben einfachen Ersetzungen unterstützt der Broker auch bedingte Ersetzungen und Ersetzungsschleifen. Innerhalb von Ersetzungsschleifen können weitere Ersetzungen definiert werden, die abhängig vom Schleifendurchgang unterschiedliche Werte in die Ausgabe einfügen. Die Syntax der einzelnen Ersetzungsoperationen ist wie folgt:

- **Einfache Ersetzung**

  Syntax: $*Label*$

  Funktion:
  Der angegebene Platzhalter wird in der Ausgabe durch einen String-Wert ersetzt. Der Wert wird durch eine `PrintSubstitution`, die beim zugehörigen `Template` unter dem Namen *Label* registriert ist, berechnet.

- **Bedingte Ersetzung**

  Syntax: $[*Label*$ Text A $]*Label*[$ Text B $*Label*]$

  Funktion:
  Der gesamte Ausdruck wird bei einer erfüllten Bedingung durch Text A ersetzt und bei einer nicht erfüllten Bedingung durch Text B. Die Bedingung wird innerhalb einer `ConditionalSubstitution` definiert und während des Ersetzungsvorganges ausgewertet. Sie ist wie die anderen `Substitution`-Klassen beim zuständigen `Template` unter dem Namen *Label* registriert. Innerhalb von Text A und Text B sind weitere Ersetzungsausdrücke, die rekursiv vom `Handler` abgearbeitet werden, erlaubt.

- **Ersetzungen in einer Schleife**

  Syntax: $[*Label cid*$ Text A Text B $]*Label*[$ Text C $*Label*]$

Funktion:
Unter dem Namen *Label* wird bei dem zugehörigen `Template` ein `ListTemplate` registriert. Das `ListTemplate` iteriert über eine Menge von Informationsobjekten und gibt in jedem Schleifendurchlauf den Text A und im letzten Durchlauf den Text B aus. Falls die Menge leer ist, wird Text C ausgegeben. Die einzelnen Ersetzungstexte können wiederum Ersetzungsausdrücke bzw. Schleifen enthalten, diese müssen allerdings einen qualifizierten Bezeichner mit dem Prefix *cid* haben. Die Ersetzungselemente können auf das aktuelle Informationsobjekt des Schleifendurchlaufs zugreifen. Die qualifizierten Bezeichner werden auch benutzt, um Text A von Text B zu unterscheiden. Dafür müssen in beiden Texten dieselben Ersetzungsausdrücke in der gleichen Reihenfolge verwendet werden.

Abbildung 4.2 zeigt ein konzeptionelles Klassendiagramm der `Templates` und `Substitutions`.

c) **Dienste:**
Die Diensteschicht des Brokers verwaltet die verschiedenen Fachkonzeptobjekte und enthält den größten Teil der Anwendungslogik. Dazu definiert der Broker auf Basis der semantischen Objekte Person (*Person*), Gruppe (*Group*), Verzeichnis (*Directory*), Dokument (*Document*) und Konzept (*Concept*) ein Wissensnetzwerk (vgl. Abbildung 4.3). Dieses bildet die Grundlage verschiedener Basisdienste.

- **Personalisierte Inhalte:**
  Anfrageergebnisse können in zweierlei Hinsicht auf den Anwender zugeschnitten sein: in Bezug auf a) die Erscheinungsform der Inhalte (*Layout*) und b) die Inhalte selbst. Inhalte können in verschiedenen Anzeigeformaten (HTML, WML, usw.) oder in verschiedenen Sprachen dargestellt werden. Weiterhin können Inhalte benutzer- oder rollenbezogen angepasst werden. Als Rolle wird die Art der Zugehörigkeit eines Benutzers zu einer bestimmten Benutzergruppe bezeichnet.

- **Dokumentenmanagement:**
  Das Dokumentenmanagement des Brokers beschränkt sich auf Funktionen zum Abspeichern und Herunterladen von Dokumenten auf den bzw. vom Server, zur Versionsverwaltung für Dokumente, zur automatischen Indexerstellung für die Volltextsuche, sowie zur Verwaltung von Dokumentmetadaten.

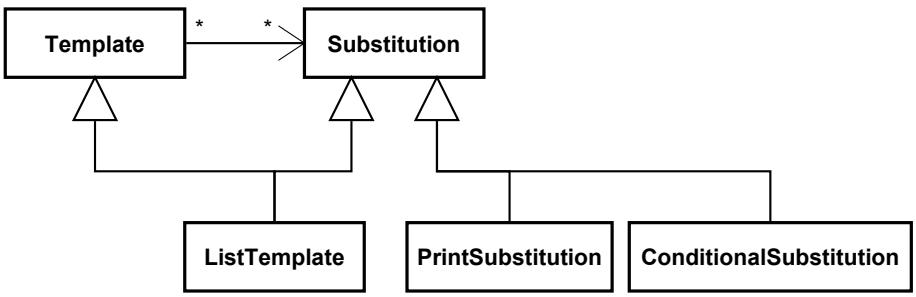


Abbildung 4.2.: Konzeptuelles Klassendiagramm der `Templates` und `Substitutions`

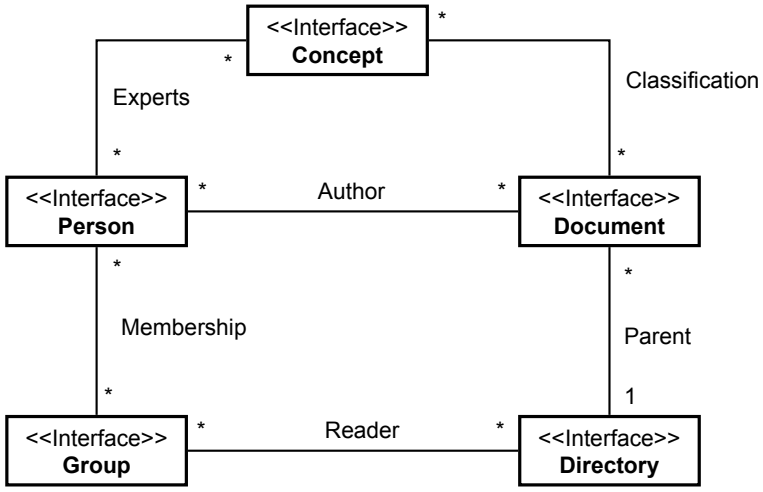


Abbildung 4.3.: Ausschnitt aus dem Metamodell für ein Wissensnetzwerk

- **Community:**
  Unter diesem Begriff werden folgende Funktionen zusammengefasst: Registrierung und Authentifizierung von Benutzern, Anlegen und Editieren von Gruppen sowie die Pflege von Gruppenmitgliedschaften mit den entsprechenden Zugriffsrechten auf verschiedene Informationsobjekte, Benutzerdaten und Registrierungsinformationen, die mit anderen Systemen ausgetauscht werden können, Verwaltung von Benutzerprofilen und Diskussionsforen.

- **Portal:**
  Es werden Funktionen zur Verfügung gestellt, die unter anderem schnelles Suchen auf gespeicherten Informationsobjekten zulassen. Das Hauptwerkzeug hierfür ist ein interaktiver, grafischer Navigator (*concept navigator*). Dieser erleichtert das Recherchieren in großen, hierarchisch angeordneten oder im Netz organisierten Informationssammlungen.

Neben diesen Basisdiensten existieren zahlreiche Zusatzdienste (*extensions*), die miteinander kombiniert werden können [inf04]. Eine Auswahl dieser Erweiterungen wird im Folgenden kurz erläutert.

- **Alerting Extension:**
  Die Alerting Extension stellt Benachrichtigungsdienste bereit, mit der Benutzer gemäß ihrer persönlichen Interessen informiert werden, wenn für sie relevante Dokumente verändert wurden oder neue Dokumente innerhalb ihres Interessengebietes hinzugefügt wurden.

- **Asset Discovery Extension:**
  Die selbstlernende Asset Discovery Extension stellt eine Fuzzy-Ähnlichkeitssuche sowie die automatische Klassifikation von Dokumenten zur Verfügung.

- **Recommendation Extension:**
  Die Recommendation Extension wendet Verfahren des Collaborative Filte-

ring an, um die Bewertung von Dokumenten durch Nutzer mit ähnlichem Interessenprofil für den Mitarbeiter nutzbar zu machen. So werden Informationsobjekte, die von Nutzern mit ähnlichen Interessen als besonders hilfreich eingestuft wurden, empfohlen und in Trefferlisten bevorzugt angezeigt.

- **Task Management Extension:**
  Die Task Management Extension dient der geregelten Abarbeitung vordefinierter Workflows. Dazu stehen In-Boxen und Controlling-Mechanismen zur Verfügung.
- **Online Test Extension:**
  Mit der Online Test Extension können Web-basierte Tests zusammengestellt, automatisch durchgeführt und ausgewertet werden. Die Online Test Extension dient typischerweise für die Personalrekrutierung oder wird im Zusammenhang mit E-Learning verwendet. Sie kann auch zur Durchführung von Online-Befragungen für Marketingzwecke oder im Rahmen einer Mitarbeiterbefragung eingesetzt werden.

d) **Persistenzabstraktion:**
Um unterschiedliche Systeme zur Datenhaltung zu unterstützen, besitzt der Broker eine Persistenzabstraktionsschicht. Diese Schicht wird auch als *Store* bezeichnet und bildet die verschiedenen Asset-Typen der darüberliegenden Dienstesschicht auf die Persistenzschicht ab.

Der Begriff *Asset* hat im Zusammenhang mit dem Broker eine andere Bedeutung als in der AOA. Die Assets des Brokers werden daher in den folgenden Kapiteln als *iAssets* bezeichnet. IAssets stehen für die semantischen Objekte des Brokers. Es wird bei ihnen nicht zwischen inhaltlichen und konzeptuellen Eigenschaften unterschieden. Außerdem werden iAssets häufig verwendet, um `n:m`-Beziehungen zu modellieren.

Die transparenten Lese-, Schreib- und Modifikationsoperationen werden über Daten-Server-spezifische Implementierungen der `Container`- und `Content`-Schnittstelle realisiert. Die Kommunikation zu den Daten-Servern erfolgt, je nach Implementierung, entweder über die JDBC-Schnittstelle oder eine herstellerspezifische API.

3. **Daten-Server:**

a) **Persistenz:**
Diese Schicht ist für die dauerhafte Speicherung der Daten verantwortlich. Der Broker unterstützt neben unterschiedlichen relationalen Datenbanken auch Content-Management-Systeme für die Datenspeicherung. Für diese Arbeit wurde die MySQL Datenbank eingesetzt. Durch Austauschen der Persistenzabstraktionsschicht ist aber auch die Verwendung einer anderen Datenbank möglich.

4.3. Semantische Objekte im INFOASSET BROKER

Innerhalb des Brokers werden semantische Objekte auf iAssets abgebildet. IAssets abstrahieren durch die Store-Schicht vom verwendeten Datenspeicher wie einer Datenbank oder

einem Content Management System (CMS). Ein Klassendiagramm für das Asset- und Persitenzmanagement im Bezug auf das semantische Objekt `Person` ist in Abbildung 4.4 zu sehen. Die `IMPService`-Klasse implementiert die `Service`-Schnittstelle und entspricht dem Architekturmuster *Singleton* [GHJV94]. Sie definiert hauptsächlich Methoden für den Zugriff auf spezielle `AssetContainer` wie `IMPPersons`. Es wird jedoch von der Implementierung durch eine entsprechende `Persons`-Schnittstelle abstrahiert. Dieser Abstraktionsmechanismus wird auf der gesamten Diensteschicht verwendet. Die `Persons`-Schnittstelle bietet wiederum Zugriff auf Objekte der `IMPPerson`-Klasse. Sowohl `IMPPersons` als auch `IMPPerson` erhalten durch Vererbung von den abstrakten Proxy-Klassen [GHJV94] den Zugriff auf die eigentliche Persistenzschicht.

Bei der Implementierung zusätzlicher iAsset-Klassen sind folgende Aufgaben zu erledigen.

- **Erweiterung der Service-Schnittstelle:**
  Die `Service`-Schnittstelle muss um Methoden für den Zugriff auf den speziellen `AssetContainer` erweitert werden. Bei speziellen, projektbezogenen Erweiterungen ist es sinnvoll, statt der Schnittstelle die `IMPService`-Klasse durch Vererbung zu erweitern. Die `IMPMyService`-Klasse wird durch eine entsprechende Konfiguration des Brokers statt der ursprünglichen `IMPService`-Klasse bei der Initialisierung geladen. Die projektbezogenen Software-Module müssen dann beim Zugriff auf das `Service`-Singleton, dieses zu einem `IMPMyService`-Objekt umwandeln (*casten*).

- **Definition der speziellen iAsset-Schnittstellen:**
  Des Weiteren müssen für die zusätzlichen iAssets und auch für die iAsset-Container entsprechende Schnittstellen definiert werden, beispielhaft mit den Namen `MyAsset` und `MyAssets`. Durch `Get-` und `Set`-Methoden werden die Attribute der `MyAsset`-Implementierung gekapselt. Suchoperationen sowie Operationen für die Erzeugung und das Löschen von `Assets` werden in der zugehörigen Container-Schnittstelle `MyAssets` definiert.

- **Implementierung der speziellen iAsset-Schnittstellen:**
  Zusätzlich müssen die im vorigen Schritt definierten Schnittstellen implementiert werden. Entsprechend der Abbildung 4.4 werden die beispielhaften Klassen `IMPMyAsset` und `IMPMyAssets` genannt. Die eigentliche Implementierungsaufgabe ist relativ einfach, da sie analog zu der Implementierung der bereits bestehenden Klassen wie `IMPPerson` erfolgt. Neben der Implementierung von Suchoperationen, die auf das Query-Framework der Persistenzschicht zugreift, muss in der `IMPMyAssets`-Klasse auch der `ContentType` des Asset-Typs definiert werden. Zugriffen auf Eigenschaften der `IMPMyAsset`-Klasse werden auf die `Content`-Klasse der Persistenzschicht delegiert.

4.4. Dokumente und Verzeichnisse des Brokers

Da bei der im folgenden Kapitel beschriebenen Umsetzung der Analyseergebnisse aus Kapitel 3 das Dokumenten- und Verzeichnis-iAsset des Brokers eine zentrale Bedeutung hat, wird diese kurz vorgestellt.

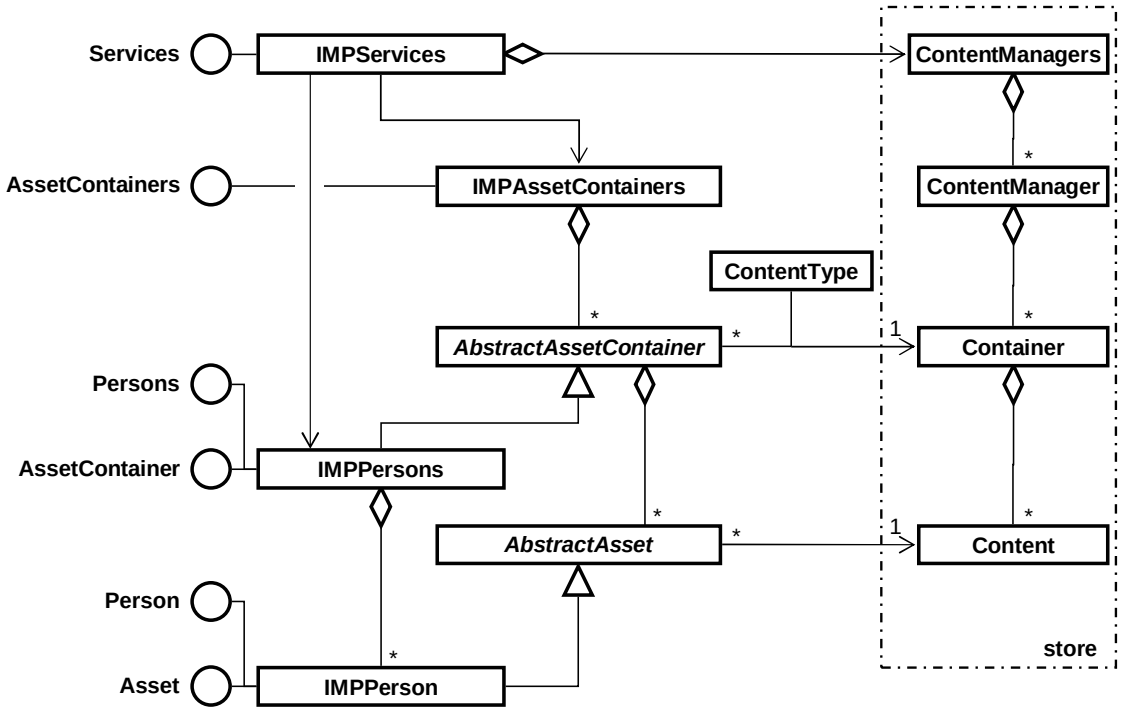


Abbildung 4.4.: Persistenz- und Assetmanagement des INFOASSET BROKERS am Beispiel des `Person`-iAssets

Dokumente bestehen hauptsächlich aus einem Verweis auf eine Datei, der allerdings nicht zwingend verwendet werden muss, sowie einer Fülle von unterschiedlichen Metainformationen. Diese umfassen z. B. Autor, Erfasser, Zusammenfassung, Datum der letzten Änderung/Erzeugung und Titel des Dokuments. Unterschiedliche Dokumentarten können anhand eines Dokumenttyp-Feldes voneinander unterschieden werden. Außerdem kann über den Dokumenttyp, der aus dem Wertebereich der Domäne `DokumentKind` ist, eine Taxonomie aufgebaut werden. Alle Dokumenttypen teilen sich jedoch die gleichen Attribute. Daher werden Dokumenttyp-spezifische Anwendungslogik und Darstellung auf Ebene der Handler und durch Typ-spezifische Templates realisiert. Dabei werden von dem jeweiligen Dokumenttyp nicht benötigte Felder ignoriert. Einen entsprechenden Aufbau haben auch Verzeichnisse. Ein Dokument, das in einem speziellen Verzeichnis erstellt wird, hat automatisch den gleichen Typ wie das Verzeichnis.

Beispielsweise existiert im Broker der Dokument- und Verzeichnistyp 'news', der für Mitteilungen benutzt wird. Im Kontext einer Nachricht bekommen die Dokumentfelder durch eine angepasste Visualisierung eine neue Semantik für den Anwender. So enthält das Feld `Abstract` bei einer Textdatei die Zusammenfassung und bei einer Nachricht die eigentliche Mitteilung. Bei der Schaffung neuer Dokumenttypen liegt es in der Verantwortung des Entwicklers, auf eine konsistente Semantik bei der Verwendung der Felder zu achten. Das bedeutet, dass ggf. fehlende Dokumentfelder bei der Definition neuer Dokumenttypen ergänzt werden müssen.

4.5. APACHE TOMCAT 4.2

APACHE TOMCAT [Apa04a] ist die offizielle Referenzimplementierung der Java-Servlet- [JSR53a] und der Java-Server-Pages-Technologie [JSR53b]. Die Java-Servlet- und die Java-Server-Pages-Spezifikationen wurden bei SUN MICROSYSTEMS im Java-Community-Process entwickelt und stehen der Öffentlichkeit zur Verfügung.

Für das Software-System sind insbesondere die Fähigkeiten Tomcats als Servlet-Engine von Bedeutung. Bereits im WIPS-Projekt wurde eine Servlet-basierte WebDAV-Schnittstelle implementiert [Ge03], die möglicherweise auch im CAPE Information-Broker zum Einsatz kommen sollte. Da sich beim WIPS-Projekt [Hup04] gezeigt hatte, dass eine Integration des Tomcat-Servers in den Broker unvorteilhaft ist, wurde in diesem Projekt der Broker als Web-Anwendung in die Servlet-Engine integriert. Dafür wurde ein Broker-Servlet geschrieben, das von der eigentlichen Broker-Interaktionsschicht abstrahiert (vgl. Abschitt 6.1.1).

Neben der WebDAV-Schnittstelle wird der Servlet-Container auch für die Integration eines Servlet-basierten Webservices benötigt. Zusätzlich bietet die Integration des Brokers in Tomcat die Möglichkeit, zukünftig Java-Server-Pages in der Interaktionsschicht einzusetzen.

4.5.1. Aufbau eines Java-Servlets

Java-Servlets sind Protokoll- und Plattform-unabhängige serverseitige Software-Komponenten, die Web-Server mit einer entsprechenden Servlet-Engine dynamisch erweitern. Sie basieren auf einem allgemeinen Rahmenwerk, das Web-Anwendungen nach dem Request-Response-Paradigma unterstützt. Dieses Programmiermodell wird auch bei der Kommunikation über HTTP oder *remote procedure calls* (RPC) angewendet.

Servlets zeichnen sich durch die Implementierung der Servlet-Schnittstelle aus, die von der Servlet-Engine für die Weiterleitung der Anfragen genutzt wird. I. Allg. wird die Schnittstelle nicht direkt, sondern durch Erweiterung einer allgemeinen oder HTTP-spezifischen Implementierung realisiert. Das einfachste mögliche Servlet definiert die `service()`-Methode:

```
import java.servlet.*;
public class MyServlet extends GenericServlet {
        public void service (
            ServletRequest   request,
            ServletResponse  response
        ) throws ServletException, IOException
        {
            [...]
        }
        [...]
    }
```

Die `service()`-Methode besitzt einen `request`- und einen `response`-Parameter, die gesendete Daten des Clients kapseln. Sie bieten neben dem Zugriff auf die gesendeten Parameter auch die Möglichkeit, Statusinformationen bereitzustellen. Normalerweise wird der Zugriff auf die empfangene Anfrage über einen Eingabestrom und das Versenden der Antwort über einen Ausgabestrom realisiert.

```
ServletInputStream    in = request.getInputStream ();
ServletOutputStream   out = response.getOutputStream ();
```

Diese Ein- und Ausgabeströme können mit Daten jeglichen Formats verwendet werden. Zum Beispiel können Applets und Servlets Daten über serialisierte Objekte austauschen, aber auch HTML und unterschiedliche Bildformate können verwendet werden.

4.6. Apache AXIS 1.1

Das Apache extensible Interaction System (AXIS) [Apa04b] ist ein Java-basiertes Open-Source-Framework für die Entwicklung und Bereitstellung von Webservices, die auf dem *simple object access protocol* (SOAP) beruhen. Es ist das Nachfolgeprojekt des Apache SOAP Frameworks der Apache Software Foundation. Die gegenwärtige Version ist AXIS 1.1., die auch für dieses Projekt verwendet wurde.

SOAP ist ein leichtgewichtiges, zustandsloses Protokoll für den Informationsaustausch in heterogenen, verteilten Systemlandschaften. Die Kommunikation zwischen den einzelnen Systemen basiert auf dem Austausch von XML-Nachrichten. Daher definiert das Protokoll im Wesentlichen einen XML-Envelope, der die jeweilige Nachricht aus optionalem XML-Header und XML-Body enthält. Prinzipiell kann SOAP auf der Basis unterschiedlicher Protokolle, wie SMTP oder HTTP, verwendet werden. In der Arbeit ist jedoch nur eine Verwendung auf Basis von HTTP vorgesehen, da das Session-Management über HTTP-Cookies realisiert wurde.

Webservices können durch die *Webservice Decsription Language* (WSDL) beschrieben werden. Mit Hilfe des AXIS-eigenen Werkzeuges `wsdl2java` lassen sich aus einer WSDL-Datei sowohl server- und clientseitige Proxy-Klassen als auch die von Axis benötigten Deployment- und Undeployment-Descriptoren generieren. Die WSDL-Datei selbst kann aus einem Java-Interface oder einer Klasse, die den Webservice implementiert, mit dem `java2wsdl`-Tool erzeugt werden. Dabei sind jedoch einige Einschränkungen zu beachten. Axis benutzt für die Abbildung von Java-Typen auf XML-Typen die *Java API for XML-based RPC* (JAX-RPC) [Sun04] . Neben den Schnittstellen definiert die API auch, wie Java mit XML-basierten Protokollen wie SOAP umgeht. So spezifiziert die JAX-RPC die Abbildung der Java-Primitive `boolean`, `byte`, `short`, `int`, `long`, `float` und `double` auf XML-Schema-Datentypen. Die Verwendung der entsprechenden Wrapper-Klassen (`java.lang.Integer`, `java.lang.Boolean` usw.) führt im Wesentlichen zu dem gleichen Ergebnis. Axis ermöglicht dann jedoch auch die Übergabe von `null`-Werten. Zusätzlich spezifiziert die JAX-RPC [Sun04] Abbildungen für folgende Standard-Java-Klassen:

- `java.lang.String`
- `java.math.BigInteger`
- `java.math.BigDecimal`
- `java.util.Calendar`
- `java.util.Date`

Zusammengefasst können die einfachen Datentypen, ihre jeweiligen *Wrapper* sowie die aufgezählten Standardklassen als einfache Bausteine für JAX-RPC-basierte Nachrichten verstanden werden.

Damit Axis in der Lage ist, aus der Schnittstelle/Klasse eine WSDL-Datei zu generieren, muss für die Datentypen, die durch Rückgabetypen und Parameter der Methoden sichtbar sind, einer der folgenden Fälle zutreffen.

- Es handelt sich um einen einfachen Baustein.
- Es handelt sich um eine Klasse oder ein Array, die jeweils auf einfache Bausteine reduziert werden können. Im Prinzip verhalten sie sich dann wie Container. Die Reduzierung auf einfache Bausteine ist bei Klassen möglich, die der *JavaBeans*-Konvention entsprechen. Dies ist der Fall, wenn die Klasse einen öffentlichen Standardkonstruktor, öffentliche Felder und/oder `Get-`, `Set-` bzw. `Is-`Methoden besitzt. Die Vererbung ist dabei nicht eingeschränkt.

Da die generierten Proxy-Klassen für das Erzeugen der XML-Nachrichten zuständig sind, wird auf das eigentliche SOAP-Format nicht weiter eingegangen, sondern auf [Wor03b], [Wor03c], [Wor03d] und [Gal03] verwiesen.

AXIS kann sowohl client-, als auch serverseitig eingesetzt werden. Wie bereits erwähnt, wird in diesem Projekt AXIS nicht als eigenständiger Server eingesetzt, sondern über ein Servlet in das System eingebunden.

5. Entwurf

5.1. Allgemeine Asset2iAsset-Abbildung

In diesem Abschnitt wird die Umsetzung eines Modells der AOA auf die iAssets des Brokers beschrieben. Die Umsetzung der konzeptuellen Sichtweise ist im Gegensatz zur medialen Sichtweise relativ einfach mit der Ausdrucksstärke der iAssets möglich.

Die charakteristischen Eigenschaften werden auf iAsset-Attribute abgebildet. Dabei werden einfache Java-Primitive (z. B. `int`, `long`, `boolean` und `float`) auf die zugehörigen *Wrapper*-Klassen oder die entsprechenden Standardklassen (z. B. `java.lang.String`) auf der iAsset-Ebene verwendet. Schwieriger ist jedoch die Abbildung von komplexen Objektstrukturen, die im Asset-Metamodell auch für die Modellierung von charakteristischen Eigenschaften erlaubt sind. Prinzipiell stehen dafür unterschiedliche Verfahren zur Verfügung. Eine Möglichkeit ist es, Objekte in serialisierter Form in BLOB-Feldern der iAssets abzulegen. Dieser Weg hat jedoch den Nachteil, dass die Unterstützung von Suchoperationen schwierig ist. Außerdem führt der Ansatz zu Problemen bei der Schemaevolution, wenn z. B. der Datentyp erweitert wird und dadurch zur Laufzeit unterschiedliche Versionen der gleichen Klasse benötigt werden. Ein alternativer Ansatz ist die Abbildung der komplexen Datentypen auf mehrere eigene iAsset-Klassen. Dieser Ansatz ist besonders gut, wenn es sich bei dem neuen Datentyp um eine flache Komposition bekannter Datentypen handelt.

Beziehungen werden je nach Art unterschiedlich im Broker umgesetzt. `1:1`- und `1:n`-Beziehungen werden direkt und `n:m`-Beziehungen mit Hilfe von zusätzlichen Beziehungs-iAssets realisiert.

Constraints aus der Systematik-Beschreibungskategorie von Peirce werden in den `Get`- und `Set`-Methoden der iAssets implementiert. Bei Verletzung der *Constraints* können wie bei *Enterprise-Java-Beans* (EJB) spezielle `VetoExceptions` geworfen werden.

Inhaltselemente der Assets werden bei den iAssets durch Referenzen auf die entsprechenden Inhaltsobjekte abgebildet. Die Inhalte werden dafür in speziellen Containern verwaltet. Bisher existiert dafür im Broker ein Datei-Container.

5.2. Abbildung des assetorientierten CAPE-Modells

Um den späteren Implementierungsaufwand klein zu halten, wird von der im letzten Abschnitt beschriebenen Methodik abgewichen. Statt der Implementierung neuer iAsset-Klassen wird versucht, Assets auf bereits vorhandene iAsset-Klassen des Brokers abzubilden.

Die in der AOA identifizierten Assets zeichnen sich dadurch aus, dass sie jeweils nur ein Inhaltselement besitzen. In allen Fällen handelt es sich dabei um eine XML-Datei, deren Format durch ein XML-Schema spezifiziert ist.

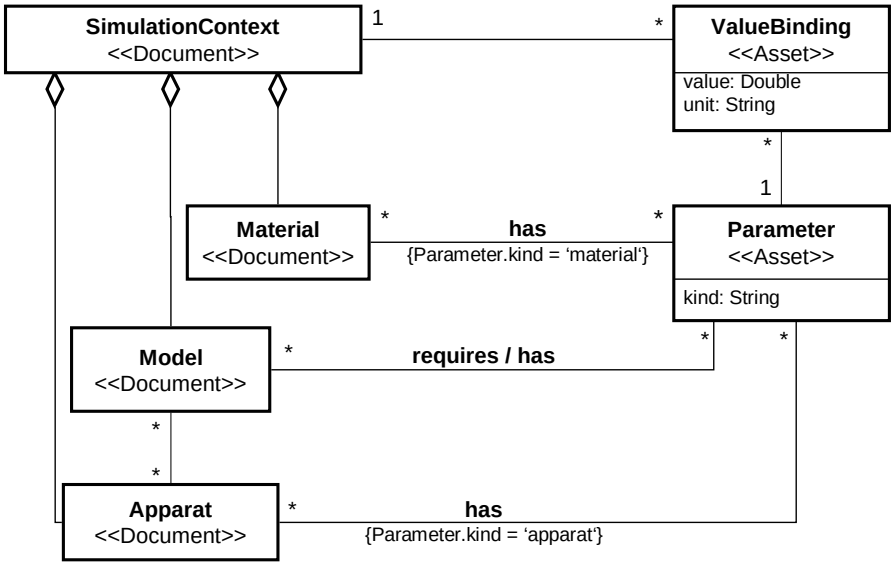


Abbildung 5.1.: Umsetzung der Ergebnisse aus der AOA auf die Plattform INFOASSET BROKER

Daher liegt die Verwendung des Dokument-iAssets als Grundlage für die Umsetzung der einzelnen identifizierten Assets (Apparat, Modell, Material und Simulationskontext) nahe. Auch die einfachen Eigenschaften dieser Assets lassen sich auf die bereits vorhandenen Dokumenteigenschaften abbilden. Die 'name'- und 'description'-Eigenschaften der Asset-Klassen Apparat, Modell und Material werden auf die Dokumentenfelder 'title' und 'abstract' abgebildet. Diese Felder sind für einen Simulationskontext nicht vorgesehen. Die komplexen Eigenschaften der Assets werden im Wesentlichen aus einer variablen Anzahl von `Parameter`- bzw. `ValueBinding`-Objekten beschrieben. Parameter und zugehörige Werte sind flache Kompositionen aus einfachen Standardklassen. Daher werden, wie im vorigen Abschnitt beschrieben, die `Parameter`- und `ValueBinding`-Klassen auf eigenständige iAsset-Klassen abgebildet.

Auch die Beziehungen zwischen den Assets werden wie beschrieben auf Beziehungsklassen des Brokers abgebildet. Die Semantik der Assets wird durch eine entsprechende Implementierung der iAssets berücksichtigt. Der sich aus der Asset2iAsset-Abbildung ergebende Entwurf auf Basis des Brokers ist als Klassendiagramm in Abbildung 5.1 zu sehen.

5.3. CAPE-Service-Schnittstelle

Die Service-Schnittstelle muss dem Client nur einen Teil der gesamten Broker-Funktionalität zur Verfügung stellen. Damit der Zugriff möglichst einfach und nicht das gesamte System exponiert ist, wurde das *Facade*-Architekturmuster [GHJV94] verwendet. Die `Facade`-Klasse wurde so entworfen, dass sie JAX-RPC konform ist. Zu diesem Zweck wurden Proxy-Klassen definiert, die sich auf JAX-RPC Bausteine reduzieren lassen. Insbesondere mussten die im Broker häufig verwendeten `Iteratoren` auf `Arrays` abgebildet werden. Die `Proxy`-

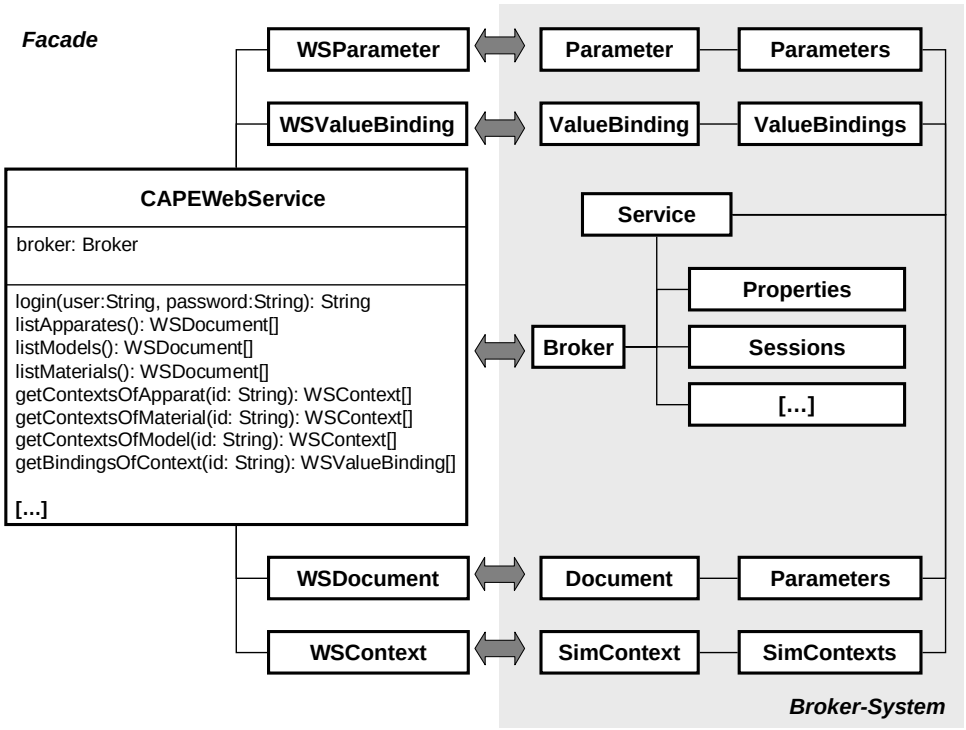


Abbildung 5.2.: Entwurf der Service-Schnittstelle

Klassen wurden auch verwendet, um nicht benutzte Felder der Dokument-iAssets zu verbergen. Dies hat sowohl Sicherheits- als auch *Performance*-Gründe.

Ein Entwurfsklassendiagramm ist in Abbildung 5.2 zu sehen. Es zeigt die `CAPEWebService`-Klasse, die das komplexe Broker-System nach außen verbirgt. Aus Gründen der Übersichtlichkeit wurde auf eine vollständige Auflistung der Service-Methoden verzichtet. Die Assoziationen auf der linken Seite der Abbildung zu den Proxy-Klassen `WSDocument`, `WSContext`, `WSParameter` und `WSValueBinding` repräsentieren entsprechend [FS00] Verantwortlichkeiten, d. h. dass nicht auf entsprechende Referenzen in der Implementierung geschlossen werden kann. Um jedoch auch die Beziehungen der Schnittstelle zum Broker-System zu verdeutlichen, wurden Blockpfeile für die entsprechenden implementierungsnahen Assoziationen verwendet. Innerhalb des Broker-Systems stehen die Assoziationen wieder für Verantwortlichkeiten.

6. Implementierung

Dieses Kapitel beschreibt die Implementierung, die im Rahmen des Projektes geleistet wurde. Die Implementierung kann entweder der Broker-Architektur, der Web-basierten Nutzerschittstelle oder der SOAP-basierten Service-Schnittstelle zugeordnet werden.

6.1. Anpassung der Broker-Architektur

Aufgrund der Anforderungen an den Broker musste die Architektur (vgl. Abschnitt 4.1) auf unterschiedlichen Ebenen erweitert bzw. angepasst werden. Die Kommunikationsschicht wurde durch die Integration des Brokers in eine Servlet-Engine angepasst und ist im folgenden Abschnitt beschrieben. Die Diensteschicht wurde um einen Parameter- und ValueBinding-Service ergänszt (vgl. Abschnitt 6.1.2). Da der zusätzliche Service Fließkommazahlen verwendet, wurde die Persistenzabstraktionsschicht um eine Unterstützung von Fließkommazahlen erweitert (vgl. Abschnitt 6.1.3).

6.1.1. Integration des Brokers in APACHE TOMCAT

Durch die Integration des Brokers in die APACHE TOMCAT Servlet-Engine wurde die bisher verwendete Web-Server-Komponente des Brokers ersetzt. Die neue Broker-Web-Anwendung basiert neben einigen Konfigurationsdateien im Wesentlichen auf einem Servlet.

Die `service()`-Methode der `BrokerServlet`-Klasse entspricht zu einem großen Teil der `processRequest()`-Methode der `HttpConnection`-Klasse, die ein Bestandteil des Broker-Web-Servers ist. Daher wird die Implementierung dieser Methode hier nicht weiter beschrieben. Neben der `Servlet`-Schnittstelle wurde auch die `ServletContextListener`-Schnittstelle implementiert. Die entsprechende `BrokerContextListener`-Klasse wird nämlich automatisch beim Starten der Web-Anwendung geladen und bei der Initialisierung des *Servlet-Contextes* von der Servlet-Engine benachrichtigt. Diese Information wird genutzt, um den eigentlichen Broker zu starten und um eine Referenz auf den Broker im `ServletContext` abzulegen. `Servlets` können dann über den `ServletContext` auf die Broker-Instanz zugreifen. Damit dies nicht bei jedem Aufruf der `service()`-Methode des `BrokerServlets` geschieht, wird einmalig bei der Initialisierung des `Servlets` (Aufruf der `init()`-Methode) eine Klassen-Variable initialisiert. Ein UML-Sequenzdiagramm des folgenden Quelltextfragments ist in Abbildung 6.1 zu sehen.

```
public class BrokerContextListener implements javax.servlet.ServletContextListener {

    public void contextDestroyed(ServletContextEvent sce){
        Broker b = (Broker) sce.getServletContext().getAttribute("Broker");
        b.services.shutdown();
```

```
    }

    public void contextInitialized(ServletContextEvent sce){
        // aus der web.xml
        String project = sce.getServletContext().getInitParameter("project");
        String home = sce.getServletContext().getInitParameter("home");

        home = "home="+home;
        StringTokenizer st = new StringTokenizer(project, ",");
        project = "project="+ (st.hasMoreTokens() ? st.nextToken() : "" );
        String[] args = {home, project};

        Broker b = new Broker(args);
        Configuration.init(b.properties);
        Iterator iter  = Configuration.instances();
        Configuration c = (iter.hasNext()? (Configuration) iter.next() : null);
        sce.getServletContext().setAttribute("Configuration", c);
        sce.getServletContext().setAttribute("Broker", b);
    }
}


public class BrokerServlet extends HttpServlet{

    protected Broker b;
    protected Configuration c;

    [...]

    public void init(ServletConfig servletConfig) throws ServletException {
        b = (Broker) servletConfig.getServletContext().getAttribute("Broker");
        c = (Configuration) servletConfig.getServletContext().getAttribute("Configuration");

        [...]
    }

    [...]
}
```

6.1.2. Parameter- und ValueBinding-Service

Entsprechend der in Abschnitt 5.2 beschriebenen Abbildung der `Parameter`- und `ValueBinding`-Eigenschaften auf die Architektur des Brokers wurden die neuen Services implementiert. Die Schnittstellen sind im Paket

`de.infoasset.broker.interfaces.solidProcesses`

und die zugehörigen Implementierungen im Paket

`de.infoasset.broker.services.solidProcesses`

zu finden.

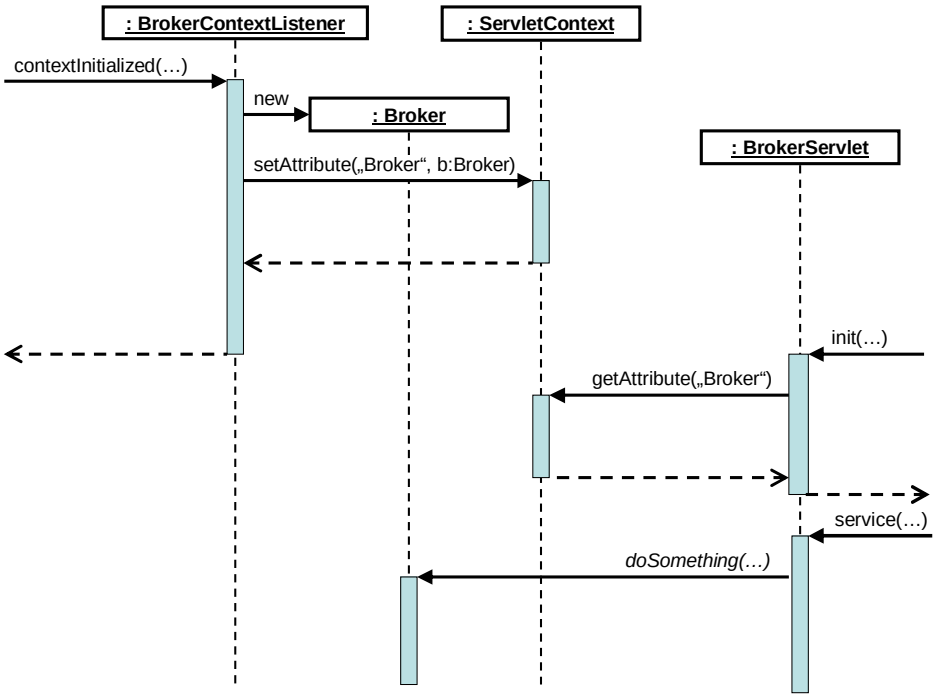


Abbildung 6.1.: Initialisierung der Broker-Web-Anwendung als Sequenzdiagramm

Da die `ValueBinding`-Klasse mehrere Datentypen für den zugeordneten Wert unterstützen muss, besitzt die Klasse entsprechend viele Attribute unterschiedlichen Typs. Alternativ zu dieser Implementierung hätte man auch für jeden Datentyp einen speziellen ValueBinding-Service entwerfen können oder den Wert in abstrakter Form als serialisiertes Objekt speichern können. Beide Alternativen haben jedoch Nachteile. Die Verteilung der Werte auf mehrere Klassen erhöht neben dem Aufwand auch den Schwierigkeitsgrad der Implementierung. Die zweite Variante schränkt Recherchemöglichkeiten stark ein, da die serialisierten Objekte nicht von der Datenbank interpretiert werden können. Falls jedoch die gewählte Implementierung bezüglich des verwendeten Speicherplatzes optimiert werden muss, ist die Verteilung auf mehrere Klassen vorzuziehen, da die Funktionalität für den Benutzer nicht eingeschränkt wird.

6.1.3. Erweiterung des Brokers um eine Unterstützung von Fließkommazahlen

Der Broker unterstützt in seiner ursprünglichen Version keine Fließkommazahlen. Da diese jedoch für eine Anwendung im Ingenieursbereich notwendig sind, wurde die Persistenzabstraktionsschicht erweitert bzw. angepasst. Vollständig implementiert wurden die Änderungen jedoch nur im Zusammenhang mit der MySQL-Datenbank. Bei der Verwendung von Fließkommazahlen zusammen mit anderer Datenbanken werden daher Fehlermeldungen geworfen.

Neben der Erzeugung einer `AttributeSignature`-Klasse für Fließkommatypen mussten hauptsächlich die Implementierungen der `ContentManager`-, `ContentContainer`- und

`Content`-Schnittstelle angepasst werden. Die Anpassungen fanden dabei auf unterschiedlichen Ebenen der Vererbungshierarchie statt und hatten neben der Funktionalität auch den Zweck, die Quelldateien übersetzbar zu halten. Da die Implementierung analog zu den bereits unterstützen Datentypen erfolgte, wird auf eine genauere Beschreibung verzichtet.

6.2. Benutzerschnittstelle

Die dynamischen Aspekte der Anwendungsfälle wurden durch eine Anpassung und Erweiterung der Interaktionsschicht realisiert. Die Interaktionsschicht (vgl. Abschnitt 4.1) besteht aus `Handler`-Klassen und zugehörigen Vorlagen (*Templates*). Das Resultat der Interaktion zwischen Handlern und Templates wird in der Benutzerschnittstelle sichtbar, die im folgenden Abschnitt erläutert wird. Das eigentliche Erstellung der *Templates* und `Handler` wird in den Abscnitten 6.2.2 und 6.2.3 beschrieben.

6.2.1. Benutzungsoberfläche

Die Unterstützung des CAPE wurde in die bestehende Oberfläche des INFOASSET BROKERS integriert. Dafür mussten sowohl vorhandene `Handler` und Templates bearbeitet als auch neue erstellt werden. Der neue CAPE-Dienst hat verschiedene Einstiegspunkte, die in die Menüleiste auf der linken Seite unter dem Punkt „CAPE“ integriert wurden (vergl. Abbildung 6.2). Da die Menüleiste während der Navigation durch die verschiedenen Seiten des Brokers nicht verändert wird, kann zu jedem Zeitpunkt in den Dienst eingestiegen werden.

Insgesamt kann man die Oberfläche des CAPE-Dienstes in zwei Bereiche einteilen. Der erste Bereich umfasst Seiten, die die Verwaltung von Parametern ermöglichen. Der andere Bereich ermöglicht die Navigation über verknüpfte Apparate, Modelle und Materialien sowie die assoziierten Simulationskontexte. Die Transitionen werden durch einen Klick-Ereignis aktiviert und durch die URL des Links sowie durch zusätzliche Parameter bestimmt. Für einzelne Operationen, wie die Zuordnung von Modellen oder Parametern zu einem Apparat wird zusätzlich die „Sammelmappe“ des Brokers verwendet.

Ein Ausschnitt der Navigationsmöglichkeiten des zweiten Bereichs wird in Abbildung 6.3 dargestellt. Die zugehörigen Screenshots sind in den Abbildungen 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10 und 6.11 zu sehen.

Um die Beziehungen zwischen den einzelnen Screenshots zu verdeutlichen, wurden einige Navigationsmöglichkeiten in die Bildbeschreibung aufgenommen. Dies ermöglicht dem Leser durch die Screenshots zu navigieren, ohne jedesmal zu einem beschreibenden Text umzublättern. Zusätzlich wird an dieser Stelle der Anwendungsfall für das „Bearbeiten eines Simulationskontextes“ beschrieben. Im konkreten Fall soll der Simulationskontext für die Apparat-Modell-Material-Kombination von „Kugelmühle“, „Prozessmodell nach Austin“ und „Hafenschlamm“bearbeitet werden.

Der Anwender meldet sich zunächst am System an und wird auf die persönliche Startseite (Abbildung 6.2) weitergeleitet. Von dort kann er entweder durch Suchen oder Navigieren über die Apparate-Verzeichnisstruktur (Abbildung 6.4) zum „Kugelmühle“-Dokument

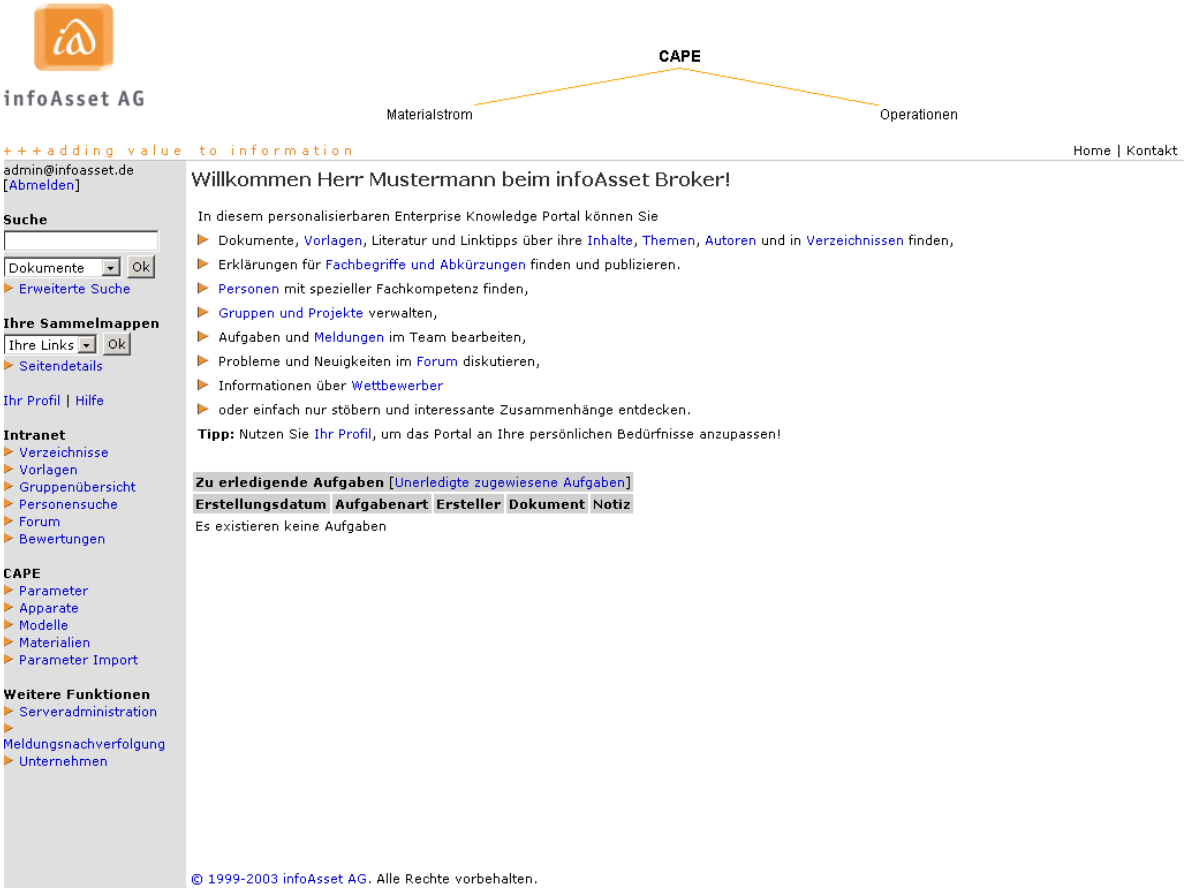


Abbildung 6.2.: GUI-Screenshot – Startseite mit CAPE-Menüeinträgen: Auf der linken Seite des Screenshots ist das Menü mit den CAPE-Einträgen zu sehen. Durch Anklicken des „Apparate“-Links gelangt man zum Apparateverzeichnis. (vgl. Abbildung 6.4).

gelangen (Abbildung 6.5). Durch Aufrufen des „anzeigen“-Links neben dem Prozessmodell werden die Simulationskontexte der entsprechenden Apparate-Modell-Kombination aufgelistet (Abbildung 6.7). Beim Klicken des „anzeigen“-Links neben dem Material „Hafenschlamm“ wird der zu bearbeitende Simulationskontext angezeigt (Abbildung 6.10). Die Aktualparameter des zugehörigen Apparats und des Modells können mit Hilfe des „bearbeiten“-Links bearbeitet werden (Abbildung 6.11).

Die Navigationsmöglichkeit für den Nutzer sowie die eigentlichen Operationen der Anwendung werden letztendlich durch Handler realisiert. Daher wird neben einem Zustandsdiagramm der Oberfläche auch eine Übersicht der Abfolgemöglichkeiten der sichtbaren und unsichtbaren Handler in Abbildung 6.12 dargestellt. Der für die Generierung der Oberfläche zuständige `Handler` wird i. Allg. durch das Anklicken eines entsprechenden Links bestimmt. Ausnahmen sind die ausgehenden Transitionen der unsichtbaren `Handler`. Sie werden aktiviert, sobald die `handleRequest()`-Methode des `Handlers` abgearbeitet wurde. Parameter haben – im Gegensatz zu den Übergängen der generierten Oberfläche – auf die Übergänge der Handler keinen Einfluss. Sie werden erst von den Handlern abgearbeitet. Aus Gründen

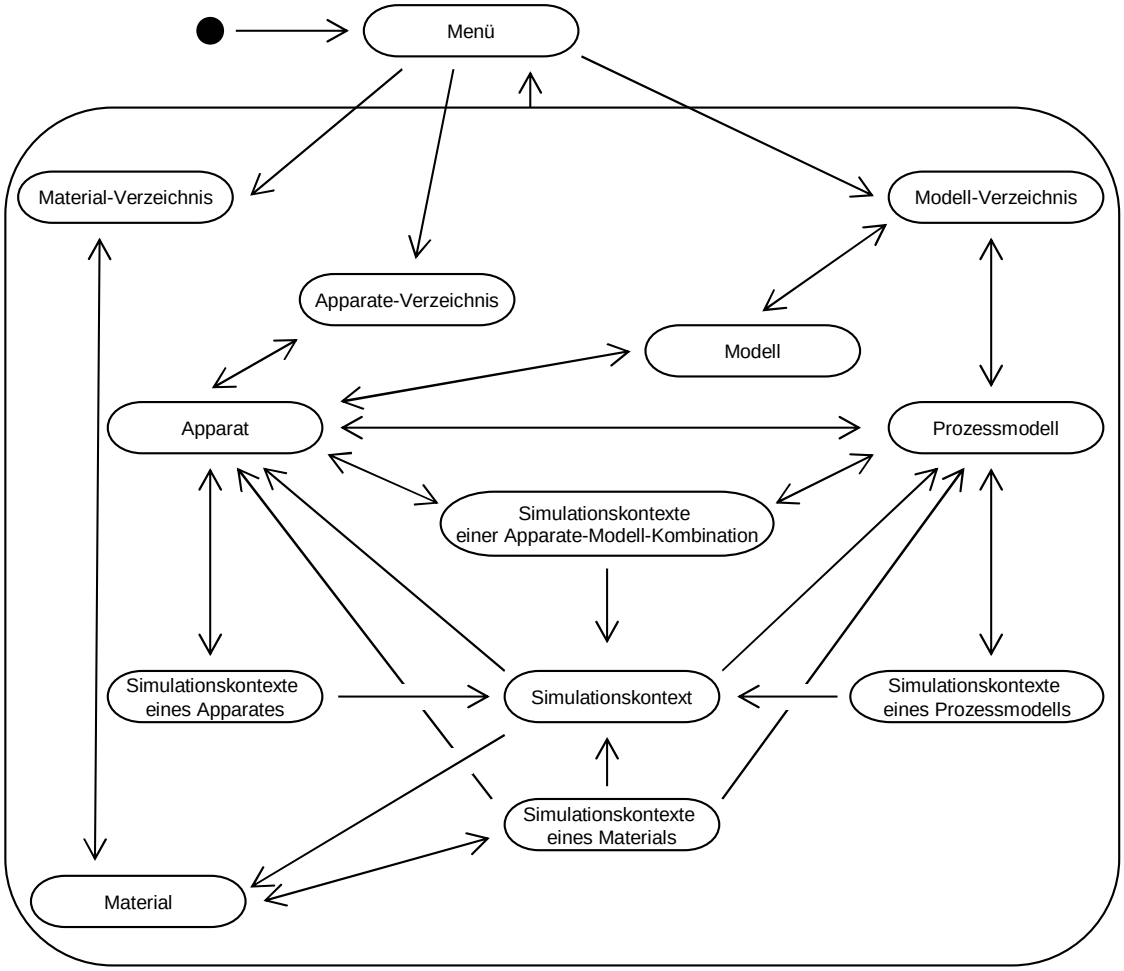


Abbildung 6.3.: UML-Zustandsdiagramm der grafischen Oberfläche

Abbildung 6.4.: GUI-Screenshot – Auflistung der Apparate eines Verzeichnisses: Die Abbildung zeigt das Apparate-Wurzelverzeichnis. Durch Anklicken des Dokuments „Kugelmühle“ gelangt man zu der entsprechenden Dokumentenansicht (vgl. Abbildung 6.5).

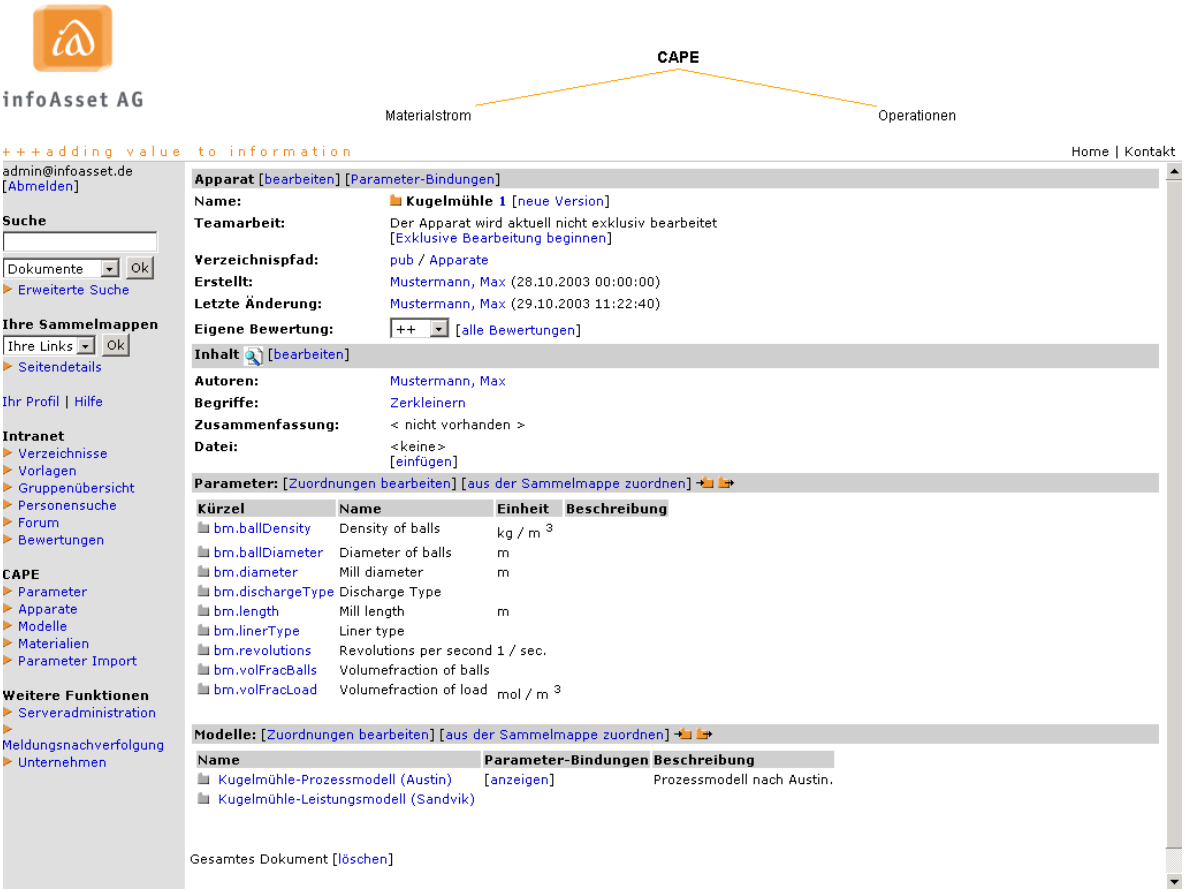


Abbildung 6.5.: GUI-Screenshot – Anzeige eines Apparats: Die Abbildung zeigt die Ansicht eines Apparatedokuments. Es werden die Formalparameter des Apparats sowie die zugeordneten Modelle aufgelistet. Beim Anklicken des Modellnamens wird das Modell in der Dokumentenansicht gezeigt (vgl. Abbildung 6.6). Durch Anklicken des „anzeigen“-Links neben dem „Kugelmühle-Prozessmodell“ werden alle Simulationskontexte der entsprechenden Apparate-Modell-Kombination aufgelistet (vgl. Abbildung 6.7).

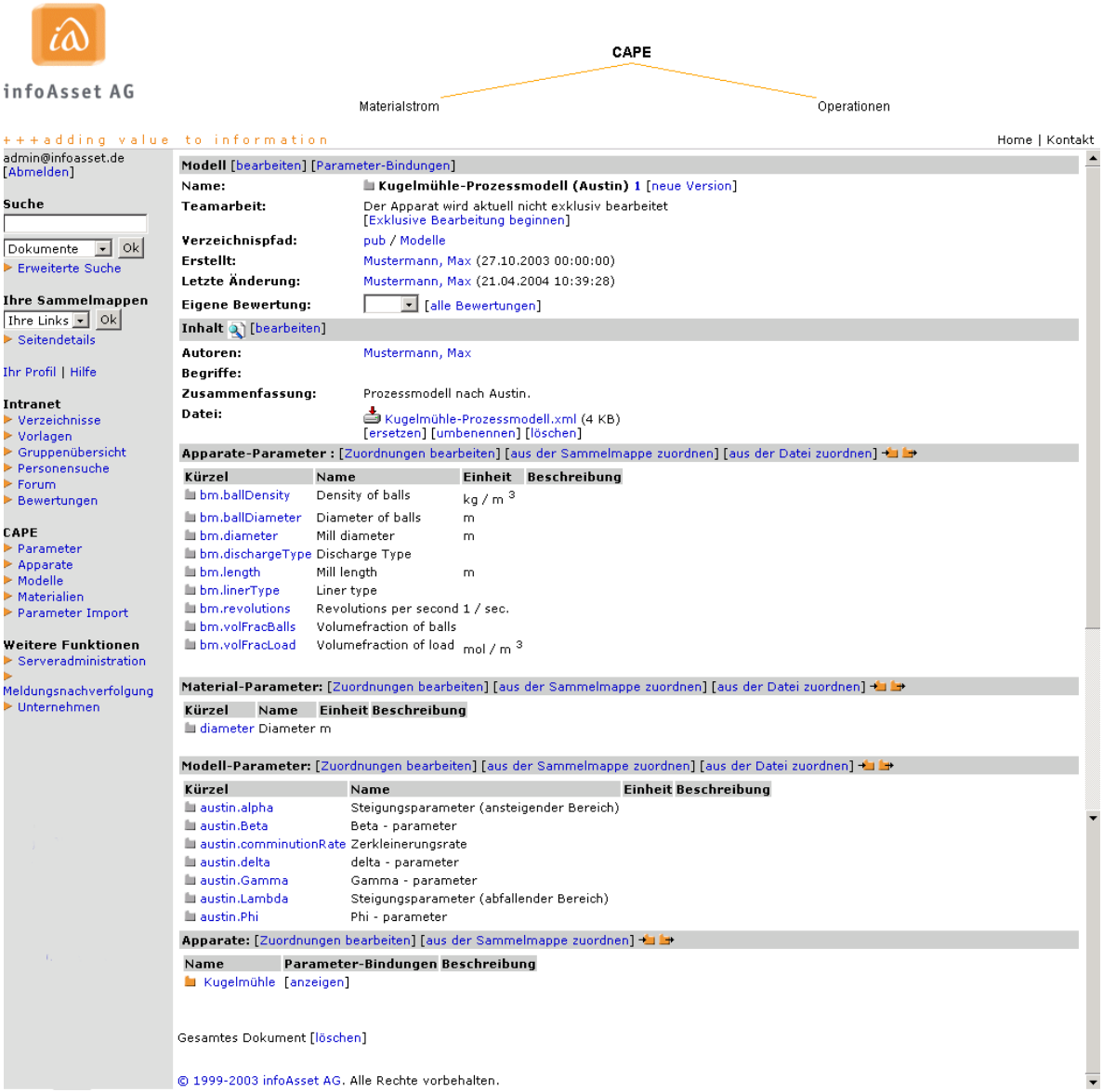


Abbildung 6.6.: GUI-Screenshot – Anzeige eines Modells: In der Abbildung wird ein Modell in der Dokumentenansicht gezeigt (vgl. Abbildungen 6.5 und 6.8). Es werden die verschiedenen Arten von Formalparametern aufgelistet. Außerdem sind die zugehörigen Apparate angegeben. Durch Anklicken des „anzeigen“-Links neben dem „Kugelmühle“-Modell gelangt man zu einer Auflistung der Simulationskontexte dieser Modell-Apparate-Kombination (vgl. Abbildung 6.7).

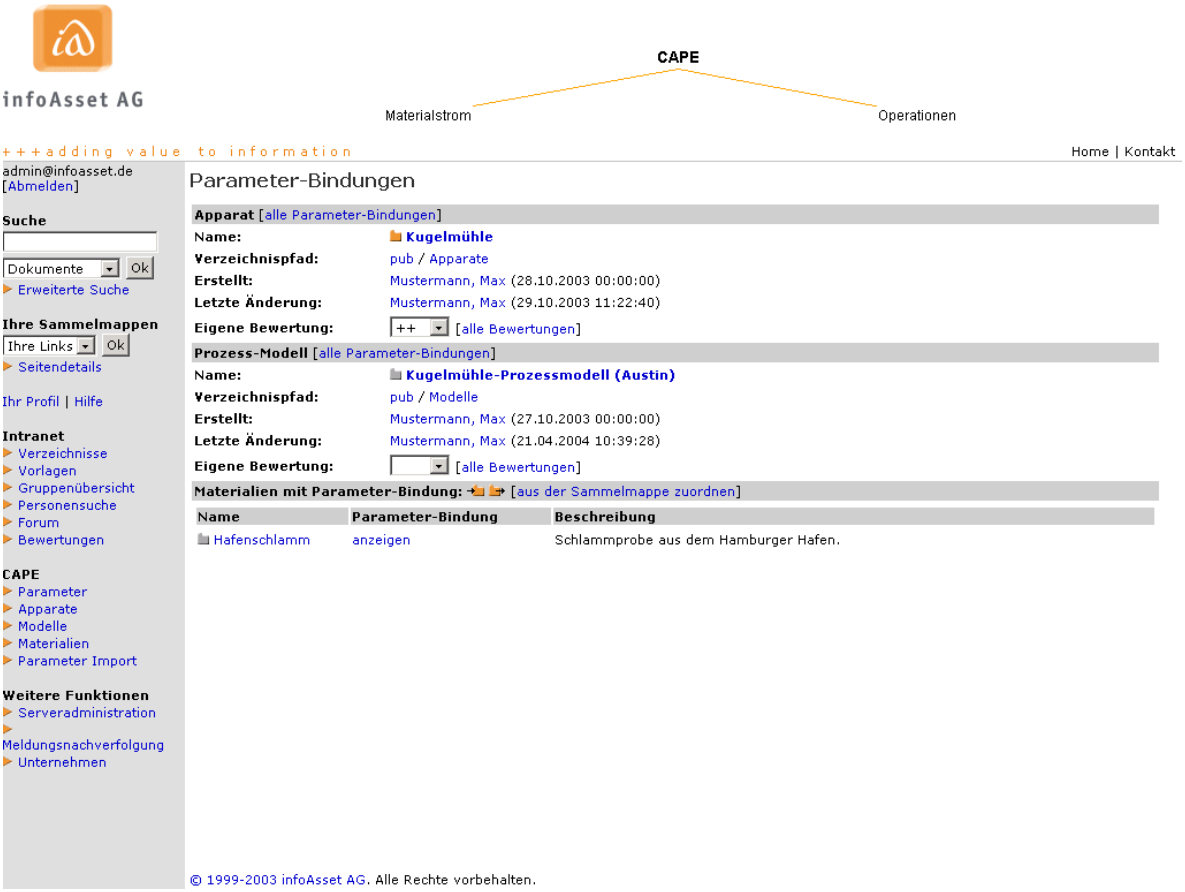


Abbildung 6.7.: GUI-Screenshot – Auflistung aller Simulationskontexte für eine gegebene Apparate-Modell-Kombination: Die Abbildung zeigt alle Simulationskontexte, die es für die Apparate-Modell-Kombination gibt. Da die Simulationskontexte keinen eigenen Namen besitzen, werden die Materialien des jeweiligen Kontextes aufgeführt. Durch Anklicken des Materialnamens gelangt man zu der Dokumentenansicht (vgl. Abbildung 6.8).

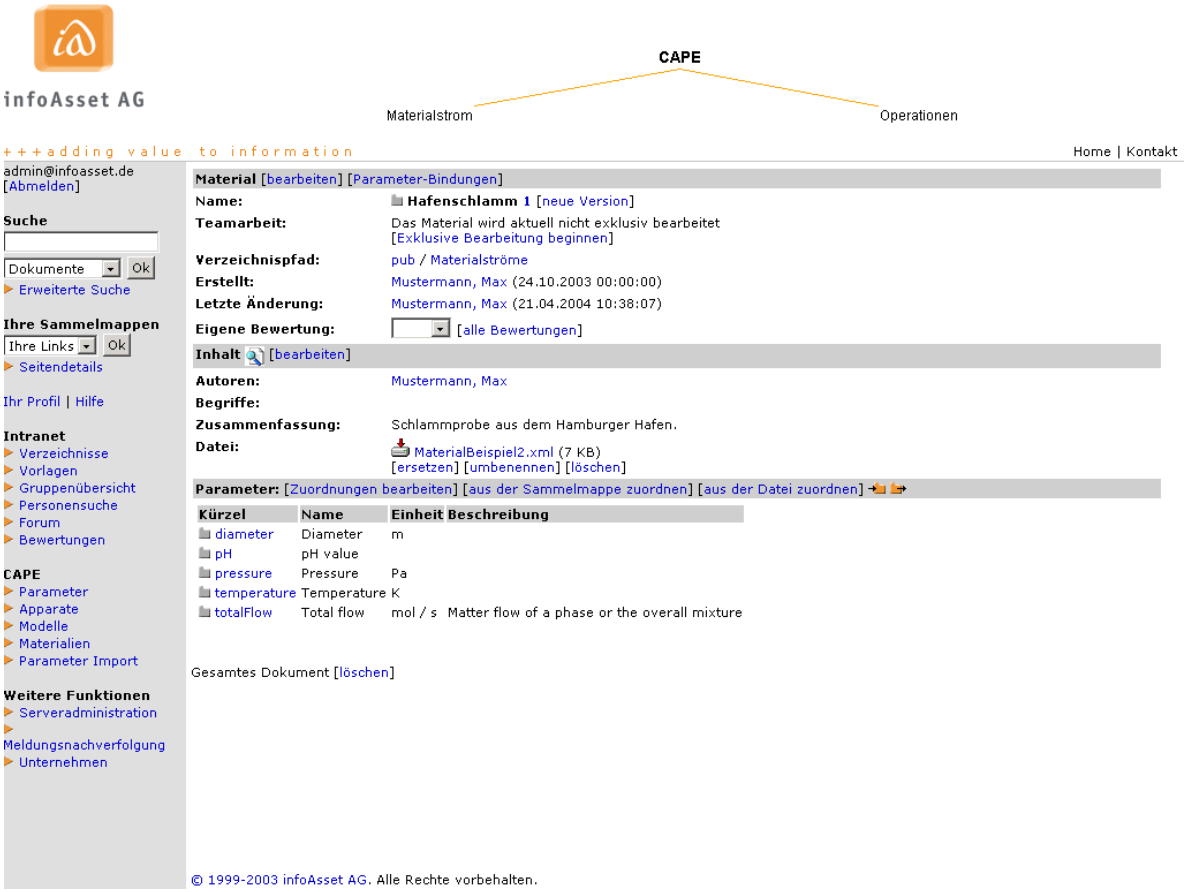


Abbildung 6.8.: GUI-Screenshot – Anzeige eines Materials: Die Abbildung zeigt ein Material in der Dokumenten-Ansicht. Es werden die Formal-Parameter des Materials aufgelistet. Eine Übersicht der vorhandenen Simulationskontexte für das Material erhält man durch Anklicken des „Parameter-Bindungen“-Links (verg. Abbildung 6.9).

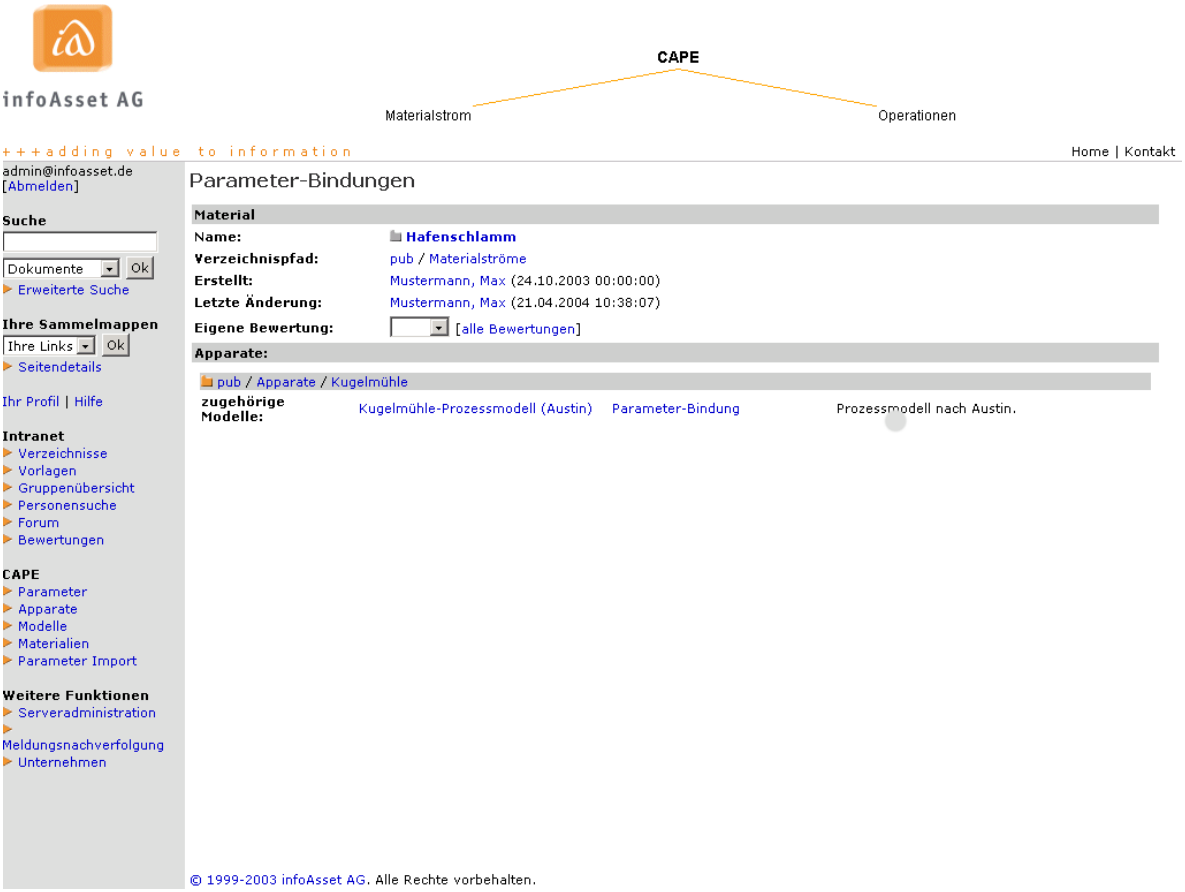


Abbildung 6.9.: GUI-Screenshot – Anzeige der Simulationskontexte eines gegebenen Materials: In dieser Ansicht werden die Simulationskontexte, die für das Material festgelegt wurden, aufgelistet. Durch Aktivieren des „Parameter-Bindungen“-Links wird der entsprechende Simulationskontext angezeigt (vgl. Abbildung 6.10).

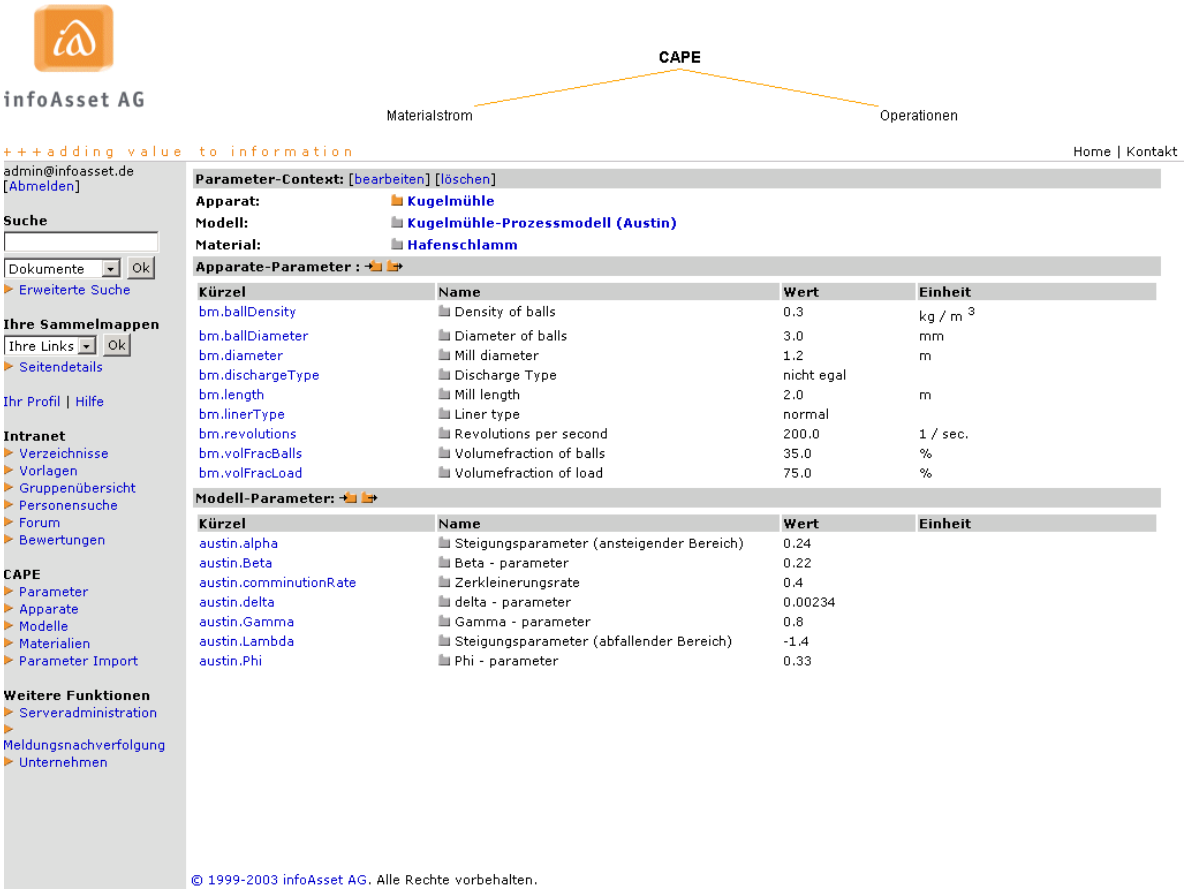


Abbildung 6.10.: GUI-Screenshot – Anzeige eines Simulationskontextes: Die Ansicht zeigt die Parameter-Bindungen eines Simulationskontextes. Die Materialeigenschaften (Aktualparameter) werden durch das Materialdokument vorgegeben. Die Aktualparameter des Apparats und des Modells sind Teil der Eigenschaften des Simulationskontextes und können daher bearbeitet werden (vgl. Abbildung 6.11).

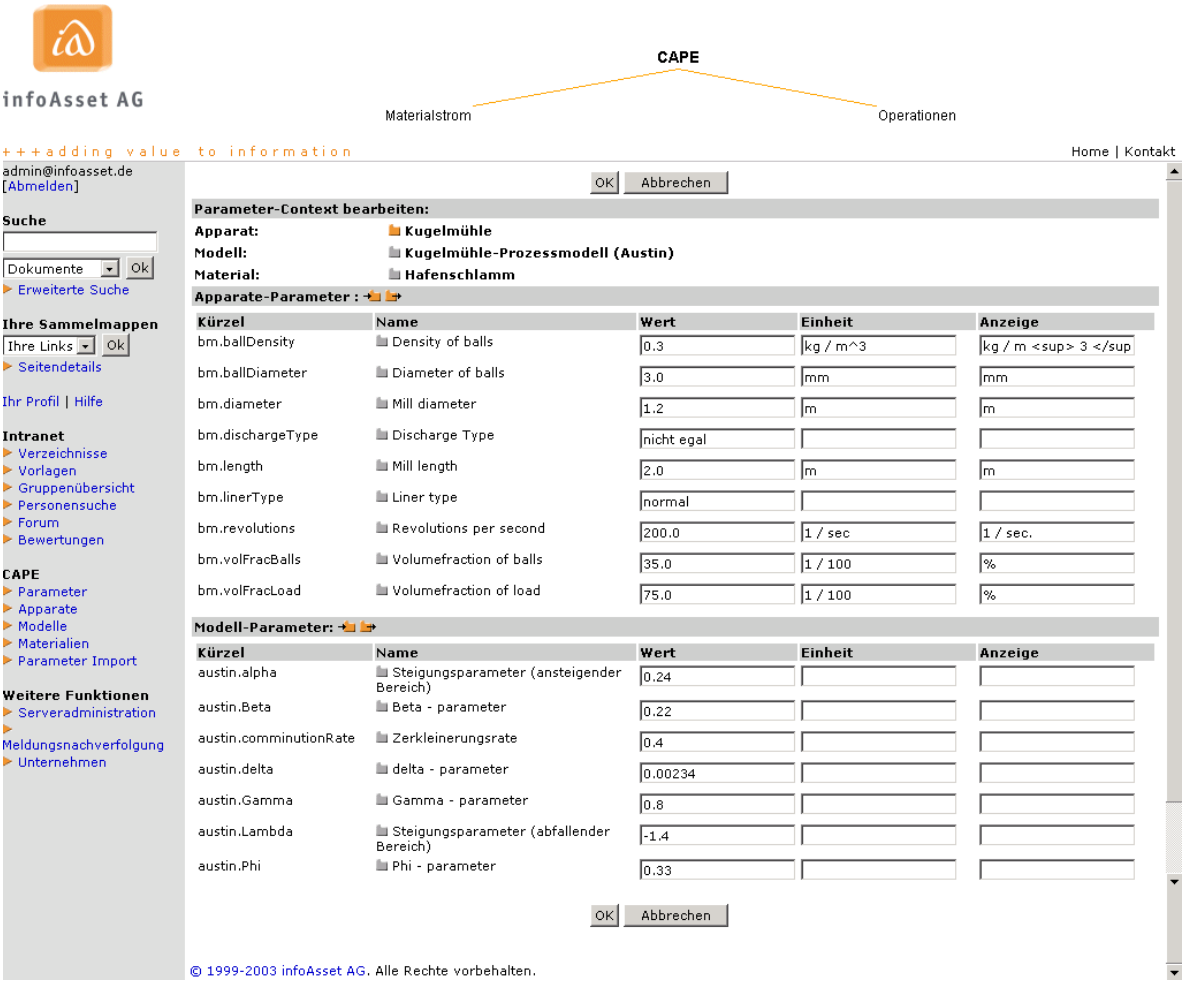


Abbildung 6.11.: GUI-Screenshot – Bearbeiten eines Simulationskontextes: Es wird das Eingabeformular für das Editieren eines Simulationskontextes gezeigt.

der Übersichtlichkeit wurde die Beschriftung der Transitionen weggelassen. Die Namen der unsichtbaren `Handler` sind in der Abbildung kursiv geschrieben, um sie von den sichtbaren zu unterscheiden.

6.2.2. Erstellung der Templates

Templates bilden den statischen Teil der Benutzungsoberfläche. Sie sind zum größten Teil in HTML verfasst und bestimmen das Layout der Nutzerschnittstelle. Für jede im Web-Browser angezeigte Seite existiert eine entsprechende Vorlage. Die für das Projekt erstellten bzw. angepassten Vorlagen sind in den Tabellen 6.1, 6.2 und 6.3 mit ihrem jeweiligen Zweck aufgeführt. Neben den in Abschnitt 4.1 beschrieben Ersetzungsausdrücken zeichnen sich die Templates durch Verwendung von JavaScript für die Verlinkung, wie im folgenden Beispiel dargestellt, aus:

```
<a href="javascript:s.show('simContext/$m.simContextId$.htm')">
```

Durch die Verwendung der `s.show()`-Methode werden Session-Informationen, wie z. B. die Session-ID, in die nächste HTTP-Anfrage kodiert. Anhand der aufgerufenen URL identifiziert der Broker mittels einer LookUp-Tabelle den zugehörigen `Handler`. Einträge der Tabelle sind in der `handler.txt`-Datei gespeichert und können z. B. folgendermaßen aussehen:

```
parameter/*=de.infoasset.broker.handler.parameters.ParameterHandler
parameters/create=de.infoasset.broker.handler.parameters.CreateHandler
```

Falls der Dokumenteneintrag mit '*' endet, ist die angegebene Klasse für alle URLs zuständig, die mit dem entsprechenden Pfad beginnen. Neben sichtbaren `Handlern` gibt es auch für den Anwender unsichtbare `Handler`. Sie haben häufig die Aufgabe, Daten eines HTML-Formulares zu analysieren und entsprechend Aktionen auf der Service-Ebene auszuführen. Die grafische Ausgabe delegieren sie dann an einen sichtbaren `Handler`. Eine Verkettung mehrerer unsichtbarer `Handler` vor der Ausgabe ist möglich.

6.2.3. Erstellung der Handler

Die `Handler`-Klassen sind Subklassen der `GenericHandler`-Klasse. Neben einigen Hilfsmethoden, die bei der Programmierung von Handlern häufig benötigt werden, definiert die Superklasse hauptsächlich die `handleRequest()`-Methode. Diese Methode muss überschrieben werden, um die Anfrage eines Clients zu bearbeiten. Das aktuelle `Session`-Objekt, der für die Ausgabe benötigte `OutputStream` sowie die ID des aufgerufenen Dokuments werden der Methode als Argumente übergeben.

Bei der Implementierung der `handleRequest()`-Methode sind einige Aufgaben zu unterscheiden, die im Folgenden beschrieben werden.

1. **Erzeugen eines `Template`-Objektes:**
   Der `Handler` greift über ein `Template`-Objekt auf eine Vorlagendatei zu. Im Normalfall wird die richtige Vorlagendatei direkt aus der vom Client aufgerufenen URL entnommen und im Konstruktor der `Template`-Klasse übergeben.

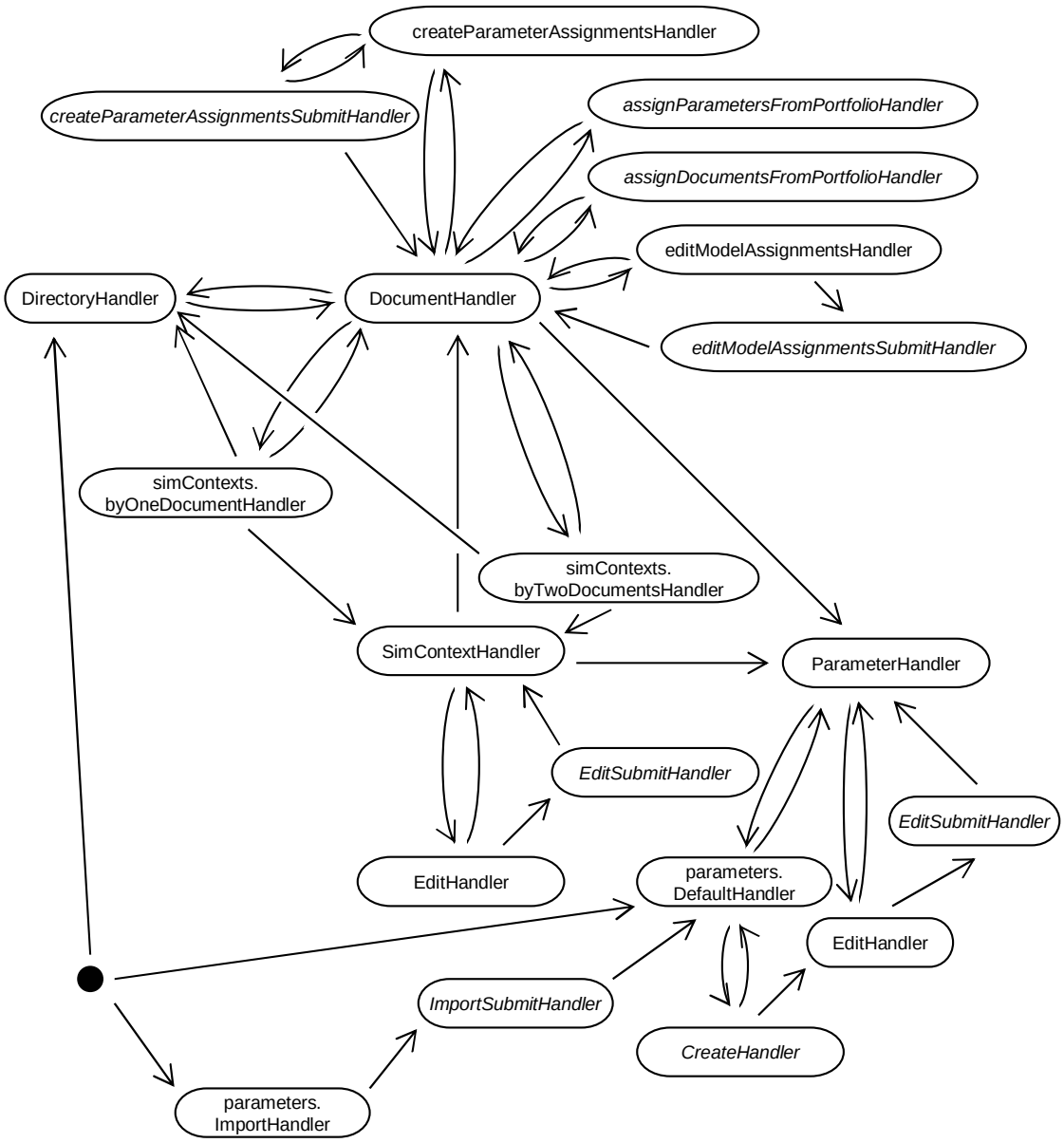


Abbildung 6.12.: Abfolgemöglichkeiten der Handler

```
Template template = new Template(documentIdString);
```

Bei Dokument- und Verzeichnis-Handlern wird zusätzlich der Typ des Dokuments berücksichtigt. So ist es z. B. möglich, für Apparate und Modelle jeweils eine Typspezifische Vorlage zu benutzen. Der Dateiname der speziellen Vorlage setzt sich aus dem normalen Dokumentnamen, einem Unterstrich '_' und dem Typschlüssel zusammen. Falls für ein Dokument keine spezielle Vorlagendatei existiert, wird anhand der Dokumenttyp-Hierarchie die nächste, möglichst spezifische Vorlage gewählt. Das entsprechende Java-Fragment sieht folgendermaßen aus:

```
Template template = new Template(getDocumentIds(
                      "documents/document",
                      document.getDocumentKind())
                    );
```

2. **Definition der Ersetzungen für ein Template:**
   Damit ein `Handler` die Ersetzungsausdrücke (vgl. Abschnitt 4.1) in einem Template mit passenden Textfragmenten ersetzen kann, müssen Ersetzungen definiert werden. Die unterschiedlichen Ersetzungsarten werden im Folgenden kurz vorgestellt [Jac02]:

   - Die einfache textuelle Ersetzung erfolgt folgendermaßen:

     ```
     template.put(placeholderString, new PrintSubstitution(){
       public void print( Session session, PrintStream p) {
         p.print(replacementString);
       }
     });
     ```

   - Für die bedingte Ersetzung wird folgender Ausdruck verwendet:

     ```
     template.put(placeholderString, new ConditionalTemplate(){
       public boolean test(Session session) {
         return replacementBoolean;
       }
     });
     ```

   - Die Ersetzung von Listenplatzhaltern erfolgt über folgenden Java-Ausdruck:

     ```
     template.put(placeholderString, new ListTemplate(){
       Iterator iterator = null;
       Element element = null;
       public void start(){
         iterator = ...;
         //Zuweisung der Menge, auf der operiert werden soll
       }

       public boolean hasNext(){
         return iterator.hasNext();
       }

       public void next(){
         element = iterator.next();
       }

       public void init(){
         /* Definition von Ersetzungen für einen Schleifendurchlauf
          */
     ```

```
  }
});
```

Das `ListTemplate` basiert auf dem *Iterator*-Architekturmuster [GHJV94]. In der `start()`-Methode wird eine Menge von Entitäten erzeugt, über die iteriert werden soll. In der Regel wird die Menge jedoch nicht durch eine `Collection` oder ein `Array` beschrieben, sondern wiederum durch einen `Iterator`. Dies hat den Vorteil, dass die `next()`- und `hasNext()`-Methodenaufrufe einfach an die entsprechenden Methoden des erzeugten `Iterators` weitergeleitet werden können. In der `init()`-Methode können Ersetzungen definiert werden, die abhängig vom jeweiligen Schleifendurchlauf unterschiedliche Ergebnisse liefern.

3. **Erzeugen der Ausgabe:**
   Bei den sichtbaren Handlern wird am Ende der `handleRequest()`-Methode durch die Instanzierung eines Templates die Ausgabe erzeugt und in den `OutputStream` geschrieben.

   ```
   template.instatiate(session, output);
   ```

4. **Weiterleiten einer Anfrage:**
   Im Gegensatz zu den sichtbaren `Handlern` leiten die unsichtbaren `Handler` Anfragen des Clients an andere `Handler` weiter. Daher wird am Ende der `handleRequest()`-Methode auch kein `Template` instanziiert, sondern die `forwardRequest()`-Methode des `Session`-Objektes aufgerufen. Als Argument müssen der `OutputStream` und die ID des nächsten Dokuments übergeben werden.

   ```
   session.forwardRequest(output, nextDocumentString);
   ```

5. **Zugriff auf einen Parameter des HTTP-Requests:**
   Wie bereits erwähnt, wird das `Session`-Objekt des Nutzers beim Aufruf der `handleRequest()`-Methode übergeben. Durch den Aufruf der `getRequestParameter()`-Methode mit dem Namen des Parameters als Argument kann auf den Wert des entsprechenden Request-Parameters zugegriffen werden. Es macht dabei keinen Unterschied, ob der Parameter mit der `Post`- oder `Get`-Methode des HTTP übermittelt wurde. Neben den Request-Parametern stellt das `Session`-Objekt auch einige andere nützliche Informationen bereit, beispielhaft kann auf den aktuellen Nutzer oder die Service-Schnittstelle zugegriffen werden.

   ```
   String valueString = session.getRequestParameter(nameString);
   ```

6. **Parameterübergabe zwischen `Handlern`:**
   Mit Hilfe des Session-Objekts können Parameter auch zwischen `Handlern` übergeben werden. Dies ist z. B. bei der Verwendung unsichtbarer `Handler` notwendig. Die entsprechenden Parameternamen und -Werte werden durch Aufruf der `putRequestParameter()`-Methode dem `Session`-Objekt mitgeteilt und können vom nächsten `Handler` wie beschrieben abgefragt werden.

   ```
   session.putRequestParameter(nameString, valueString);
   ```

Tabelle 6.1.: Templates für die Bearbeitung von Dokumenten

Template	**Funktion**
document_apparatus.htm	Das Template ist für die Anzeige eines Apparatedokuments zuständig. Es werden die zugeordneten Apparateparameter und Modelle bzw. Prozessmodelle aufgelistet.
document_model.htm	Das Template ist für die Anzeige eines Modelldokuments zuständig.
document_material.htm	Das Template ist für die Anzeige eines Materialdokuments zuständig. Es werden die zugehörigen Materialparameter aufgelistet.
preview_xml.htm	Das Template ermöglicht die Vorschau von XML-Dateien. Für jeden Subtyp kann eine Typ-spezifische Vorschau durch eine XSLT-Dateien definiert werden.
editModelAssignments.htm	Das Template dient zur Bearbeitung der zugeordneten Modelle bzw. Dokumente.

Tabelle 6.2.: Templates für die Bearbeitung von Parametern

Template	**Funktion**
parameter.htm	Das Template ist für die Anzeige eines einzelnen Parameters zuständig.
default.htm	Das Template dient zur Anzeige aller definierten Parameter des Systems.
createParameterAssignments.htm	Das Template enthält ein Formular für die Zuordnung von Parametern zu einem Dokument.
edit.htm	Das Template dient zum Bearbeiten von Parametern.
delete.htm	Das Template fragt nach, ob ein Parameter wirklich gelöscht werden soll.
import.htm	Das Template ist für den Import von Parameterdefinitionen aus einer XML-Datei.

Tabelle 6.3.: Templates für die Bearbeitung von Simulationskontexten

Template	**Funktion**
byOneDocument.htm	Das Template listet zu einem Dokument (Apparat, Modell oder Material) die vorhandenen Simulationskontexte auf.
byTwoDocuments.htm	Das Template listet zu zwei gegebenen Dokumenten unterschiedlichen Typs die vorhandenen Simulationskontexte auf.
edit.htm	Das Template dient zur Bearbeitung von Parameterwerten eines Simulationskontextes.
simContext.htm	Das Template zeigt einen Simulationskontext inklusive seiner Parameterwerte an.

6.3. Service-Schnittstelle

Die Implementierung der SOAP-basierten Service-Schnittstelle ist mit den Werkzeugen des APACHE AXIS Projektes recht einfach. Der in Abschnitt 5.3 beschriebene Entwurf wurde direkt übernommen. Problematisch bei der Implementierung der Schreiboperationen war die Deserialisierung von Objekten durch Axis. Die Problematik bestand darin, dass die verwendeten Proxy-Klassen wie z. B. `WSApparat` Lese- und Schreibzugriffe zu iAssets weiterleiten. Bei der Deserialisierung durch die `BeanDeserializer`-Klasse des Axis-Systems wird die Proxy-Klasse durch Aufrufen des Standardkonstruktors instanziert, wie im folgenden Quelltextfragment zu sehen ist:

```
// Construct BeanDeserializer for the indicated class/qname and meta Data
public BeanDeserializer(Class javaType, QName xmlType, TypeDesc typeDesc, Map propertyMap){
  this.xmlType = xmlType;
  this.javaType = javaType;
  this.typeDesc = typeDesc;
  this.propertyMap = propertyMap;

  // create a value
  try {
    value=javaType.newInstance();
  } catch (Exception e) {
    [...]
  }
}
```

Da zu diesem Zeitpunkt die iAssetID der *Adaptee*-Klasse [GHJV94] noch nicht bekannt ist, wird die Proxy-Klasse nicht vollständig initialisiert und spätere Aufrufe der `Get-` und `Set-`Methoden können nicht delegiert werden.

Dieses Problem kann auf unterschiedliche Weisen gelöst werden. Beim konsequentesten Ansatz wird die Axis-Architektur durch einen speziellen `AssetDesirializer` erweitert. Dieser Ansatz war für den Rahmen der Diplomarbeit jedoch zu aufwändig. Der gewählte Ansatz benutzt stattdessen interne Attribute als Zwischenspeicher. Beim späteren Abarbeiten der entsprechenden Service-Methode wird dann die Initialisierung nachgeholt und die bearbeiteten Attribute an das *Adaptee*-iAsset weitergeleitet. Ein schematisches Sequenzdiagramm ist in Abbildung 6.13 zu sehen.

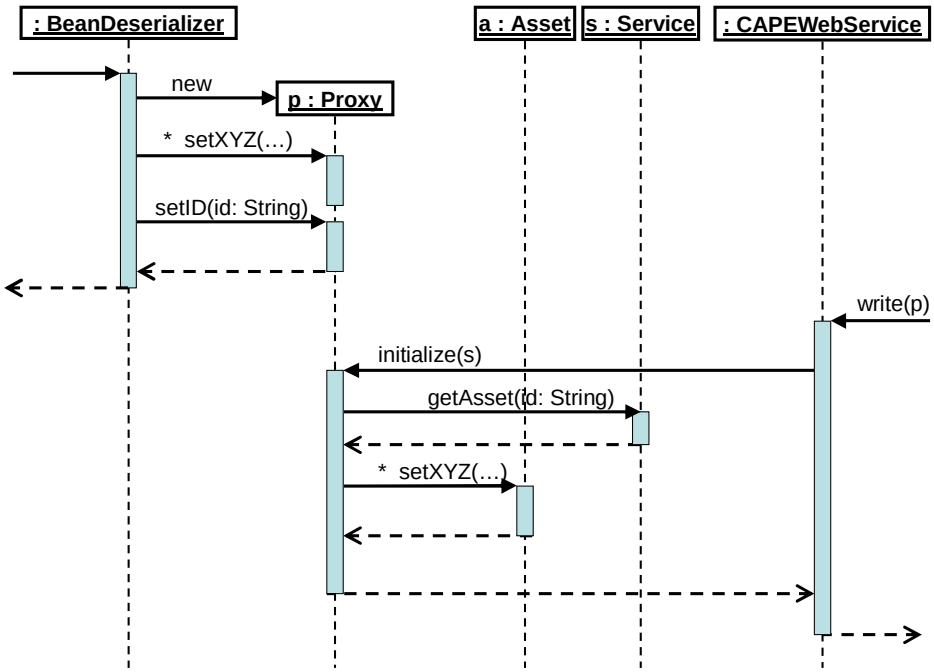


Abbildung 6.13.: Schematisches Sequenzdiagramm von schreibenden Operationen der Service-Schnittstelle

7. Zusammenfassung und Ausblick

In diesem Kapitel wird die Arbeit zusammengefasst und ein Ausblick auf mögliche Weiterentwicklungen gegeben.

7.1. Zusammenfassung

In der Arbeit wurde zunächst das assetorientierte Entitäten-Metamodell vorgestellt. Dieses Modell basiert auf den Erkenntnissen von Peirce und Cassirer. Der statische Teil des Modells trennt zwischen einer medialen Wahrnehmung und einem konzeptuellen Verständnis auf unterschiedlichen Beschreibungsebenen von Entitäten. Der dynamische Teil fordert eine offene und dynamische System-Architektur. Dieses Modell wurde bei der Analyse des CAPE angewandt, um die Praxistauglichkeit dieses Ansatzes exemplarisch zu prüfen.

Die im CAPE verwendeten Inhalte bestehen oftmals aus komplizierten Datenstrukturen, wie z. B. den Beschreibungen von Materialströmen. Diese Formate sind nur mit einer speziellen Visualisierung für den Endbenutzer verständlich. Daher ist eine zweigeteilte Sichtweise, wie sie im statischen Teil des assetorientierten Entitäten-Metamodell enthalten ist, sinnvoll. Durch die Trennung von Inhalt und Konzept können einerseits die Bedürfnisse des menschlichen Benutzers und andererseits die Anforderungen spezieller Anwendungssoftware wie z. B. SOLIDSIM erfüllt werden. Eine übersichtliche und verständliche Darstellung der Inhalte für den Nutzer wird durch die konzeptuelle Sichtweise erreicht, während die Anwendungssoftware die Inhalte direkt verarbeiten kann.

Wissensmanagement im Bereich des CAPE erfordert ein grundlegendes Verständnis des methodischen Vorgehens in der Prozessentwicklung sowie zusätzlich Verständnis von systemtechnischen und naturwissenschaftlichen Zusammenhängen. Wissen wird bei der Entwicklung von Feststoffprozessen nicht nur in der Synthese von Prozessen, sondern auch bei der Analyse benötigt. Diese Eigenschaft ist in Bezug auf das Wissensmanagement der wesentliche Unterschied zu Fluidprozessen. Durch die Komplexität der Datenstrukturen ist es schwierig, ein fertiges konzeptuelles Modell zu entwickeln, das sich als Grundlage für hochwertige Recherche-Möglichkeiten anbietet. Aus Sicht des Wissensmanagements ist daher die Entwicklung eines passenden Modells noch nicht abgeschlossen. Dies zeigt, dass Ansätze des Wissensmanagements, die zwischen dem Metawissensprozess und dem eigentlichem Wissensprozess unterscheiden, Vorteile gegenüber einer rein wissensorientierten Sichtweise bieten.

Die Trennung von Inhalt und Konzept legt auch den Grundstein für eine erfolgreiche Trennung der beiden Wissensprozess-Typen. Die eigentliche Dynamik innerhalb der Prozesse wird durch die Unterstützung von Offenheit und Dynamik im assetorientierten Entitäten-Metamodell gewährleistet. Daher ist der assetorientierte Ansatz auch unter dem Gesichtspunkt des Wissensmanagements ein geeignetes Modell.

Diese Betrachtungen waren bisher losgelöst von der verwendeten Systemplattform. Bei der Implementierung hat sich gezeigt, dass Die Umsetzung des Modells auf Basis des Brokers möglich ist. Da aber besonders die Unterstützung von Offenheit und Dynamik fehlen, geht ein wesentlicher Vorteil des assetorientierten Ansatzes, nämlich die fortlaufende Unterstützung des Metawissensprozesses, verloren.

Auch die Wahl der anderen Komponenten des Systems hat während der Entwicklung überzeugt. Es ist jedoch anzumerken, dass bisher keine Erfahrungen im Produktivbetrieb mit der Kombination von TOMCAT, INFOASSET BROKER, AXIS und MYSQL vorliegen.

7.2. Ausblick

Der entwickelte Prototyp unterstützt in erster Linie die Analyse von Feststoffprozessen, indem Simulationskontexte verwaltet werden können. Da jedoch eine ganzheitliche Unterstützung des Entwicklungsprozesses sinnvoll ist, wäre eine entsprechende Weiterentwicklung zu begrüßen.

Dabei bilden der komponentenorientierte Aufbau von verfahrenstechnischen Anlagen (vgl. Abschnitt 3.1.2) und die Beschreibung durch Fließschemadiagramme die Grundlage für die Dokumentation einzelner Entwurfsentscheidungen im Entwicklungsprozess. Durch Methoden der *Knowledge Discovery* und des *Data Minings* können die gesammelten Daten zur Verbesserung von Simulationsmodellen dienen. Außerdem können durch Verfahren des *Case-Based-Reasoning*s Entwurfsentscheidungen unterstützt werden. Durch die Extrapolation von Simulationsdaten auf andere Anwendungsfälle kann auch die Simulation von Feststoffprozessen weiter verbessert werden.

Eine vollständige Unterstützung des Entwicklungsprozesses beinhaltet auch eine standardisierte Schnittstelle für die Tool-Integration. Unter diesem Gesichtspunkt ist daher die Verwendung von CORBA als Middleware-Technologie, wie sie auch im CAPE-OPEN-Standard beschrieben ist, denkbar.

Neben den verfahrenstechnischen Erweiterungsmöglichkeiten sind auch zahlreiche Erweiterungen der Broker-Plattform wünschenswert. Zunächst ist eine automatische Abbildung des Asset-Modells auf das iAsset-Modell sinnvoll. Darüber hinaus sollten auch die dynamischen Fähigkeiten des Brokers erweitert werden. Diese Erweiterungen würden insbesondere die evolutionäre Weiterentwicklung des fachlichen Entitätenmodells erleichtern. Unabhängig von diesem Projekt wäre eine bessere Integration des Brokers in die Tomcat Servlet-Engine zu begrüßen. Dazu gehört eine gemeinsam genutzte Session-Verwaltung auf Basis der Servlet-Engine sowie eine gemeinsame Nutzerverwaltung auf Basis des Brokers. Außerdem sollte die Bearbeitung von XML-Dateien über eine HTML-Schnittstelle ermöglicht werden.

A. Konzeptuelles Modell der Anwendungsdomäne (ADL)

```
1  model CAPE;
2
3  class Apparat{
4    content apparatDefinition: ApparatXML;
5
6    concept characteristic name: String;
7      characteristic description: String;
8
9      relationship models: Model[*] <-> Model.apparats;
10     relationship parameters: Parameter[*];
11     relationship contexts: SimulationContext[*]
12       <-> SimulationContext.apparat;
13 }
14
15 class Model{
16   content modelDefinition: ModelXML;
17
18   concept characteristic name: String;
19     characteristic description: String;
20     characteristic valueKind: Domain[ModelKind];
21     characteristic materialParameters: Parameter[*];
22     characteristic modelParameters: Parameter[*];
23     characteristic apparatParameters: Parameter[*];
24
25     relationship apparats: Apparat[*] <-> Model.models;
26     relationship contexts: SimulationContext[*]
27       <-> SimulationContext.model;
28
29     constraint materialParameters.parameterKind = ‘material‘;
30     constraint modelParameters.parameterKind = ‘model‘;
31     constraint apparatParameters.parameterKind = ‘apparat‘;
32 }
33
34 class Material{
35   content materialFeed: MaterialXML;
36
37   concept characteristic name: String;
38     characteristic description: String;
39     characteristic parameters: Parameter[*];
40
41     relationship contexts: SimulationContext[*]
42       <-> SimulationContext.realInMaterialStream;
43
44     constraint materialParameters.parameterKind = ‘material‘;
45 }
46
```

```
47 class SimulationContext{
48   content simContext: SimulationContextXML;
49
50   concept characteristic values: ValueBinding[*];
51
52     relationship model: Model[1] <-> Material.contexts;
53     relationship apparat: Apparat[1] <-> Apparat.contexts;
54
55     relationship realInMaterialStream: Material[1]
56       <-> Material.contexts;
57     relationship realOutMaterialStreams: Material[*];
58     relationship simOutMaterialStreams: Material[*];
59 }
60
61 class Person{
62   content portrait: Image;
63     workplace: Image;
64
65   concept characteristic email: String;
66     characteristic firstName: String;
67     characteristic lastName: String;
68
69     relationship authorOfFile: File[*] <-> File.author;
70     relationship memberships: Membership[1] <-> Membership.person;
71
72     constraint behavior accessControl;
73     constraint behavior user;
74     constraint unique email;
75 }
76
77 class Membership{
78   content memberCard: ImageFile;
79
80   concept characteristic memberSince: Date;
81     characteristic role: Domain[GroupRole];
82
83     relationship person: Person[1];
84     relationship group: Group[1];
85 }
86
87 class Group{
88   content groupImage: Image;
89
90   concept characteristic name: String;
91     characteristic description: String;
92     characteristic since: Date;
93
94     realtionship members: Membership[*] <-> Membership.group;
95 }
96
97 contenttype Base{
98   container filecontainer;
99
100  concept characteristic:
```

```
101     characteristic name: String;
102     characteristic mimeType: String;
103     characteristic size: Integer;
104     characteristic path: String;
105     characteristic creation: Date ;
106     characteristic lastModification: Date;
107     characteristic authorNames: String;
108
109     relationship authors: Person[*] <-> Person.authorOfFile;
110     constraint behavior accessControl;
111 }
112
113 contenttype Image refines Base{
114   concept characteristic dpi: Integer;
115     characteristic width: Integer;
116     characteristic hight: Integer;
117     characteristic colourspace: String;
118     characteristic colourDepth: Integer;
119 }
120
121 contenttype Text refines Base{
122   concept characteristic encoding: String;
123     characteristic linebreak: String;
124     characteristic whitespace: String;
125     characteristic nOfWords: Integer;
126     characteristic nOfLines: Integer;
127
128     constraint behavior fulltextIndex;
129 }
130
131 contenttype XML refines Text{
132   concept characteristic schema: String;
133     characteristic DTD: String;
134     characteristic schemaFileURL: URL;
135 }
136
137 contenttype ApparatXML refines XML{
138   concept characteristic schema: String = 'apparatus.xsd';
139 }
140
141 contenttype ModelXML refines XML{
142   concept characteristic schema: String = 'model.xsd';
143 }
144
145 contenttype MaterialXML refines XML{
146   concept characteristic schema: String = 'material.xsd';
147 }
```

B. Mediales Modell der Anwendungsdomäne (XML-Schema)

Schema: CAPEApparatus.xsd

Elements	*Complex types*
apparat	apparat
	properties

Schema: CAPEMaterial.xsd

Elements	*Groups*	*Complex types*
materialFeed	intervalAndProperty	compoundRef
	quantityAndProperty	compounds
		dimensions
		distributedProperties
		fraction
		fractions
		interval
		material
		materialFeed
		phase
		phases
		propertyBindings
		quantity
		stream

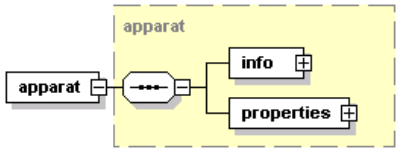
Schema: CAPEModel.xsd

Elements	*Complex types*
model	model
	outputstreams
	parameters
	specification
	streamSpecification
	streamspecifications

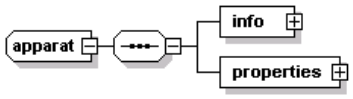
Schema: CAPEGeneral.xsd

Complex types
authors
compound
description
info
multilang
property

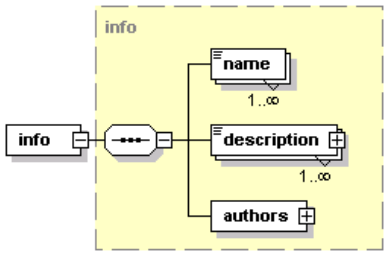
Element: apparat

diagram	
type	apparat
children	info, properties

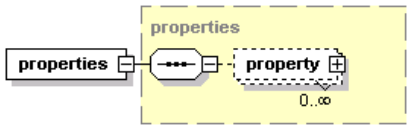
ComplexType: apparat

diagram	
children	info, properties
used by	*element* apparat

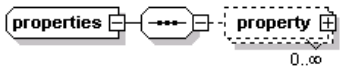
Element: apparat/info

diagram	
type	info
children	name, description, authors

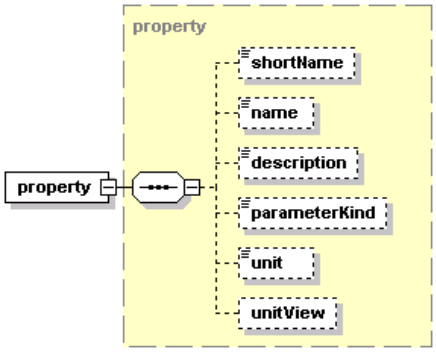
Element: apparat/properties

diagram	
type	properties
children	property

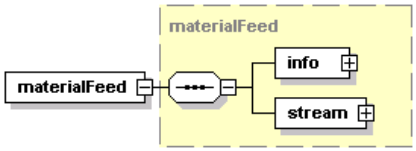
ComplexType: properties

diagram	
children	property
used by	*element* apparat/properties

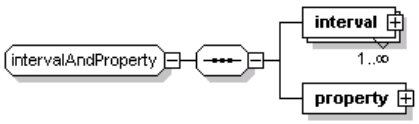
Element: properties/property

diagram	
type	property
children	shortName, name, description, parameterKind, unit, unitView
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* ref xsd:string

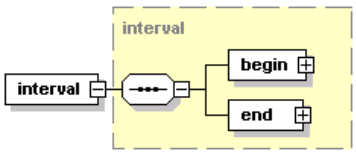
Element: materialFeed

diagram	
type	materialFeed
children	info, stream
identity constraints	*Name* *Refer* *Selector* *Field(s)* key compoundRefKey stream/compounds/compound @iupacName

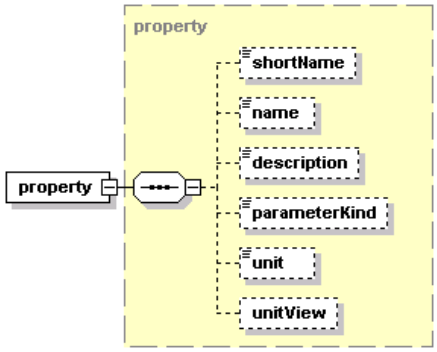
Group:: intervalAndProperty

diagram	
children	interval, property
used by	*element* dimensions/dimension

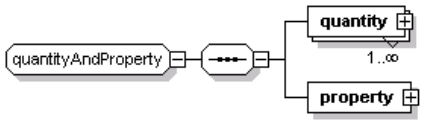
Element: intervalAndProperty/interval

diagram	
type	interval
children	begin, end
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* key xsd:string required

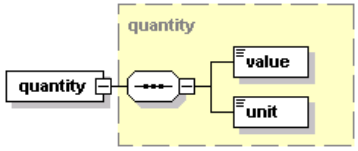
Element: intervalAndProperty/property

diagram	
type	property
children	shortName, name, description, parameterKind, unit, unitView
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* ref xsd:string

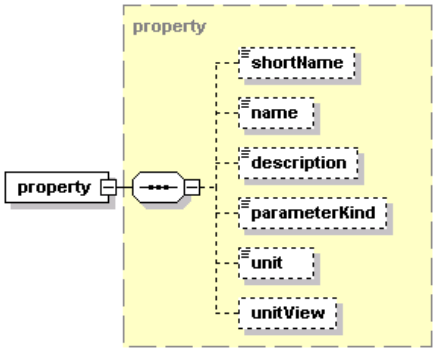
Group:: quantityAndProperty

diagram	
children	quantity, property
used by	*element* dimensions/dimension

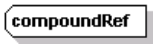
Element: quantityAndProperty/quantity

diagram	
type	extension of quantity
children	value, unit
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* key xsd:string required

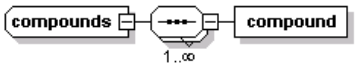
Element: quantityAndProperty/property

diagram	
type	property
children	shortName, name, description, parameterKind, unit, unitView
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* ref xsd:string

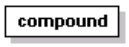
ComplexType: compoundRef

diagram	
type	restriction of xsd:anyType
used by	*element* dimensions/dimension/material
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation*

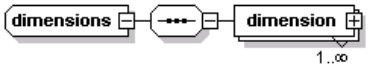
ComplexType: compounds

diagram	
children	compound
used by	*element* stream/compounds

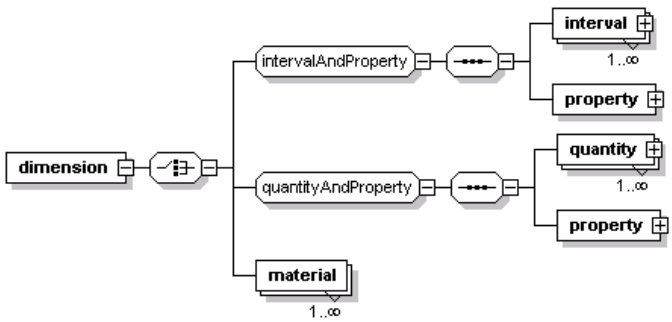
Element: compounds/compound

diagram	
type	compound
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* iupacName xsd:string chemicalFormula xsd:string casRegistryNumber xsd:string structureFormula xsd:string

ComplexType: dimensions

diagram	
children	dimension
used by	*element* distributedProperties/dimensions

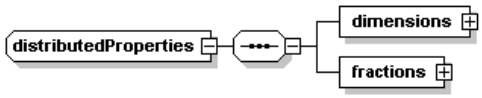
Element: dimensions/dimension

diagram	
children	interval, property, quantity, property, material
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* key xsd:string required
identity constraints	*Name* *Refer* *Selector* *Field(s)* unique intervallID ./* @key

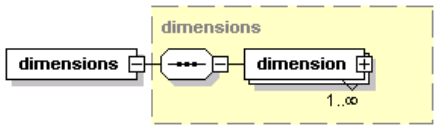
Element: dimensions/dimension/material

diagram	material
type	extension of compoundRef
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* compound xsd:string required key xsd:string required

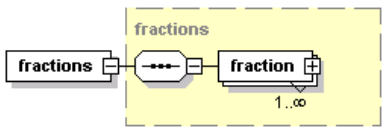
ComplexType: distributedProperties

diagram	
children	dimensions, fractions
used by	*element* phase/distributedProperties

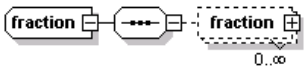
Element: distributedProperties/dimensions

diagram	
type	dimensions
children	dimension
identity constraints	*Name* *Refer* *Selector* *Field(s)* unique dimensionID dimension @key

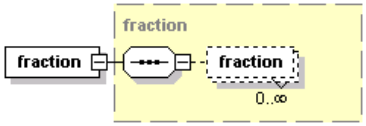
Element: distributedProperties/fractions

diagram	
type	fractions
children	fraction

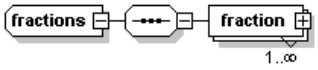
ComplexType: fraction

diagram	
children	fraction
used by	*elements* fractions/fraction, fraction/fraction
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* value xsd:double required interval xsd:string required dimension xsd:string required

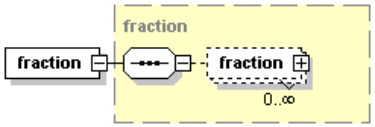
Element: fraction/fraction

diagram	
type	fraction
children	fraction
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* value xsd:double required interval xsd:string required dimension xsd:string required

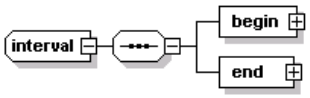
ComplexType: fractions

diagram	
children	fraction
used by	*element* distributedProperties/fractions

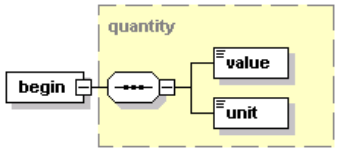
Element: fractions/fraction

diagram	
type	fraction
children	fraction
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* value xsd:double required interval xsd:string required dimension xsd:string required

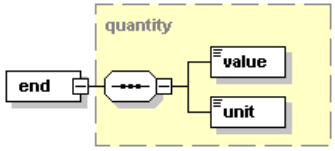
ComplexType: interval

diagram	
children	begin, end
used by	*element* intervalAndProperty/interval
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* key xsd:string required

Element: interval/begin

diagram	
type	quantity
children	value, unit

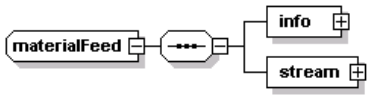
Element: interval/end

diagram	
type	quantity
children	value, unit

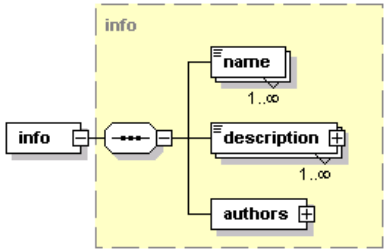
ComplexType: material

diagram						
type	extension of xsd:string					
attributes	*Name*	*Type*	*Use*	*Default*	*Fixed*	*Annotation*
	key	xsd:string	required			

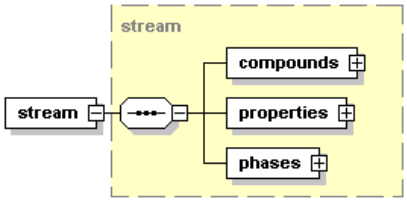
ComplexType: materialFeed

diagram	
children	info, stream
used by	*element* materialFeed

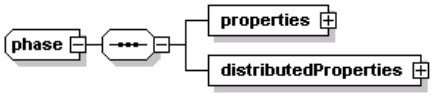
Element: materialFeed/info

diagram	
type	info
children	name, description, authors

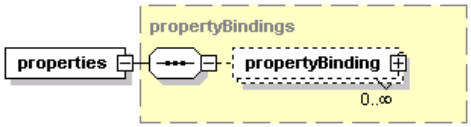
Element: materialFeed/stream

diagram	
type	stream
children	compounds, properties, phases

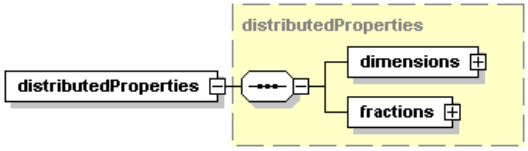
ComplexType: phase

diagram	
children	properties, distributedProperties
used by	*element* phases/phase
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* name xsd:string phaseFraction xsd:double stateOfAggregation xsd:string

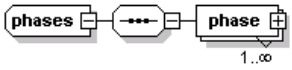
Element: phase/properties

diagram	
type	propertyBindings
children	propertyBinding

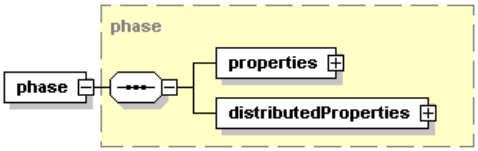
Element: phase/distributedProperties

diagram	
type	distributedProperties
children	dimensions, fractions

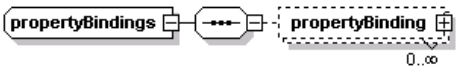
ComplexType: phases

diagram	
children	phase
used by	*element* stream/phases

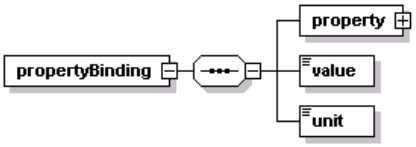
Element: phases/phase

diagram	
type	phase
children	properties, distributedProperties
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* name xsd:string phaseFraction xsd:double stateOfAggregation xsd:string

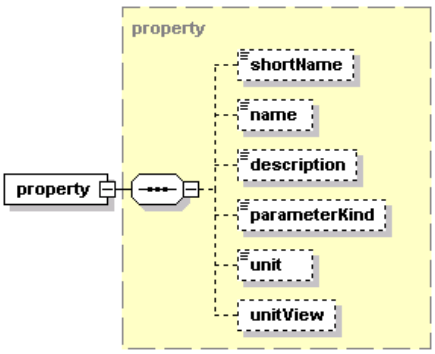
ComplexType: propertyBindings

diagram	
children	propertyBinding
used by	*elements* phase/properties, stream/properties

Element: propertyBindings/propertyBinding

diagram	
children	property, value, unit

Element: propertyBindings/propertyBinding/property

diagram	
type	property
children	shortName, name, description, parameterKind, unit, unitView
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* ref xsd:string

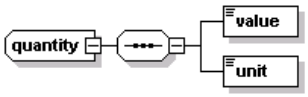
Element: propertyBindings/propertyBinding/value

diagram	
type	xsd:double

Element: propertyBindings/propertyBinding/unit

diagram	
type	xsd:string

ComplexType: quantity

diagram	
children	value, unit
used by	*elements* interval/begin, interval/end, quantityAndProperty/quantity

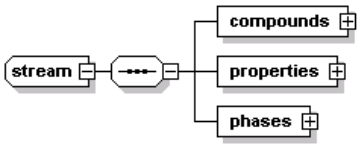
Element: quantity/value

diagram	
type	xsd:double

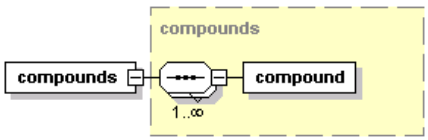
Element: quantity/unit

diagram	
type	xsd:string

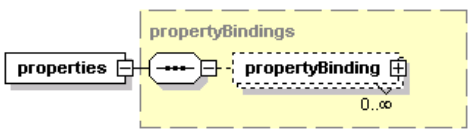
ComplexType: stream

diagram	
children	compounds, properties, phases
used by	*element* materialFeed/stream

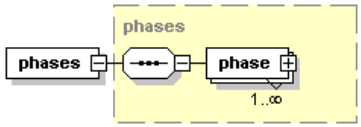
Element: stream/compounds

diagram	
type	compounds
children	compound

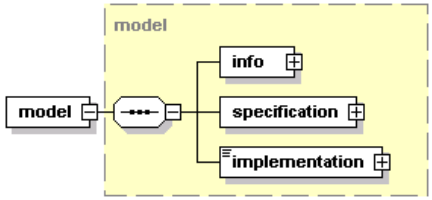
Element: stream/properties

diagram	
type	propertyBindings
children	propertyBinding

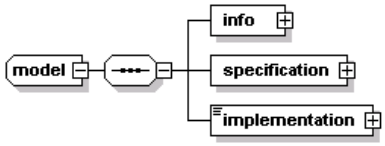
Element: stream/phases

diagram	
type	phases
children	phase

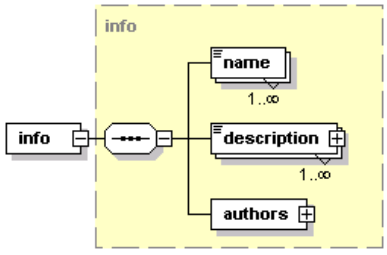
Element: model

diagram	
type	model
children	info, specification, implementation
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* type xsd:string

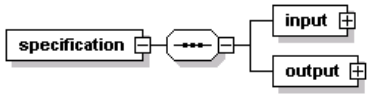
ComplexType: model

diagram	
children	info, specification, implementation
used by	*element* model
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* type xsd:string

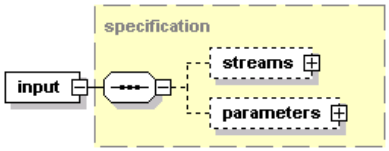
Element: model/info

diagram	
type	info
children	name, description, authors

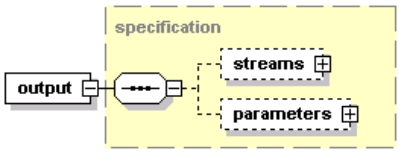
Element: model/specification

diagram	
children	input, output

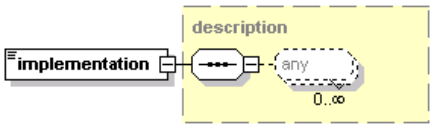
Element: model/specification/input

diagram	
type	specification
children	streams, parameters

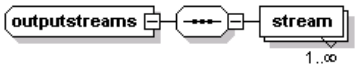
Element: model/specification/output

diagram	
type	specification
children	streams, parameters

Element: model/implementation

diagram	
type	description
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* lang xsd:language required

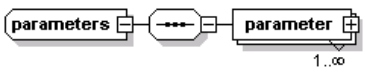
ComplexType: outputstreams

diagram	
children	stream

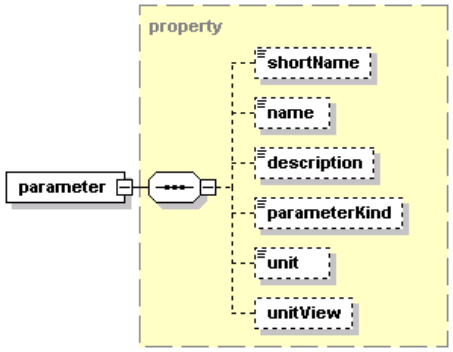
Element: outputstreams/stream

diagram	
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* role xsd:string optional general

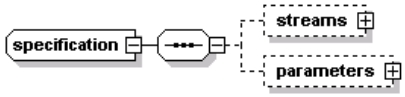
ComplexType: parameters

diagram	
children	parameter
used by	*elements* streamSpecification/fluidPhase, streamSpecification/general, specification/parameters, streamSpecification/solidPhase

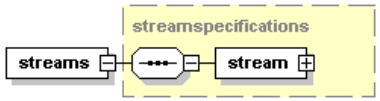
Element: parameters/parameter

diagram	
type	property
children	shortName, name, description, parameterKind, unit, unitView
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* ref xsd:string

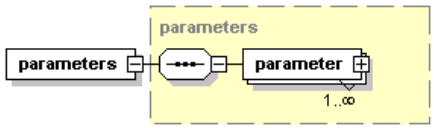
ComplexType: specification

diagram	
children	streams, parameters
used by	*elements* model/specification/input, model/specification/output

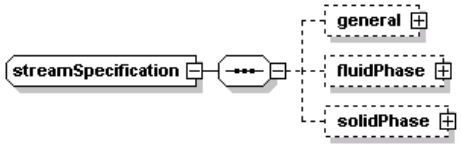
Element: specification/streams

diagram	
type	streamspecifications
children	stream

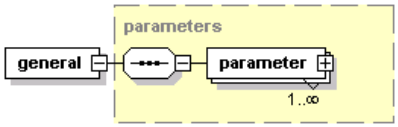
Element: specification/parameters

diagram	
type	parameters
children	parameter

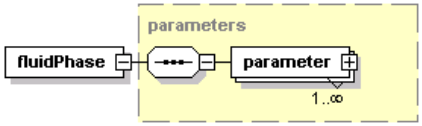
ComplexType: streamSpecification

diagram	
children	general, fluidPhase, solidPhase
used by	*element* streamspecifications/stream
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* role xsd:string general

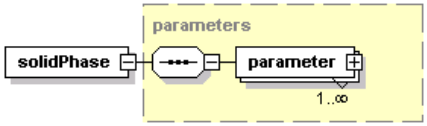
Element: streamSpecification/general

diagram	
type	parameters
children	parameter

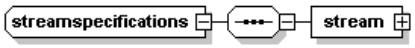
Element: streamSpecification/fluidPhase

diagram	
type	parameters
children	parameter

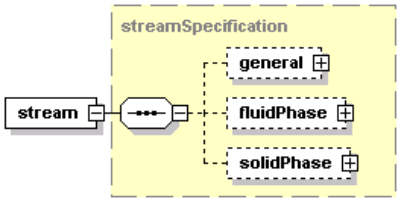
Element: streamSpecification/solidPhase

diagram	
type	parameters
children	parameter

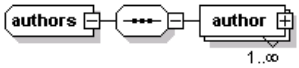
ComplexType: streamspecifications

diagram	
children	stream
used by	*element* specification/streams

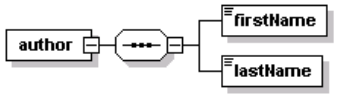
Element: streamspecifications/stream

diagram	
type	streamSpecification
children	general, fluidPhase, solidPhase
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* role xsd:string general

ComplexType: authors

diagram	
children	author
used by	*element* info/authors

Element: authors/author

diagram	
children	firstName, lastName

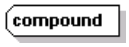
Element: authors/author/firstName

diagram	
type	xsd:string

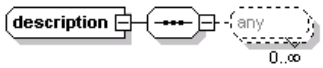
Element: authors/author/lastName

diagram	
type	xsd:string

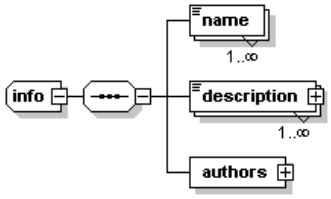
ComplexType: compound

diagram						
used by	*element* compounds/compound					
attributes	*Name*	*Type*	*Use*	*Default*	*Fixed*	*Annotation*
	iupacName	xsd:string				
	chemicalFormula	xsd:string				
	casRegistryNumber	xsd:string				
	structureFormula	xsd:string				

ComplexType: description

diagram	
type	restriction of xsd:anyType
used by	*elements* info/description, model/implementation
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation*

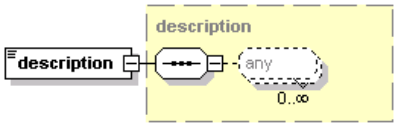
ComplexType: info

diagram	
children	name, description, authors
used by	*elements* apparat/info, materialFeed/info, model/info

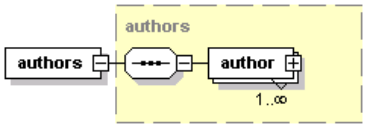
Element: info/name

diagram					
type	multilang				
attributes	*Name*	*Type*	*Use*	*Default*	*Fixed*	*Annotation*
	lang	xsd:language	required			

Element: info/description

diagram	
type	description
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* lang xsd:language required

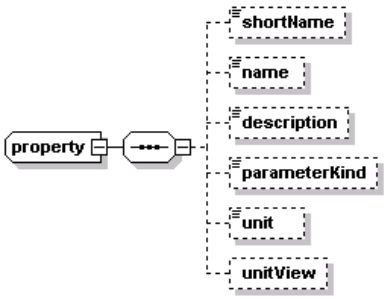
Element: info/authors

diagram	
type	authors
children	author

ComplexType: multilang

diagram	
type	extension of xsd:string
used by	*elements* property/description, info/name, property/name
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* lang xsd:language required

ComplexType: property

diagram	
children	shortName, name, description, parameterKind, unit, unitView
used by	*elements* parameters/parameter, properties/property, propertyBindings/propertyBinding/property, intervalAndProperty/property, quantityAndProperty/property
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* ref xsd:string

Element: property/shortName

diagram	
type	xsd:string

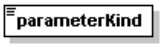
Element: property/name

diagram	
type	multilang
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* lang xsd:language required

Element: property/description

diagram	
type	multilang
attributes	*Name* *Type* *Use* *Default* *Fixed* *Annotation* lang xsd:language required

Element: property/parameterKind

diagram	
type	xsd:string

Element: property/unit

diagram	
type	xsd:string

Element: property/unitView

diagram	

C. Modellbeschreibung eines Kegelbrechers

In den folgenden Abschnitten werden mathematische Modelle eines Kegelbrechers[1] aus [Ros98] beschrieben. Zusätzlich wird auf die in [Ros98] angegebenen Orginalquellen verwiesen.

Parameter und Attribute

Apparateparameter

- Brechertyp [–]
- Spaltweite (offen) [m]
- Spaltweite (geschlossen) [m]
- Hub [m]
- mittlerer Durchmesser [m]
- mittlerer Umfang [m]
- Drehzahl $[\frac{1}{s}]$
- Exzentrizität [m]
- oberer Durchmesser [m]
- unterer Durchmesser [m]
- Kegeldurchmesser [m]

Erforderliche Feststoffattribute

- Partikelgrößenverteilung

Kostenansatz

$$\frac{K_I}{\mathrm{DM}} = 197.829,0 \cdot \left(\frac{\dot{M}_s}{\mathrm{t/h}}\right)^{0,3017}$$

Quelle:
[Baugeräte-Liste, Bau-Verlag, 1991, Umrechnung mit Kostenindex nach 1998]

[1] Entsprechend der Implementierung der `CConecrusher_GapSize`-Klasse der Software SOLIDSIM

Leistungsaufnahme

$$\frac{K_I}{\mathrm{KW}} = 3,0 \cdot 10^{-5} \cdot \left(\frac{\dot{M}}{\mathrm{t/h}}\right)^3 - 0,0113 \cdot \left(\frac{\dot{M}}{\mathrm{t/h}}\right)^2 + 1,69 \cdot \left(\frac{\dot{M}}{\mathrm{t/h}}\right) + 11,408$$

Quelle:
[Baugeräte-Liste, Bau-Verlag, 1991]

Dimensionierungsmodell für Flachkegelbrecher

Vorgabewert: Spaltweite im geschlossenen Zustand CSS in m

Hub

$$h = 7,0 \cdot CSS$$

Exzentrizität

$$e = 0,5 \cdot h$$

Spaltweite offen

$$OSS = CSS + h$$

Konusneigungswinkel

$$\delta = 45^\circ$$

Reibbeiwert

$$\mu = 0,3$$

Kalibrierungslänge

$$l_K = 0,5 \cdot h$$

Drehzahl

$$\frac{N}{\mathrm{s}^{-1}} = \sqrt{9,81 \cdot \frac{\sin(\delta) - \mu\cos(\delta)}{2 \cdot l_K}}$$

Maulweite

$$w = 2,4 \cdot x_{\mathrm{max},F}$$

mittlerer Brecherdurchmesser

$$D_m = \frac{\dot{V}_{SG,F}}{\pi \cdot CSS \cdot l_K \cdot N}$$

unterer Brecherdurchmesser

$$D_u = D_M + l_K \cdot \cos(\delta) + CSS + 2 \cdot e$$

Quelle:
[Höffl, H: Zerkleinerungs- und Klassiermaschinen, Springer-Verlag 1987]

Prozessmodell

$$p_i = \mathrm{D}(x_{i,\mathrm{oben}}) - \mathrm{D}(x_{i,\mathrm{unten}})$$

mit

$$\mathrm{D}(x) = \left(\frac{x}{CSS}\right)^k$$

mit

x	Partikelgröße
CSS	Spaltweite geschlossener Spalt
k	apparate- und materialabhängiger Streuungsparameter

Abkürzungsverzeichnis

m	Masse
p	Druck
R	Gaskonstante
T	Temperatur
V	Volumen
ADL	asset definition language
AL	asset language
AML	asset manipulation language
AOA	Assetorientierte Analyse
API	application programming interface
AQL	asset querying language
AXIS	apache extensible interaction system
BLOB	binary large object
BOEP	Business-Oriented Software Engineering Process
bzw.	beziehungsweise
CAPE	computer aided process engineering
CMS	content management system
d. h.	das heißt
EJB	Enterprise Java Bean
FTP	file transfer protocol
ggf.	gegebenenfalls
HTML	hypertext markup language
HTTP	hypertext transfer protocol
i. Allg.	im Allgemeinen
JAX-RPC	Java API for XML-based RPC
JDBC	java database connectivity
OLAP	online analytical processing
OOA	Objektorientierte Analyse
RPC	remote procedure call
RUP	Rational Unified Process
SA	Strukturierte Analyse
SOAP	simple object access protocol
UML	unified modelling language
usw.	und so weiter
vgl.	vergleiche
WAP	wireless application protocol
WML	wireless markup language
WSDL	webserservice decsription language
z. B.	zum Beispiel

Literaturverzeichnis

[AC96] ABADI, Martin ; CARDELLI, Luca: *A Theory of Objects*. Springer-Verlag New York, Inc., 1996

[Apa04a] APACHE SOFTWARE FOUNDATION. *The Jakarta Site – Apache Tomcat*. URL `http://jakarta.apache.org/tomcat/`. Februar 2004

[Apa04b] APACHE SOFTWARE FOUNDATION. *WebServices - Axis*. URL: `http://ws.apache.org/axis/`. Februar 2004

[Bal98] BALZERT, Helmut: *Lehrbuch der Softwaretechnik*. Bd. 2 Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg, Berlin : Spektrum, Akademischer Verlag, 1998

[Bal99] BALZERT, Heide: *Lehrbuch der Objektmodellierung - Analyse und Entwurf*. Heidelberg, Berlin : Spektrum Akademischer Verlag, 1999

[Bal01] BALZERT, Helmut: *Lehrbuch der Softwaretechnik*. Bd. 1 Software-Entwicklung. 2. Auflage. Heidelberg, Berlin : Spektrum Akademischer Verlag, 2001

[Büc02] BÜCHNER, Thomas: *Entwurf und Realisierung eines Java-Frameworks zur inhaltlichen Erschließung von Dokumenten*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diplomarbeit, 2002

[Cas02] CASSIRER, Ernst: *Philosophie der symbolischen Formen*. Bd. 11-13: *Gesammelte Werke*. Hamburger Ausgabe edition. Felix Meiner Verlag GmbH, 2001-2002

[CO-04] CO-LAN. *The CAPE-OPEN Laboratories Network*. URL: `http://colan.adduce.de/portal/index.html`. März 2004

[Det03] DETTMANN, Kai: *Eine generische Online-Beschaffungsplattform: Anforderungsanalyse, objektorientierter Entwurf, Realisierung*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diplomarbeit, 2003

[Deu04] DEUTSCHE GESELLSCHAFT FÜR SEMIOTIK DGS E.V. . *Homepage*. URL: `http://www.semiotik.org/`. Januar 2004

[Die01] DIESTELHORST, Lars: *Recommendation Engines*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Studienarbeit, 2001

[DIN 28004] Norm DIN 28004 1977. Fließbilder verfahrenstechnischer Anlagen, Fließbilderarten, Informationsgehalt

[DUD01] DUDEN (Hrsg.): *Das Fremdwörterbuch*. Dudenverlag, 2001

[EN ISO 10628] Norm EN ISO 10628 2000. Fließschemata für verfahrenstechnische Anlagen - Allgemeine Regeln (ISO 10628:1997)

[Eve95] EVERLING, W.: Leserbrief. In: *Informatik-Spektrum* 18 (1995)

[FS00] FOWLER, Martin ; SCOTT, Kendall: *UML Distilled: a brief guide to the standard object modelling language*. second. Addison-Wesley, 2000

[Gal03] GALINSKI, Jan: *Eine Architektur für den Zugriff heterogener Clients auf ein Content-Management- und Workflow-Management-System unter Verwendung des offenen Protokolls SOAP*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Studienarbeit, 2003

[Ge03] GE, Jing: *Einbindung des WebDAV-Dienstes in ein Informationsportal*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diplomarbeit, 2003

[GHJV94] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994

[GRWT95] GRUHN, G. ; ROSENKRANZ, J ; WERTHER, J ; TÖBERMANN, J.-Chr.: *Entwicklung eines Modell- und Fließschemasimulationssystems für komplexe Feststoffprozesse*. 1995. – Vortrag zur internen Arbeitssitzung des GVC-Fachausschusses *Prozeß- und Anlagentechnik*, Merseburg

[HS03] HOMANN, Dennis ; SPILKER, Daniel: *Erweiterung eines Portalsystems um einen mengenorientierten Zugriff auf semi-strukturierte Informationsbestände*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Studienarbeit, 2003

[Hup04] HUPE, Patrick. *WIPS IT 1: Internet Services for Ship Technology (Schiffstechnische Dienste im Internet)*. URL: `http://www.sts.tu-harburg.de/projects/WIPS/entry.html`. April 2004

[inf01] INFOASSET AG: *Der infoAsset Broker*, 2001. – Whitepaper

[inf04] INFOASSET AG. *Knowledge Management Software infoAsset Broker*. URL: `http://www.infoAsset.de`. Februar 2004

[Jac02] JACOBSEN, Cornelia: *Personalisierung: Konzepte und Techniken*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diplomarbeit, 2002

[JSR53a] Norm JSR-053 (Teil I) August 2001. Java Servlet Specification - Version 2.3

[JSR53b] Norm JSR-053 (Teil II) August 2001. Java Server Pages Specification - Version 1.3

[Koc02] KOCH, Leif M.: *Wissensportalsoftware als Learning Management System: Fachliche Analyse, Entwurf, prototypische Realisierung*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diplomarbeit, 2002

[Kru96] KRUCHTEN, Philippe: A Rational Development Process. In: *Crosstalk* 9 (1996), July, Nr. 7, S. 11–16

[Kru98] KRUCHTEN, Philippe: *The Rational Unified Process*. Addison-Wesley, 1998

[Leh01] LEHEL, Vanda: *Entwurf und Realisierung eines Basisdienstes zur Unterstützung arbeitsteiliger Geschäftsvorgänge in einem Unternehmensportal*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diplomarbeit, 2001

[Mar03] MARQUARDT, Wolfgang: *Vorlesungsmanuskript: Prozessentwicklung in der Verfahrenstechnik*. URL: `http://www.lfpt.rwth-aachen.de/Studium/Vorlesungen/Prozessentwicklung/umdruck_03.pdf`. Oktober 2003. – Scriptum zur Vorlesung

[NMMZ00] NOACK, Jörg ; MEHMANESH, Hamarz ; MEHMANECHE, Homayoun ; ZENDLER, Andreas: Architekturen für Network Computing. In: *WIRTSCHAFTSINFORMATIK* 42 (2000), Nr. 1, S. 5–14

[Obj03] OBJECT MANAGEMENT GROUP. *UML Resource Page*. URL: `http://www.omg.org/uml/`. Dezember 2003

[Obj04] OBJECT MANAGEMENT GROUP. *Welcome To The OMG's CORBA Website*. URL: `http://www.corba.org/`. März 2004

[Oes98] OESTEREICH, Bernd: *Objektorientierte Softwareentwicklung : Analyse und Design mit der Unified modeling language*. München, Wien : R. Oldenbourg Verlag, 1998

[Pei31] PEIRCE, C. S.: *Collected Papers of Charles Sanders Peirce.* Cambridge : Harvard University Press, 1931

[Rie03] RIEBEL, Ulrich. *Fundamentals of Process Engineering: Particle Technology*. Internet, Scriptum zur Vorlesung. 2003

[Ros98] ROSENKRANZ, Jan: *Rechnergestützte Fließschemasimulation und Optimierung komplexer Feststoffprozesse*, Technische Universität Hamburg-Harburg, Germany, Diss., 1998

[Sch95] SCHULER, H. (Hrsg.): *Prozeßsimulation*. VCH-Verlagsgesellschaft, 1995

[Sch99] SCHEFE, P.: Softwaretechnik und Erkenntnistheorie. In: *Informatik Spektrum* 22 (1999), S. 122–135

[Seh03] SEHRING, H.-W.: *Konzeptorientiertes Content Management: Model, Systemarchitektur und Prototypen*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diss., 2003

[SS03] SCHMIDT, J. W. ; SEHRING, H.-W.: Conceptual Content Modeling and Management: The Rational of an Asset Language. In: *Perspectives Of System Informatics (PSI'03)*, Springer-Verlag, 2003

[Sta02] STAAB, Steffen: Wissensmanagement mit Ontologien und Metadaten. In: *Informatik-Spektrum* 25 (2002), June, Nr. 3, S. 194–209

[Sun04] SUN MICROSYSTEMS, INC. *Java API for XML-based RPC*. URL: `http://java.sun.com/xml/jaxrpc/index.jsp`. März 2004

[Tha93] THALHEIM, B.: Foundations of Entity-Relationship Modeling. In: *Annals of Mathematics and Artificial Intelligence* 7 (1993), Nr. 1-4, S. 197–256

[Weg02] WEGNER, Holm: *Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement*, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, Germany, Diss., 2002

[Wik04] WIKIPEDIA. *Semiotisches Dreieck*. URL: `http://de.wikipedia.org/wiki/Semiotisches_Dreieck`. Januar 2004

[Wit63] WITTGENSTEIN, L.: *Tractatus logico-philosophicus*. Frankfurt, 1963

[Wit67] WITTGENSTEIN, L.: *Philosophische Untersuchungen*. Frankfurt, 1967

[Wor03a] WORLD WIDE WEB CONSORTIUM: *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*. URL: `http://www.w3.org/TR/MathML/`. Oktober 2003. – W3C Recommendation

[Wor03b] WORLD WIDE WEB CONSORTIUM: *SOAP Version 1.2 Part 0: Primer*. URL: `http://www.w3.org/TR/2003/REC-soap12-part0/`. Juni 2003. – W3C Recommendation

[Wor03c] WORLD WIDE WEB CONSORTIUM: *SOAP Version 1.2 Part 1: Messaging Framework*. URL: `http://www.w3.org/TR/2003/REC-soap12-part1/`. Juni 2003. – W3C Recommendation

[Wor03d] WORLD WIDE WEB CONSORTIUM: *SOAP Version 1.2 Part 2: Adjuncts*. URL: `http://www.w3.org/TR/2003/REC-soap12-part2/`. Juni 2003. – W3C Recommendation