



Universität Karlsruhe (TH)
Fakultät für Informatik
*Institut für Programmstrukturen und
Datenorganisation (IPD)*



Anpassung eines Metamodells zur Beschreibung imperfekter Informationen in einem Data-Warehouse-System

Studienarbeit

von

Nils Hilt

30. April 2005

Verantwortlicher Betreuer: Prof. Dr.-Ing. Klemens Böhm
Betreuender Mitarbeiter: Dipl.-Inform. Heiko Schepperle

Ich erkläre hiermit, die vorliegende Arbeit selbstständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Die verwendeten Literaturquellen sind im Literaturverzeichnis angegeben.

Karlsruhe, den 30. April 2005

Zusammenfassung

Im Projekt OVID beschäftigt man sich mit imperfekten und aggregierten Verkehrsdaten. Dabei wird auf Techniken aus dem Data Warehousing zurückgegriffen. Damit die an einem Data-Warehouse-Prozess beteiligten Werkzeuge eine Chance haben, sinnvoll mit imperfekten Informationen umzugehen, muss die Imperfektion mit Hilfe eines Metamodells beschrieben werden. Im Rahmen dieser Studienarbeit wird das Common Warehouse Metamodel (CWM) erweitert, damit die Imperfektion mit Mitteln des CWM beschrieben werden kann. Dabei wird das Profil-Konzept aus UML 2.0 bereits auf das CWM angewendet. Das angepasste Modell wird anschließend verwendet, um imperfekte Informationen in einem bereits existierenden Visualisierungswerkzeug zu importieren.

Inhaltsverzeichnis

1	Einleitung	1
1.1	OVID-Projekt	1
1.2	Szenario	1
1.3	Ziel und Vorgehensweise	2
2	Erweiterungen des Common Warehouse Metamodel (CWM)	3
2.1	Das Common Warehouse Metamodel (CWM)	3
2.2	Standardisierte Erweiterungstechniken des CWM	4
2.2.1	Stereotypes und TaggedValues	4
2.2.2	Objektorientierte Erweiterung	5
2.3	Der Profiling-Mechanismus aus der UML Version 2	7
2.3.1	Erweiterungsmöglichkeiten der UML2	7
2.3.2	OMG Modellierungs- und Metadatenstandards	9
2.3.3	Entwicklung der Profile von UML Version 1.3 nach 2.0	10
2.3.4	Erweiterung des CWM mit UML2 Profilen	12
2.4	Auswahl einer Erweiterungstechnik	13
3	Imperfektion in Datenbanken	15
3.1	Theorie über Imperfektion	15
3.1.1	Wahrscheinlichkeitstheorie	16
3.1.2	Fuzzy-Theorie	16
3.2	Das Kontextmodell	17
3.2.1	Scharfe Kontexte	18
3.2.2	Unscharfe Kontexte	18
3.3	Erste Erweiterungen des CWM für die Modellierung imperfekter Daten	18
3.3.1	TaggedValues und Stereotypes	19
3.3.2	Objektorientierte Erweiterung	19
3.3.3	Wertung der vorgestellten Erweiterungen	20

4	Weitergehende CWM-Erweiterung zur Beschreibung von Imperfektion	21
4.1	Vorüberlegungen	21
4.2	Erweiterung des CWM zur Beschreibung unsicherer Informationen	23
4.2.1	Ansatzpunkte einer Erweiterung	23
4.2.2	Modellierung	24
4.3	Erweiterung des CWM zur Beschreibung unscharfer Informationen	26
4.3.1	Ansatzpunkte einer Erweiterung	26
4.3.2	Erste Modellierungsmöglichkeit	28
4.3.3	Zweite Modellierungsmöglichkeit	29
4.4	Was ist mit ungenauen Informationen	32
4.5	Zusammenfassung	33
5	Evaluierung	35
5.1	Schwierigkeit	35
5.2	Lösungsansätze	35
5.2.1	Simulation mit standardisierten CWM 1.0 Stereotypen	36
5.2.2	Simulation durch neue Unterklassen von bestehenden CWM 1.0 Meta- klassen	37
5.2.3	Auswahl einer Lösungsmöglichkeit	37
6	Konzeption und Implementierung eines Importmechanismus in ein Analyse- Werkzeug	39
6.1	Das Visualisierungswerkzeug	39
6.2	Konzeption eines CWM-Importmechanismus	39
6.3	Implementierung	40
7	Zusammenfassung und Ausblick	43
A	Erweiterung der CWM-DTD	45

Abbildungsverzeichnis

1	CWM-Pakete	3
2	CWM-Modell der relationalen Tabelle <i>Stau</i>	4
3	CWM::Core-Paket	5
4	Erweiterung einer CWM -lasse durch Vererbung	6
5	UML-Profil einer Enterprise Java Bean	8
6	Anwendung des UML-Profiles für EJB	8
7	OMG Metadaten Architektur	9
8	Zusammenspiel der neuen OMG Modellierungsstandards	11
9	UML Version 1.3 Erweiterungsmechanismus	11
10	OO-Erweiterung des Relational-Pakets um Imperfektion	19
11	TrapezoidMembershipFunction a) unscharf (fuzzy) — b) scharf	20
12	Zusammenhang der Pakete auf der Resource-Schicht	22
13	Spezialisierungen von Core::Attribute und Core::Class	22
14	Ausschnitt CWM::Core-Paket	23
15	Auszug aus dem Profil Imperfection: Stereotyp <<uncertain>>	25
16	Beispiel: unsichere Stautabelle	26
17	Idee der Modellierung eines unscharfen Datentyps	27
18	Linguistische Variable	28
19	Auszug aus dem Profil Imperfection: Stereotyp <<fuzzy>>	29
20	Beispiel: Stautabelle mit unscharfem Attribut	30
21	Zugehörigkeitsfunktionen der linguistischen Variable „Staulänge“	30
22	Auszug aus dem Profil Imperfection: Stereotyp <<fuzzy2>>	31
23	Beispiel: Stautabelle mit unscharfer Staulänge ohne Zugehörigkeitsfunktionen . .	32
24	Stereotyp <<uncertain>>	36
25	Ausschnitt aus dem Klassendiagramm des erweiterten Visualisierungswerkzeuges	40
26	Simulation der Stereotypen für imperfekte Datentypen	45
27	Simualtion der Stereotypen <<probability>> und <<term>>	46

1 Einleitung

1.1 OVID-Projekt

Kilometerlange Staus sind inzwischen ein tägliches Bild auf unseren Autobahnen. Aktuelle Pressemitteilungen [1] zitieren Christoph Huß, den Leiter der Wissenschaft- und Verkehrspolitik der BMW Group, wonach der zähfließende Verkehr einen volkswirtschaftlichen Schaden von jährlich etwa 100 Milliarden Euro verursacht. Zwar vertreten einzelne Studien die Thesen, dass in Zukunft die Verkehrsleistung im Personenverkehr in Deutschland stagnieren wird [4], aber dafür muss man auf jeden Fall davon ausgehen, dass im Güterverkehr die Verkehrsleistung noch enorm steigen wird [3].

Der Einsatz von Telematik und Informationssystemen im Verkehrswesen wird als eine der großen Möglichkeiten angesehen, die immense Belastung auf unseren Straßen in den Griff zu bekommen. Gesucht sind Lösungen, die die zu erwartende Verkehrsnachfrage optimal auf die vorhandene Verkehrsinfrastruktur verteilen. Die Idee ist, dass man unter anderem das Stauproblem ohne größere Investitionen in das Straßennetz, das heißt ohne Neubau oder Ausbau von Straßen, lösen kann. Im vom Bundesministerium für Bildung und Forschung geförderten Verbundprojekt OVID (Stärkung der SelbstOrganisationsfähigkeit im Verkehr durch I+K gestützte Dienste) der Universität Karlsruhe (TH) [19] beschäftigen sich mehrere Institute und Partnerfirmen aus der Wirtschaft mit dem Aufbau einer Plattform zur Modellierung und Bewertung von verkehrsinfrastrukturellen, verkehrstelematischen und logistischen Maßnahmen im Verkehrs- und sozio-ökonomischen System.

Das Institut für Programm- und Datenstrukturen (IPD) ist am OVID-Teilprojekt **B1: „Verlässliche Datenbanken für die Informationsbereitstellung im Verkehr“** beteiligt. Dort löst es unter anderem das Problem, wie imperfekte und aggregierte Verkehrsdaten mit Hilfe von Data-Warehouse-Techniken verarbeitet werden können.

Diese Studienarbeit wird im Rahmen des OVID-Projektes angefertigt.

1.2 Szenario

Stellen wir uns ein fiktives Data-Warehouse-System vor. Ausgehen wollen wir von einer Staudatenbank, die Informationen über die aktuellen Staus in Baden-Württemberg enthält. Diese Informationen bestehen aus der Autobahn- oder Bundesstraßen-*Nummer* auf der ein Stau vorliegt, der *Staulängen* und der Angabe der *Orte des Staubeginns und -endes*.

Erfahrungen zeigen, dass eine Staumeldung im Radio noch lange nicht bedeutet, dass auf der Straße auch ein Stau vorherrscht. Je nachdem, ob die Staumeldung von der Polizei oder einem der zahlreichen freiwilligen Staumelder kommt, muss ihr mehr oder weniger Glauben geschenkt werden. Die Ortsangaben von Staumeldern können ungenau sein, die Staulänge kann oft auch nur geschätzt werden, und es können widersprüchliche Informationen über den gleichen Stau vorliegen. Datenbanken speichern aber prinzipiell nur präzise und genaue Werte. Die Diplomarbeit von Erik Koop [9] beschreibt, wie ungenaue, unsichere und unscharfe Daten — also imperfekte Daten — aus dem Verkehrsbereich in relationalen Datenbanken gespeichert werden können.

Diese imperfekten Informationen über Staus auf baden-württembergischen Straßen sollen mit anderen Systemen ausgetauscht werden. Zum Beispiel sollen die Daten auf ein Navigationssystem übertragen oder einem Data-Warehouse-System zur Analyse der langfristigen Entwicklungen auf deutschen Straßen zur Verfügung gestellt werden. Hier kommt nun das Common Warehouse Metamodel (CWM) mit ins Spiel. Dieser Standard der Object Management Group (OMG) [18] stellt eine Möglichkeit dar, den gesamten Data-Warehouse-Prozess zu modellieren und diese Metadaten auszutauschen [20]. Der Datenaustausch zwischen der Staudatenbank und dem Analysetool kann somit, dank des gemeinsamen Verständnisses der Metadaten aufgrund des Einsatzes des CWM problemlos stattfinden.

Leider unterstützt das Common Warehouse Metamodel nicht die Modellierung imperfekter Daten. Dies bedeutet, dass beim Austausch der Verkehrsdaten die Kenntnis über die Imperfektion der Stau-meldungen verloren geht.

1.3 Ziel und Vorgehensweise

Das Ziel dieser Studienarbeit ist es, ausgehend von einer Vorgängerarbeit [8] das Common Warehouse Metamodel zu erweitern, so dass man damit imperfekte Informationen beschreiben kann. Um die modellierte Erweiterung an einem konkreten Beispiel vorstellen zu können, wird im Rahmen dieser Studienarbeit auch ein Importmechanismus, der auf dem erweiterten Metamodell basiert, für ein Werkzeug zur Visualisierung von imperfekten Informationen konzipiert und implementiert.

In den nächsten Kapiteln wird zunächst untersucht, welche Möglichkeiten es zur Erweiterung des CWM gibt. Anschließend wird vorgestellt, wie imperfekte Informationen in Datenbanken repräsentiert werden können. Der Schwerpunkt dieser Arbeit liegt dann in Kapitel 4, in dem eine Erweiterung des CWM zur Beschreibung imperfekter Informationen vorgestellt wird. Abschließend wird dann noch kurz das Visualisierungswerkzeug und der dafür implementierte Importmechanismus beschrieben.

2 Erweiterungen des Common Warehouse Metamodel (CWM)

2.1 Das Common Warehouse Metamodel (CWM)

Das Common Warehouse Metamodel ist ein Standard der Object Management Group (OMG) [18] für den Metadaten austausch in Data-Warehousing Umgebungen. Es bietet eine gemeinsame Sprache, basierend auf UML 1.3, um Metadaten zu beschreiben und die Möglichkeit diese XML basiert auszutauschen [20, 12].

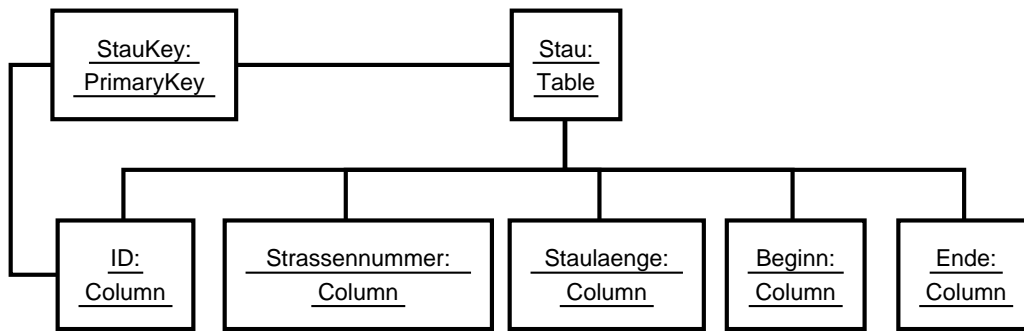
Management	Warehouse Process			Warehouse Operation		
Analysis	Transformation	OLAP	Data Mining	Information Visualization	Business Nomenclature	
Resource	Object	Relational	Record	Multidimensional	XML	
Foundation	Business Information	Data Types	Expressions	Keys and Indexes	Software Deployment	Type Mapping
Object Model	Core		Behavioral	Relationships		Instance

Abbildung 1: CWM-Pakete

Das CWM besteht aus 21 Paketen (das *Object*-Paket in der Resource-Schicht existiert nicht als selbständiges Paket), die in fünf Schichten organisiert sind (siehe Abbildung 1) [20].

Wie es für Schichtenarchitekturen üblich ist, basieren Pakete höherer Schichten nur auf Paketen niedriger Schichten. Die Aufteilung in mehrere Pakete dient nicht nur der klareren Strukturierung, sondern ermöglicht auch, dass man sich nur auf die zur Modellierung benötigten Pakete und die zusätzlichen abhängigen Pakete aus niedrigeren Schichten beschränken kann. Beispielsweise reicht es zur Modellierung eines relationalen Datenbasisschemas aus, das *Relational*-Paket und fünf weitere Pakete (*Core*, *Behavioral*, *Instance*, *DataTypes* und *Keys and Indexes*), auf denen das *Relational*-Paket basiert, zu kennen [20, Seite 108].

Der folgende Absatz verdeutlicht an einem Beispiel, wie man das *Relational*-Paket zur Modellierung eines einfachen Datenbasisschemas verwendet. Beschrieben wird eine relationale Tabelle *Stau*. Diese Stautabelle habe fünf Spalten: *ID*, *Autobahn- oder Bundesstraßennummer*, *Staulänge*, *Beginn* und *Ende*. Die Spalte *ID* stelle auch den Primärschlüssel der Tabelle dar. Abbildung 2 zeigt nun wie diese Tabelle mit dem CWM modelliert wird. Eine Instanz der CWM-Metaklasse *Relational::Table* steht für die Stautabelle. Nach dem Metamodell kann sie beliebig viele Assoziationen mit *Columns* eingehen. Im Beispiel also gibt es fünf Instanzen der Metaklasse *Relational::Column*. Durch eine Instanz der Metaklasse *PrimaryKey* wird der Primärschlüssel modelliert. Objekte der Klasse *PrimaryKey* müssen

Abbildung 2: CWM-Modell der relationalen Tabelle *Stau*

mindestens zwei Assoziationen eingehen, zum einen zur Tabelleninstanz und des weiteren zu den Spalten, die die Schlüsselbedingung bilden.

Der CWM-Standard ermöglicht die Modellierung vieler Komponenten, die am Data-Warehouse-Prozess beteiligt sind. Trotzdem werden natürlich nicht alle möglichen Anwendungsfelder unterstützt. Als Beispiel wäre hier das Entity-Relationship-Modell zu erwähnen, welches vom CWM-Standard nicht abgedeckt wird. Um nun auch spezielle Anwendungsfelder zu unterstützen, beschreibt die Spezifikation des CWM-Standards zwei Möglichkeiten das Common Warehouse Metamodel an eigene Bedürfnisse anzupassen. Neben einer Erweiterung um das Entity-Relationships-Modell werden ergänzend zum CWM-Standard von der OMG in [13] noch weitere Anpassungen des CWM an spezielle Anwendungsgebiete vorgestellt, die diese Erweiterungstechniken nutzen. Im folgenden Abschnitt 2.2 werden diese Techniken vorgestellt und erläutert, wie sie verwendet werden können, um das CWM an eigene Bedürfnisse anzupassen.

Mit dem neuen UML 2.0 Standard gewinnt auch der Erweiterungsmechanismus für UML — das sogenannte *UML-Profiling* — weiter an Mächtigkeit. Es bietet sehr vielfältige Möglichkeiten zur Modellierung von Erweiterungen. Die betrachtete CWM Version 1.0 basiert zwar noch auf UML 1.3, aber im Rahmen einer Vereinheitlichung der verschiedenen OMG-Standards ist zu erwarten, dass auch zukünftige CWM Versionen mit dieser Technik angepasst werden können. In dieser Studienarbeit werde ich deshalb auch untersuchen (siehe Abschnitt 2.3), ob und wie weit Möglichkeiten bestehen das UML2-Profiling für eine Erweiterung der noch aktuellen CWM Version zu nutzen.

2.2 Standardisierte Erweiterungstechniken des CWM

Die Spezifikation des Common Warehouse Metamodel [12] beschreibt zwei verschiedene Möglichkeiten, das CWM an eigene Bedürfnisse anzupassen, zum einen den Einsatz von *Stereotypes* und *TaggedValues* und des weiteren die Verwendung objektorientierter Erweiterungstechniken.

2.2.1 Stereotypes und TaggedValues

Eine sehr einfache Erweiterungstechnik ist der Einsatz von *Stereotypes* und *TaggedValues*. Ein Blick auf das *Core*-Paket des Metamodells (Abbildung 3) zeigt, dass an ein beliebiges *ModelElement* eine beliebige Anzahl von *Stereotypes* und *TaggedValues* angehängt werden kann. Auch ein *Stereotype* kann mehrere *TaggedValues* besitzen. Ein *ModelElement*, welches mit so einem *Stereotype* ausgezeichnet ist, wird impliziert auch um die *TaggedValues* des *Stereotypes* erweitert [20, Seite 136f].

Ein *Stereotype* ist ein Erweiterungsmechanismus, der existierende CWM Klassen mit einem Namen markiert. Der Name hilft die Rolle, die eine Klasse einnimmt, genauer zu spezifizieren. Ein *TaggedValue*

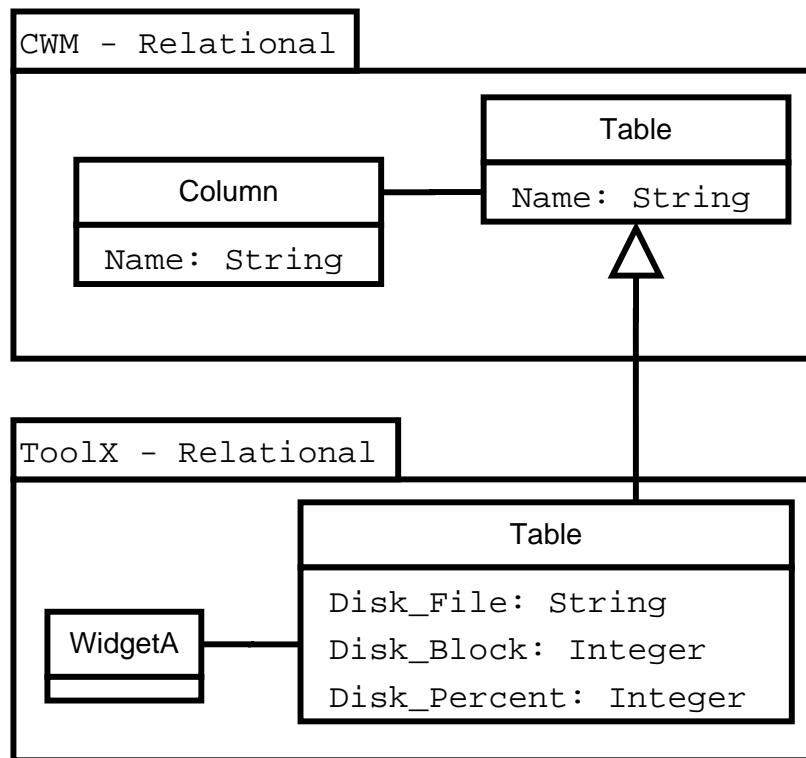


Abbildung 4: Erweiterung einer CWM -lasse durch Vererbung

Beispielsweise kann er eine eigene Metaklasse modellieren, die von einer Standardmetaklasse des CWM erbt und dann um spezielle Eigenschaften ergänzt wird. Abbildung 4 zeigt, wie die Standard CWM-Metaklasse *Table* erweitert wird, um anwendungsspezifische Informationen mit aufzunehmen, in diesem Fall den Speicherort der Tabelle. Dazu erbt die neue Metaklasse *Table* aus dem Anwendungspaket *ToolX - Relational* von der Standardmetaklasse aus dem *Relational*-Paket. Die *ToolX::Table* hat somit alle Eigenschaften, die auch einer normalen Tabelle zu eigen sind, wie zum Beispiel den Tabellennamen. Zusätzlich hat sie noch neue Attribute um den Speicherort angeben zu können. Des weiteren kann sie in diesem Beispiel noch weitere Assoziationen eingehen. Eine *ToolX::Table* besteht nicht nur aus *Columns*, sondern ist auch mit einem *WidgetA* assoziiert [20].

Leider schränkt der Einsatz von eigenen Metaklassen, die Austauschfähigkeiten der Metadaten ein. Ein CWM-konformes Tool wird keine anwendungsspezifischen Erweiterungen des Metamodells verstehen können.

Für viele Anwendungsszenarien ist es auch nicht nötig, dass andere Tools die Erweiterungen verstehen. In unserem Beispiel ist es für fremde Tools absolut uninteressant, ja sogar überhaupt nicht gewünscht, dass fremde Tools wissen, wo auf der Platte *ToolX* die Tabellen speichert. Es reicht somit aus, wenn vor dem Metadaten austausch die *ToolX::Table* zu einer *Relational::Table* upgecastet wird. Aber es gibt auch Fälle, in denen die Erweiterungen beim Austausch mit anderen Tools nicht verloren gehen sollen. Ein Beispiel ist genau die von uns gesuchte Erweiterung des CWM für den Umgang mit imperfekten Daten. Hier schlägt die OMG als Lösungsmöglichkeit den Einsatz von *TaggedValues* und *Stereotypes* vor [20, Seite 231]. Dies hat den Vorteil, dass der Metadaten austausch immer noch CWM-konform geschehen könnte. Ein direkter Austausch von Metadaten, die mit einer objektorientierten Erweiterung des CWM modelliert sind, wird nämlich scheitern, da zum Beispiel diese Metadaten Instanzen neuer Klassen enthalten können, die nicht aus dem CWM-Standard sind. Ein fremdes Tool muss über zusätzliches Wissen verfügen, damit der Austausch von Metadaten, die mit einer erweiterten CWM modelliert sind, klappen kann und damit die spezielle Semantik der Erweiterung verstanden wird.

2.3 Der Profiling-Mechanismus aus der UML Version 2

Die Unified Modelling Language Version 2 wird in zwei Spezifikationen beschrieben, der UML2 Infrastructure [17] und der UML2 Superstructure [15]. Die Infrastructure besteht aus zwei großen Paketen, dem *Core*-Paket und dem *Profile*-Paket. Das *Core*-Paket ist wiederum in vier Unterpakete aufgeteilt und beschreibt die wesentlichen Elemente, die nötig sind um die gesamte Unified Modelling Language zu definieren.

Der folgende Abschnitt beschreibt die Möglichkeiten, die das *Profile*-Paket zur Erweiterung der UML2 zur Verfügung stellt. Die weiteren Abschnitte beschreiben die Zusammenhänge zwischen den verschiedenen Metamodelstandards der OMG. Daraus leite ich dann abschließend eine Vermutung ab, wie die zukünftige Version des Common Warehouse Metamodel erweitert werden kann.

2.3.1 Erweiterungsmöglichkeiten der UML2

Wie auch beim Common Warehouse Metamodel stellt sich auch bei der Unified Modeling Language die Frage nach anwendungsspezifischen Erweiterungen derselben. Bei der Entwicklung einer Enterprise-*Java-Bean*-Komponente will man zum Beispiel bei der Modellierung nicht auf die Standardklasse *Component* des UML-Metamodell zurückgreifen, sondern ein spezifisches Konstrukt einsetzen, welches schon alle Eigenschaften einer *Bean* mit beschreibt. Dieses neue Konstrukt soll die *Component* um Enterprise-*Bean*-Eigenschaften, wie zum Beispiel die Unterscheidung in *Session*- und *Entitybean*, erweitern.

Leider unterstützt die UML keine speziellen Konstrukte zur Modellierung von *Java Enterprise Beans*. Dafür bietet sie aber Möglichkeiten sich selbst um anwendungsspezifische Konstrukte zu erweitern, das sogenannte *UML-Profiling*, welches im *Profile*-Paket der UML2 Infrastructure beschrieben ist.

In einem Profil für ein spezifisches Anwendungsgebiet werden mehrere speziell angepasste Konstrukte der UML zusammengefasst. Es ist nicht möglich das UML-Metamodell mit einem Profil zu verändern. Sondern es ist nur möglich, ein eigenes Metakonstrukt als sogenanntes Stereotyp eines UML-Metamodellelements zu definieren [2, Seite 245].

Ein spezielles UML-Profil wird wiederum mit UML selbst modelliert. Ein UML-Profil wird durch ein Paket dargestellt, das das Schlüsselwort `<<profile>>` erhält. Innerhalb dieses Paketes werden die zu spezialisierenden UML Sprachkonstrukte als Metaklassen eingefügt. Der neue spezialisierte Metatyp wird dann zusammen mit dem Schlüsselwort `<<stereotype>>` angegeben. Der Zusammenhang zwischen der erweiterten Basis-Metaklasse und dem neuen Metatyp wird durch einen *Extension*-Pfeil ausgedrückt. Eine optionale Auszeichnung `{required}` am Erweiterungspfeil gibt an, ob der Einsatz des neuen Metatyps beim Einsatz des Profils zwingend ist oder ob dem Anwender immer noch die Möglichkeit gelassen wird, die ursprüngliche Metaklasse einzusetzen.

Wie kann man nun eine Basisklasse spezialisieren? Das UML Metamodell führt einen Stereotyp als eine Spezialisierung der Metaklasse *Class* ein. Stereotypen haben somit Klasseneigenschaften, das heißt sie dürfen weiter spezialisiert werden und weitere Attribute besitzen. Im Beispiel der *Java Enterprise Beans* könnte man also einen abstrakten Stereotyp *EnterpriseBean* einführen, von dem zwei Spezialisierungen abgeleitet werden, nämlich die schon erwähnten *Session*- und *EntityBeans*. Dann könnte man die *SessionBean* noch um ein Attribut erweitern, das beschreibt, ob die *Bean* zustandsbehaftet oder zustandslos sein soll. Abbildung 5 zeigt das Enterprise *Java Bean* Beispiel [2].

Abbildung 6 zeigt die Verwendung des Profils zur Beschreibung von *EJB*. Die Nutzung eines UML Profils in einem UML Modell wird durch eine spezielle Form des Import-Mechanismus angegeben. Ein Paket, das ein Profil verwenden möchte, kennzeichnet den Import des Profilpaketes mit dem Schlüsselwort `<<apply>>`. Falls im UML Profil die Auszeichnung `{required}` angegeben ist, so muss

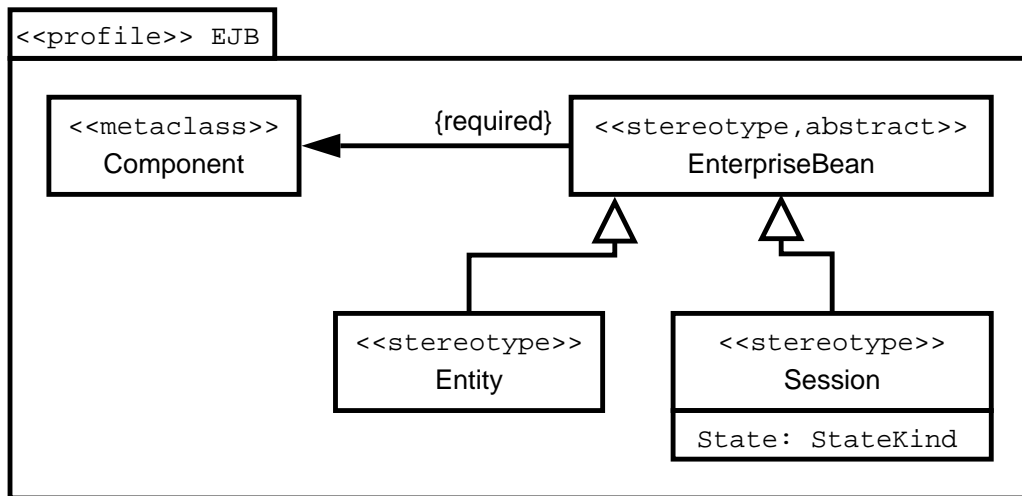


Abbildung 5: UML-Profil einer Enterprise Java Bean

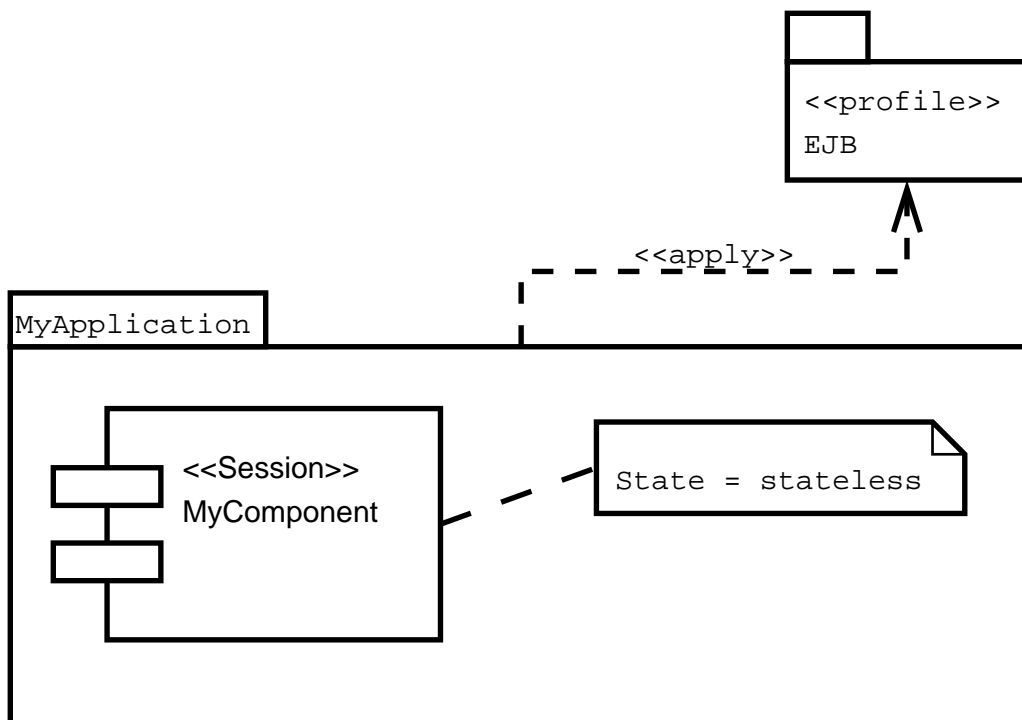


Abbildung 6: Anwendung des UML-Profiles für EJB

in dem Paket, das das Profil verwendet, der neu eingeführte Metatyp verwendet werden und es darf nicht mehr die ursprüngliche Basisklasse zum Einsatz kommen.

2.3.2 OMG Modellierungs- und Metadatenstandards

In diesem Abschnitt werden die Zusammenhänge zwischen den verschiedenen Modellierungs- und Metadatenstandards der Object Management Group (OMG) erläutert.

METAEBENE	MOF BEZEICHNUNGEN	BEISPIELE
M3	Meta-Metamodell	MOF Modell
M2	Metamodell Meta-meta Daten	UML Metamodell CWM Metamodell
M1	Modell	UML Modelle
M0	Objekt, Daten	modellierte Systeme Warehouse-Daten

Abbildung 7: OMG Metadaten Architektur

Neben dem Standard für das CWM und die UML hat die OMG unter anderem auch das Meta Object Framework (MOF) veröffentlicht [16]. Das MOF-Modell ist ein Meta-Metamodell mit dem Metamodelle, wie zum Beispiel das CWM und die UML, beschrieben werden können. Tabelle 7 verdeutlicht, wie die verschiedenen Ebenen der Modellierung zusammenhängen.

Die folgende Beschreibung des MOF bezieht sich zuerst auf die Version 1.3, die auch dem CWM zu Grunde liegt. Das Konzept der Metadaten Architektur, wie es in Tabelle 7 beschrieben ist, entstand erst nachdem die UML etabliert war. Dies ist der Grund, warum das MOF-Modell auf grundlegenden Konzepten und Konstrukten der UML basiert. Die grundsätzliche Idee aber ist eigentlich, dass Metamodelle wie die UML oder das CWM mit in MOF beschriebenen Konstrukten definiert werden. Das MOF-Modell führt keine eigene graphische Notation ein, sondern verwendet die aus UML bekannte Notation der Klassendiagramme. Das MOF-Modell baut auf Elementen wie Klassen, Objekten, Attributen und Operationen auf. Eine *Klasse* beschreibt eine Menge von *Objekten*, die die gleichen *Attribute*, *Operationen* und Semantik teilen. Alle Objekte sind Instanzen einer Klasse. Die grundlegenden Beziehungen, die das MOF-Modell beschreibt, sind Assoziationen und Generalisierungen. Eine *Assoziation* ist eine Beziehung zwischen zwei Klassen, die eine Verbindung zwischen Objekten beschreibt. Eine *Generalisierung* ist eine Beziehung zwischen einem allgemeineren und einem spezielleren Element. Eine Instanz des spezielleren Elements kann immer dann verwendet werden, wenn auch das allgemeinere Element zugelassen wäre. Diese Konzepte und Konstrukte sind die gleichen wie die aus UML, aber liegen auf einer höheren Metaebene. Dies bedeutet, dass MOF Assoziationen auf der Metaebene M3 liegen und verwendet werden um Metamodelle (M2) zu definieren, während UML Assoziationen auf der Ebene M2 benutzt werden um Modelle (M1) zu beschreiben. Zwar ist das MOF Modell mit den statischen Strukturen der UML identisch, aber es existieren in den Versionen 1.* noch kleinere Differenzen: Zum einen müssen MOF Assoziationen binär sein und zum zweiten existieren in MOF keine Assoziationsklassen [20].

Es existieren mehrere von der OMG veröffentlichte Spezifikationen, die auf MOF aufbauen und verschiedene Transformationen MOF-konformer, also auf dem MOF Modell basierender, Metamodelle beschreiben. Das **MOF-to-IDL-Mapping** [20, Seite 67ff] definiert eine standardisierte Abbildung eines Metamodells, das MOF konform ist, nach der Corba IDL (Interface Definition Language). Das entstandene Interface ermöglicht dem Benutzer, Instanzen des Metamodells (also Metaobjekte oder Modelle auf Ebene M1) über die Corba Schnittstelle, somit auch aus allen Programmiersprachen heraus zu erzeugen, zu bearbeiten oder darauf zuzugreifen etc. **XML Metadata Interchange (XMI)** [20,

Seite 69f] beschreibt, wie die Metadaten MOF-konformer Metamodelle in einem standardisiertem Format, basierend auf XML, gespeichert werden können. XMI definiert auch Regeln, wie aus einem beliebigen MOF Metamodell, eine **XML DTD** (Document Type Definition) [20, Seite 70ff] erzeugt werden kann. Diese DTD kann dann ein XML Parser dazu verwenden, die syntaktische Korrektheit und die strukturelle Übereinstimmung eines XML Dokumentes mit dem zugrundeliegenden Modell zu überprüfen.

Alle Metamodelle, die auf MOF basieren, das trifft insbesondere auf die UML [11, Abschnitt 2.2.1.1] und das CWM [12, Abschnitt 4.2.2] zu, können die im vorherigen Absatz vorgestellten Techniken einsetzen, um ihre Modelle in einer standardisierten Form zu beschreiben. Dies bedeutet, dass ein beliebiges UML oder auch CWM Modell mit den gleichen Tools erstellt, dargestellt und auch gespeichert werden kann.

Wie oben schon erwähnt, sind das MOF und die UML Grundelemente doch nicht absolut identisch. Mit der Standardisierung der Versionen 2 von UML und MOF packt die OMG auch dieses Problem an. Ihr Ziel ist es die verschiedenen Standards zu vereinheitlichen. Inzwischen ist auch für das MOF-Modell eine Vorabspezifikation der Version 2 veröffentlicht worden [16], auf die im weiteren Bezug genommen wird. Auch diesmal zeigt sich, dass die Spezifikation zur UML2 vor der zu MOF abgeschlossen wurde, aber diesmal schon mit dem Ziel, beide Standards sauber in die Metadatenarchitektur einzugliedern. Ein Blick in die Spezifikation der MOF Version 2 [16, Abschnitt 7.3] zeigt, dass die Pakete *Core::Basic* und *Core::Constructs* der UML2 Infrastructure von MOF importiert werden. Dies sind genau die Pakete, die wiederum alle aus UML bekannten grundlegenden Konstrukte und Abstraktionen, wie zum Beispiel Klassen, Assoziationen und Generalisierung etc. definieren [17]. Das MOF-Modell baut somit auf den gleichen Grundlagen auf, wie die UML2. Beider Kern ist die UML2 Infrastructure. Somit ist die UML2 Infrastructure ein Meta-Metamodell [2, Seite 69] und liegt auf der Metaebene M3 und es ist korrekt, dass das ebenfalls auf Ebene M3 liegende MOF Modell Pakete der UML Infrastructure importiert. Das MOF-Modell ergänzt die UML2 Infrastructure dann nur noch um einige wenige weitere Sprachkonstrukte. Die UML2 Infrastructure ist der Werkzeugkasten, der die Sprachelemente enthält, um weitere Metamodelle zu bauen. Ein Beispiel ist die gesamte UML2 selbst, die ja in einer weiteren Spezifikation — der UML2 Superstructure — mit Hilfe der Sprachkonstrukte aus der UML2 Infrastructure beschrieben ist.

Abbildung 8 zeigt wie in Zukunft der Zusammenhang zwischen den verschiedenen Metadatenstandards der OMG aussehen wird [17]. Zentral wird das *Core*-Paket aus der UML2 Infrastructure stehen, auf dem zur Zeit schon die gesamte UML2 — spezifiziert in der UML2 Superstructure — und das MOF basieren. Bezüglich der Bezeichnung des Profilverpaketes deutet sich jedoch ein Widerspruch an. Abbildung 8 und auch [17, Figure 1] weisen das *Core*- und *Profile*-Paket als zwei getrennte Pakete in der UML2 *InfrastructureLibrary* aus. In Abschnitt 13 der UML2 Infrastructure [17] wird das Profilverpaket aber nun als *Core::Profile* bezeichnet, was suggeriert, dass es Teil des *Core*-Paketes ist.

2.3.3 Entwicklung der Profile von UML Version 1.3 nach 2.0

Die Version 1.3 der Unified Modelling Language kennt noch keine Profile. Sie definiert aber die aus dem CWM schon bekannten und in Abschnitt 2.2 beschriebenen *Stereotypes* und *TaggedValues*. Abbildung 9 zeigt den UML 1.3 Erweiterungsmechanismus im Metamodell. Ein Stereotyp ist nach dem Metamodell ein *GeneralizableElement*. Er kann also spezialisiert werden oder als Oberklasse in einer Spezialisierung dienen. Er hat aber nicht die Eigenschaften, die man von einer *Class* erwarten würde: er kann zum Beispiel keine Assoziationen eingehen.

Mit der UML Version 1.4 [14] wird der Begriff Profil eingeführt. Er entspricht von der Semantik aus schon dem Profiling aus UML2, das heißt mehrere für ein Anwendungsgebiet spezifischen Anpassungen

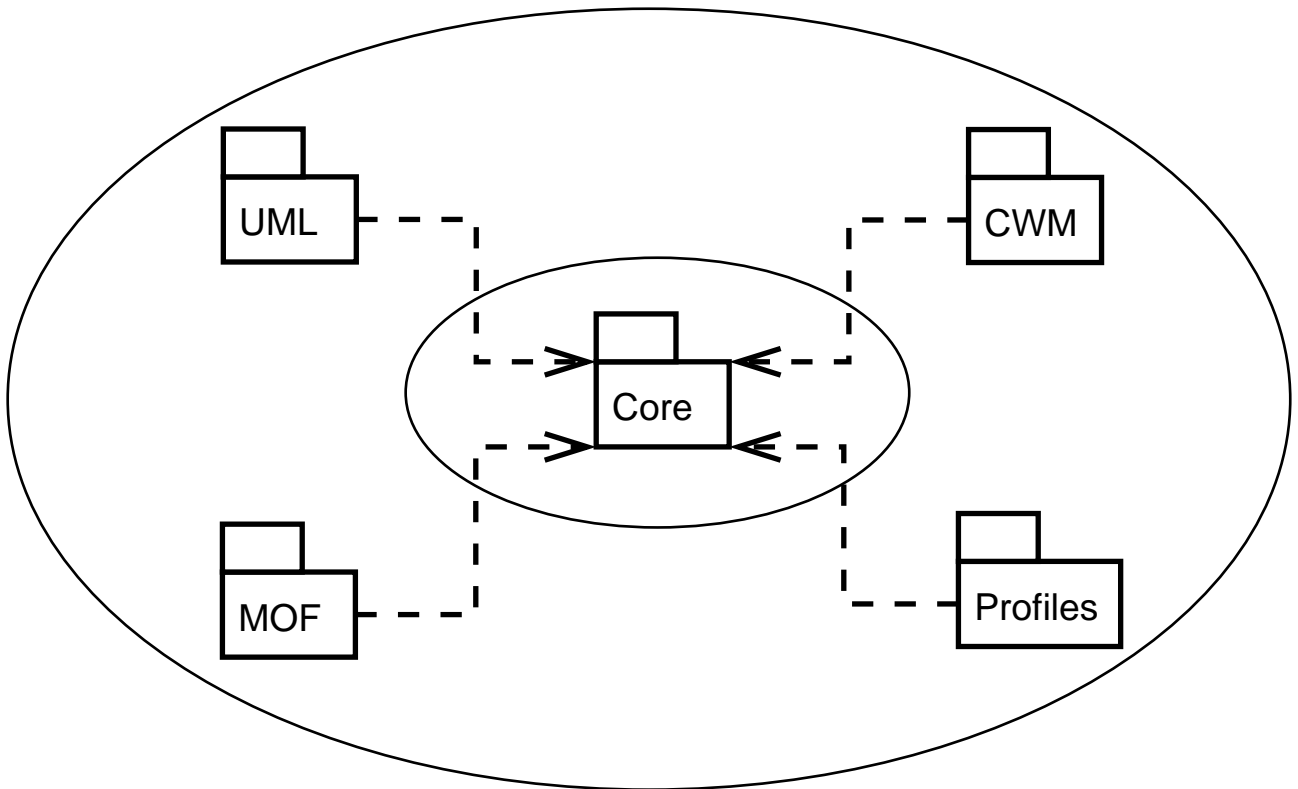


Abbildung 8: Zusammenspiel der neuen OMG Modellierungsstandards

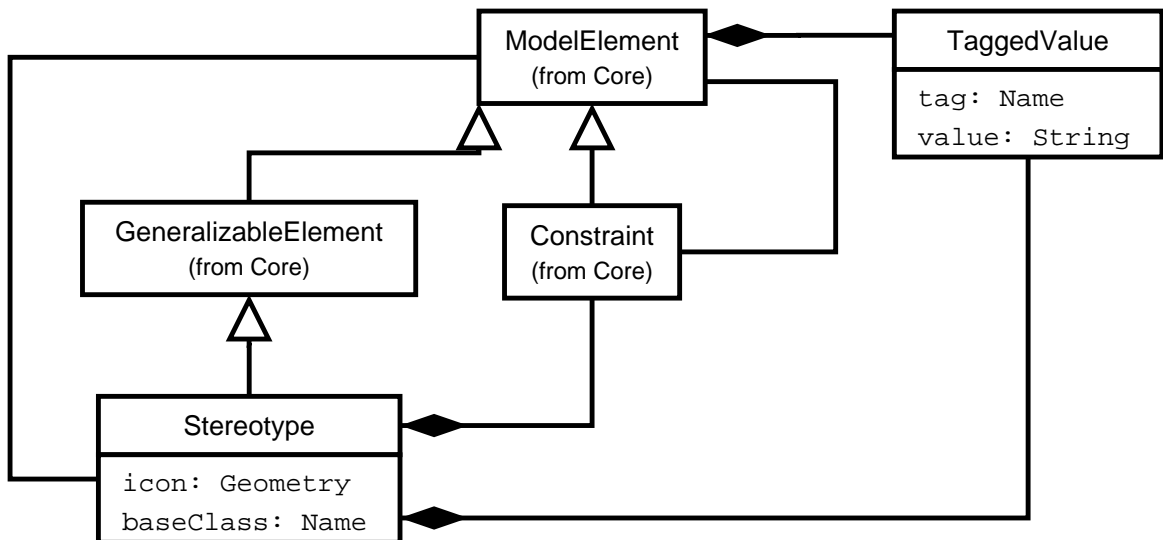


Abbildung 9: UML Version 1.3 Erweiterungsmechanismus

werden in einem besonderen Paket (nämlich dem Profilkpaket, ausgezeichnet mit <<profile>>) zusammengefasst. Anschließend kann das so definierte Profil mit einem besonderen Importmechanismus, der in UML 1.4 mit <<appliedProfile>> gekennzeichnet ist, in ein Modellpaket importiert werden. Die möglichen Anpassungen entsprechen weitestgehend aber noch denen aus UML 1.3. Ein *ModelElement* kann um *Stereotypes*, *TaggedValues* und *Constraints* ergänzt werden. Neu eingeführt ist die Angabe einer *TagDefinition*, die es ermöglicht den Typ eines *TaggedValues* zu definieren.

Auch wenn die Version 2 der Unified Modelling Language den Erweiterungsmechanismus neu beschreibt, stecken viele in UML 1.4 eingeführten Techniken darin. Ein ausgezeichnetes Profilkpaket und der besondere Importmechanismus sind von UML 1.4 übernommen worden. Das Konzept des Stereotyps hat aber weiter an Mächtigkeit gewonnen. Ein Stereotyp ist ein *Classifier* und erbt somit alle grundlegenden Eigenschaften, die man von Klassen erwartet. In Abschnitt 2.3.1 ist das Profiling von der UML Version 2 schon genau beschrieben.

2.3.4 Erweiterung des CWM mit UML2 Profilen

Vergleicht man die Erweiterungsmechanismen der Unified Modeling Language in der Version 1.3 (Abbildung 9) und des Common Warehouse Metamodel (Abbildung 3 oder ausführlicher [12, Figure 7-3-3]), das ja auf dieser UML Version basiert, erkennt man, dass die CWM-Erweiterungstechniken zu denen aus UML absolut identisch sind.

Ein beliebiges *ModelElement* kann um einen *Stereotype* ergänzt werden. An diesen Stereotyp können beliebige *TaggedValues* und *Constraints* angehängt werden. Wie im vorherigen Abschnitt erwähnt, bietet das Konzept des Stereotyps aber nur sehr einfache Ausdrucksmöglichkeiten.

Es gibt nun zwei Gründe, die sehr deutlich darauf hinweisen, dass zukünftige CWM Versionen als Erweiterungsmechanismus auf das in UML2 vorgestellte Profiling setzen werden:

1. In Abschnitt 2.3.2 habe ich schon vorgestellt, dass die OMG anstrebt ihre verschiedenen Modellierungsstandards einander anzupassen. Ein erster Schritt hierzu ist mit der Vereinheitlichung der MOF2 mit der UML2 Infrastructure schon geschehen. Abbildung 8 zeugt auch schon davon, dass zukünftige CWM Versionen auf dem UML2 Infrastructure *Core*-Paket basieren werden.

Noch deutlicher ist eine Aussage auf Seite 171 der UML2 Spezifikation [17], die besagt, dass der Profil-Mechanismus mit allen Metamodellen genutzt werden kann, die mit MOF erzeugt worden sind, was insbesondere die UML 2.0 und neue Versionen des CWM mit einschließt.

In diesem Zusammenhang wurde in Abschnitt 2.3.2 schon erwähnt, dass unklar ist, ob nun das *Profile*-Paket Bestandteil des *Core*-Paketes ist oder nicht. Die Frage ist nun, wie der Profilingmechanismus mit in andere Metamodelle übernommen wird: automatisch durch das Importieren des *Core*-Paketes oder ob es einer expliziten Behandlung in den Metamodellen bedarf?

Unabhängig davon ist aber zu erwarten, dass auch das *Profiling* für das CWM einsetzbar sein wird. Die UML basiert nach Abbildung 8 auch nur auf dem *Core* und kann mit Profilen erweitert werden, ähnliches ist auch für das CWM zu erwarten.

2. Wie in Abschnitt 2.3.1 beschrieben, hat sich mit der neuesten UML Version der Profilingmechanismus geändert. CWM hat aber in der bisherigen Version den Erweiterungsmechanismus von UML übernommen. Also kann man davon ausgehen, dass auch neue CWM Versionen, die auf UML2 oder besser gesagt im Rahmen der Anpassung der verschiedenen Modellierungs- und Metadatenstandards auf MOF2 basieren, das UML2 Profiling mit eingebaut haben.

In den letzten Abschnitten wurden Ansatzpunkte beschrieben, die andeuten, wie die noch nicht spezifizierte neue Version des Common Warehouse Metamodel aussehen könnte. Besonders der neue Profiling Mechanismus wird vermutlich auch Bestandteil neuerer CWM Versionen werden. Deshalb sehe ich die Möglichkeit, das aktuelle CWM mit den neuen Profiltechniken zu erweitern, die erst mit der Unified Modelling Language Version 2.0 eingeführt worden sind.

2.4 Auswahl einer Erweiterungstechnik

In der Studienarbeit von Alexander Haag [8] werden zwei Ansätze beschrieben, wie man mit dem CWM eine imperfekte relationale Spalte modellieren kann. Alexander Haag verwendet dort die in den CWM Spezifikationen beschriebenen Verfahren der Erweiterung zum einen mit Hilfe von Stereotypes und TaggedValues, sowie über objektorientierte Modellierungsmöglichkeiten (siehe Abschnitt 2.2).

In dieser Studienarbeit werde ich davon ausgehen, dass das Profiling aus der UML2 schon für die aktuelle CWM Version verwendbar wäre. In Abschnitt 2.3.4 habe ich aufgezeigt, warum sich dies anbietet. Ich werde also die benötigten Erweiterungen des Common Warehouse Metamodel mit den Techniken des neuen Erweiterungsmechanismus aus der Unified Modeling Language Version 2.0, dem UML2-Profiling, entwickeln. Diese Entscheidung wird, wie die Entwicklung der UML und CWM Versionen zeigt, vermutlich ein richtiger Schritt in die Zukunft sein.

3 Imperfektion in Datenbanken

Dieses Kapitel beschreibt, welche Ansätze bisher existieren um mit Imperfektion in Verkehrsdaten aus dem OVID-Projekt [19] umzugehen. Dazu werde ich mich auf die Arbeiten von Erik Koop, Andreas Merkel und Alexander Haag beziehen.

Erik Koop hat in seiner Diplomarbeit [9] untersucht, welche Daten im Verkehrsumfeld mit Imperfektion behaftet sind und wie diese Imperfektionen in Datenbanken erhalten bleiben können. Dazu fasst er auch die grundlegenden Theorien über Imperfektion zusammen. Andreas Merkel hat dann in seiner Studienarbeit [10] das Kontextmodell, ein Konzept, das relationale Schemata so erweitert, dass imperfekte Daten abgebildet werden können, auf konkrete Messdaten aus dem Verkehrsbereich angewendet. In der Studienarbeit von Alexander Haag [8] wurde begonnen das Common Warehouse Metamodel darauf aufbauend so zu erweitern, dass die Informationen über Imperfektionen auch beim Metadaten austausch erhalten bleiben können.

3.1 Theorie über Imperfektion

Das Wort Imperfektion stammt aus dem Lateinischen. Das dem Wortstamm zugrunde liegende lateinische *imperfectus* bedeutet „unvollständig“, „unvollendet“ und „mangelhaft“. Imperfektes Wissen ist somit Wissen, das einen Sachverhalt nur unvollständig oder eben unvollendet beschreibt.

In [9] wird basierend auf der Dissertation von René Witte [22] imperfekte Informationen in drei Unterkategorien eingeteilt, die im Rahmen des OVID-Projektes von Bedeutung sein werden. Wichtige Bedeutung erhält diese Einteilung auch, da für die verschiedenen Arten von imperfekten Informationen unterschiedliche Theorien benötigt werden, um sie formal beschreiben zu können.

Unsichere Informationen: Eine Information ist unsicher, wenn nicht entschieden werden kann, ob sie wahr oder falsch ist. So ist beispielsweise die Meldung eines Staus auf der Autobahn A8 zwischen Karlsruhe und Pforzheim von einem Staumelder grundsätzlich eine unsichere Information. Der Wahrheitsgehalt der Information kann aber durch das Abfragen weiterer Quellen gefestigt oder widerlegt werden.

Unschärfe Informationen: Unschärfe Informationen entstehen, wenn eine Einteilung in Klassen nicht anhand scharfer Grenzen möglich ist. Man kann zum Beispiel die Qualität des Verkehrszustandes anhand des Verkehrsflusses beschreiben. Dazu wird der Verkehrsfluss in sechs Klassen eingeteilt: *frei*, *teilgebunden*, *synchron*, *stockend*, *gestaut* und *stehend*. Der Übergang zwischen den einzelnen Klassen ist fließend und kann kaum exakt festgelegt werden [21].

Ungenau Informationen: Ungenau wird eine Information bezeichnet, die nur durch Intervallgrenzen oder andere ähnliche grobkörnige Einheiten angegeben werden kann. Ein Beispiel sind physikalische Messungen, die immer von Messungenauigkeiten betroffen sind.

Es existieren unter anderem zwei unterschiedliche formale Theorien über die Darstellung imperfekter Informationen. Zum einen die Wahrscheinlichkeitstheorie und zum zweiten die Fuzzy-Theorie. In den nächsten beiden Abschnitten werden beide Theorien kurz vorgestellt, eine ausführlichere Beschreibung findet man in [9] und der Literatur über Imperfektion in Informationssystemen — wie zum Beispiel in [23].

Der Ausgangspunkt bei der Betrachtung beider Theorien, ist das Einteilen von Sachverhalten in Mengen. Diese Einteilung ist bei imperfekten Informationen nicht so einfach möglich, beide Theorien bieten nun verschiedene Ansätze, um mit diesem Problem umzugehen.

3.1.1 Wahrscheinlichkeitstheorie

Die Wahrscheinlichkeitstheorie — auch Probabilitätstheorie genannt — basiert auf der klassischen Aussagenlogik. In der klassischen Aussagenlogik ist eine Aussage entweder wahr oder falsch. Dies bedeutet, dass für ein Element immer sicher entschieden werden kann, ob es nun in einer Teilmenge enthalten ist oder nicht. Die charakteristische Funktion zur Beschreibung der Teilmengenzugehörigkeiten darf nur die Werte 0 (Element ist nicht in der Menge enthalten) und 1 (Element ist in der Menge enthalten) annehmen. Sind die Informationen aber imperfekt, ist es eben dies nicht so einfach möglich. Ein einfaches und deutliches Beispiel sind die oben erwähnten unsicheren Informationen, deren Wahrheitsgehalt natürlicherweise nicht sicher festzulegen ist, und somit auch keine 100%-ige Einteilung in wahre und falsche Aussagen ermöglichen.

In der Wahrscheinlichkeitstheorie wird deshalb eine Aussage mit dem Grad der Überzeugung belegt, der beschreibt für wie wahrscheinlich man die Aussage hält — für wie wahrscheinlich man es hält, dass ein Element sicher in einer Teilmenge enthalten ist. Das Wissen, das der Zuordnung der Wahrscheinlichkeiten zu Grunde liegt, wird **Evidenz** genannt. Zur formalen Beschreibung der Wahrscheinlichkeiten von Zugehörigkeiten zu bestimmten Mengen verwendet man die Begriffe Wahrscheinlichkeitsmaß und Wahrscheinlichkeitsverteilung [9].

Wahrscheinlichkeitsmaß: Das Wahrscheinlichkeitsmaß P gibt an, mit welcher Wahrscheinlichkeit $P(A)$ ein beliebiges Element x aus einer Grundmenge Ω ($x \in \Omega$) zu einer wohl-definierte Teilmenge $A \subseteq \Omega$ gehört. Es ist eine Zuordnung der Potenzmenge einer Grundmenge Ω in das Einheitsintervall: $P: 2^\Omega \rightarrow [0, 1]$.

Wahrscheinlichkeitsverteilung: Die Wahrscheinlichkeitsverteilung ist definiert als die Zuordnung der Grundmenge Ω in das Einheitsintervall: $p: \Omega \rightarrow [0, 1]$,

$$\forall x \in \Omega: p(x) = P(\{x\}).$$

Die Wahrscheinlichkeitstheorie eignet sich gut für den Umgang mit unsicheren Informationen. Basierend auf vorhandenem Wissen wird der Wahrheitsgehalt einer Information mit Wahrscheinlichkeiten belegt. Ist zum Beispiel bekannt, dass bisherige Staumeldungen zu 80% zutrafen, so wird auch eine neueintreffende Meldung erstmal als zu 80% wahr betrachtet, bis weitere Daten diesen Wert be- oder widerlegen.

Interpretiert werden muss die Angabe, dass eine Staumeldung zu 80% wahr ist, wie folgt: In 80% der Meldungen liegt eine Verkehrssituation vor, die garantiert als Stau bezeichnet werden kann. In den anderen 20% der Meldungen ist aber definitiv kein Stau auf der gemeldeten Strecke. Diese strikte Einteilung liegt in der zugrundeliegenden Aussagenlogik begründet. Deshalb ist auch eine Interpretation der Art, dass die gemeldete Verkehrssituation vielleicht von 80% der Verkehrsteilnehmer als Stau betrachtet werden könnte und gleichzeitig vom Rest nicht, eben nicht zulässig.

3.1.2 Fuzzy-Theorie

Wie oben erwähnt teilt die klassische Aussagenlogik Sachverhalte in Mengen mit scharfen Grenzen ein. Damit kann eindeutig bestimmt werden, ob eine Aussage über einen Sachverhalt zutrifft oder nicht. Die Fuzzy-Logik erweitert nun die Aussagenlogik um Zwischenstufen zu wahr und falsch, beziehungsweise teilt Sachverhalte in unscharfe Mengen, den **Fuzzy-Mengen** ein.

Eine Menge wird durch die sogenannte charakteristische Funktion beschrieben. Bei einer klassischen Mengen ist die Bildmenge der charakteristischen Funktion auf die Werte 0 (Element ist nicht in

der Menge enthalten) und 1 (Element gehört zu der Menge) beschränkt. In der Fuzzy-Theorie, ist die Bildmenge einer normalisierten charakteristischen Funktion zur Definition einer Fuzzy-Menge das gesamte kontinuierliche Einheitsintervall $[0, 1]$. Die Fuzzy-Menge gibt also den Zugehörigkeitsgrad eines Elementes zu einer Menge an.

Betrachtet man wieder das Beispiel der Staumeldungen. Die Verkehrssituation auf den Autobahnen könnte man in zwei Klassen (Teilmengen) einteilen: „*Stau*“, „*kein Stau*“. Eine Verkehrssituation auf der Autobahn A8 zwischen Pforzheim-West und Pforzheim-Nord, die man als stockenden Verkehr bezeichnen würde, liegt dem gesunden Menschenverstand nach, nicht in der Klasse „*kein Stau*“, aber auch nicht in der Klasse „*Stau*“. In der Fuzzy-Theorie kann man diese Dilemma nun lösen, in dem man die Verkehrssituation zu bestimmten Zugehörigkeitsgraden beiden Klassen zuordnet. Hier wäre vielleicht eine Zuordnungsgrad von 0,8 zur Klasse „*Stau*“ und 0,2 zu Klasse „*kein Stau*“ angemessen. Die Bestimmung der Zugehörigkeitsgrade durch die charakteristischen Funktionen der Fuzzy-Mengen „*Stau*“, „*kein Stau*“ erfordert weiteres Wissen.

So wie die Wahrscheinlichkeitstheorie gut für die Modellierung von unsicheren Informationen geeignet ist, lassen sich mit der Fuzzy-Theorie unscharfe Begriffe gut beschreiben. Mit ihrer Hilfe lassen sich die Ungenauigkeiten umgangssprachlicher Ausdrücke formalisieren. Man spricht dann von linguistischen Variablen.

Linguistische Variable Eine linguistische Variable nimmt als Werte verbale Begriffe, sogenannte Terme, auf. Definiert wird eine linguistische Variable x über der Grundmenge U durch die Angabe der Termmenge $T(x)$ mit den möglichen Termen (den linguistischen Werten) und der Definition unscharfer Mengen über U , die die Bedeutung der Terme repräsentieren. So kann für jedes Element $e \in U$ und jedem Term $t \in T(x)$ der Zugehörigkeitsgrad von e zu t bestimmt werden.

Beispielsweise könnte die unscharfe Information über den Verkehrsfluss nun mit Hilfe einer linguistischen Variablen wie folgt detailliert modelliert werden: Der linguistischen Variablen *Verkehrsfluss* ordnet man die Termmenge

$$T(\text{Verkehrsfluss}) = \{ \text{„frei“}, \text{„teilgebunden“}, \text{„synchron“}, \text{„stockend“}, \text{„gestaut“}, \text{„stehend“} \}$$

zu. Zu jedem Term müsste dann noch eine Zugehörigkeitsfunktion definiert werden, die den Zugehörigkeitsgrad einer Verkehrssituation, die zum Beispiel durch die Verkehrstärke Q und die Verkehrsdichte K beschrieben ist, zu diesem Term bestimmt.

3.2 Das Kontextmodell

Eine Möglichkeit mit ungenauen und unscharfen Informationen in einer relationalen Datenbank umzugehen ist das in diesem Abschnitt vorgestellte **Kontextmodell**. Es ist in [10] verwendet worden, um imperfekte Daten aus dem OVID-Projekt in einer relationalen Datenbank darzustellen.

Kontexte sind ein Konzept zur Klassifikation von Daten. Das Ziel der Einteilung des Wertebereiches eines Attributes in Klassen ist es, die Daten besser veranschaulichen zu können: Sei es damit der Benutzer einen besseren Überblick über die Daten bekommt oder weil der exakte Attributwert sowieso als ungenau betrachtet werden muss und es ausreicht, die Intervallgrenzen zu kennen, in denen der exakte Wert liegt.

3.2.1 Scharfe Kontexte

Bei scharfen Kontexten wird der Wertebereich eines Attributes in Äquivalenzklassen eingeteilt. Dies bedeutet auch, dass ein Attributwert in genau eine Klasse fällt und dass alle Klassen den gesamten Wertebereich abdecken. Es ist somit immer möglich für einen Attributwert seine Äquivalenzklasse zu bestimmen, ebenso kann immer entschieden werden, ob zwei Attributwerte ähnlich sind, das heisst, ob sie in derselben Klasse liegen.

Ungenaue Informationen lassen sich nun mit scharfen Kontexten modellieren. Dazu erfolgt eine Einteilung des Attributwertebereiches in Klassen, die der natürlichen Ungenauigkeit der Information entsprechen. Da es im Normalfall nicht möglich ist, die Länge eines Staus genau anzugeben, beziehungsweise die Länge eines Stau in den seltensten Fällen statisch ist, könnte man beispielsweise den Wertebereich des Staulängenattributes in Klassen von einem Kilometer Länge einteilen. Ein Stau der Länge 2,465 km würde demnach in die Klasse der Staus der Längen zwischen 2 und 3 km fallen.

Wäre aber nicht auch eine Einteilung in die Klassen

$$\{\{kurz\}, \{mittel\}, \{lang\}\}$$

möglich? In welche Klasse fällt nun der Stau der Länge 2,465 km?

Ist eine exakte Einteilung in scharfe Klassen nicht möglich oder nicht gewünscht, dann hat man es mit unscharfen Informationen zu tun. Dem wird das Kontextmodell mit einer Einteilung in unscharfe Kontexte gerecht.

3.2.2 Unscharfe Kontexte

Ein unscharfer Kontext führt keine scharfen Grenzen zwischen Klassen ein, sondern unterstützt den fließenden Übergang zwischen verschiedenen Klassen. Ein Element wird nicht mehr exakt einer Klasse zugeordnet, sondern liegt zu gewissen Anteilen in mehreren Klassen.

Zur formalen Beschreibung unscharfer Klassen dient das Konzept der **Linguistischen Variablen** aus der Fuzzy-Theorie (Abschnitt 3.1.2): Jedem Attribut des relationalen Schemas wird eine linguistische Variable zugeordnet. Damit wird der Wertebereich des Attributes in unscharfe Mengen eingeteilt, man erhält somit eine Aufteilung in unscharfe Kontexte.

Die scharfen Kontexte sind ein Spezialfall der Einteilung in unscharfe Kontexte. Eine scharfe Klassentrennung kann unter Einsatz von linguistischen Variablen erfolgen, soweit deren Zugehörigkeitsfunktionen genau dann eins sind, falls der Attributwert zur Klasse gehört und die null sind, falls der Term nicht auf das Element zutrifft. Man erreicht somit eine Einteilung des Attributwertebereiches in klassische Mengen.

3.3 Erste Erweiterungen des CWM für die Modellierung imperfekter Daten

In [8] werden erste Ansätze beschrieben wie das Common Warehouse Metamodel um Imperfektion erweitert werden kann. Beide auch in dieser Arbeit vorgestellten standardisierten Erweiterungstechniken des Common Warehouse Metamodel (siehe Abschnitt 2.2) werden in [8] verwendet, um das Kontextmodell in das *Relational*-Paket des CWM zu integrieren. Die folgenden Abschnitte geben einen kurzen Überblick über die modellierten Erweiterungen.

3.3.1 TaggedValues und Stereotypes

Der erste Ansatz, der in [8] untersucht wurde, war eine Erweiterung des CWM mit Hilfe von Stereotypes und TaggedValues. Verkehrsdaten aus dem OVID-Projekt sind bekanntlich mit dreierlei Arten von Imperfektion behaftet (Abschnitt 3.1), deshalb werden auch drei Stereotypes eingeführt, die einzelne Spalten als imperfekt kennzeichnen: <<Uncertain>>, <<Imprecise>> und <<Fuzzy>>. Ein weiterer Stereotyp kennzeichnet eine Tabelle als imperfekt, wenn mindestens eine Spalte schon als imperfekt beschrieben ist (<<Imperfect Table>>).

Dem Stereotyp <<Uncertain>> wird unter anderem der verbindliche TaggedValue „Wahrscheinlichkeitsfunktion“ zugeordnet, der den Grad des Wahrheitsgehaltes des Attributwertes beschreibt basierend auf der Wahrscheinlichkeitstheorie (siehe Abschnitt 3.1.1). Der Stereotyp <<Fuzzy>> basiert auf der Fuzzy-Theorie (Abschnitt 3.1.2) und erhält somit eine Zugehörigkeitsfunktion, die in einem weiteren TaggedValue gespeichert wird. Auch der für Ungenauigkeit zuständige Stereotyp <<Imprecise>> bedarf einer Zugehörigkeitsfunktion, die aber nur die Werte 0 und 1 annimmt.

Wie schon bei der Vorstellung dieser Erweiterungstechnik in Abschnitt 2.2 erwähnt, erlaubt der Wert eines TaggedValues nur die Speicherung eines untypisierten Stringwertes. Damit eine automatische Weiterverarbeitung der Daten, zum Beispiel eine Berechnung der Zugehörigkeitsfunktionen, möglich ist, müssen nach [8] die imperfekten Datentypen und Zugehörigkeitsfunktionen explizit als Klassen modelliert werden. Dort wird dann auch ein zweiter Ansatz vorgestellt, der die objektorientierten Techniken nutzt, um das CWM zu erweitern.

3.3.2 Objektorientierte Erweiterung

Die Grundidee hinter dem zweiten Erweiterungsansatz ist es, die imperfekten Spalten als eigene Metaklassen des CWM zu modellieren und so dann auch eine automatisierte Berechnung von Zugehörigkeitsfunktionen und ähnlichem zu ermöglichen.

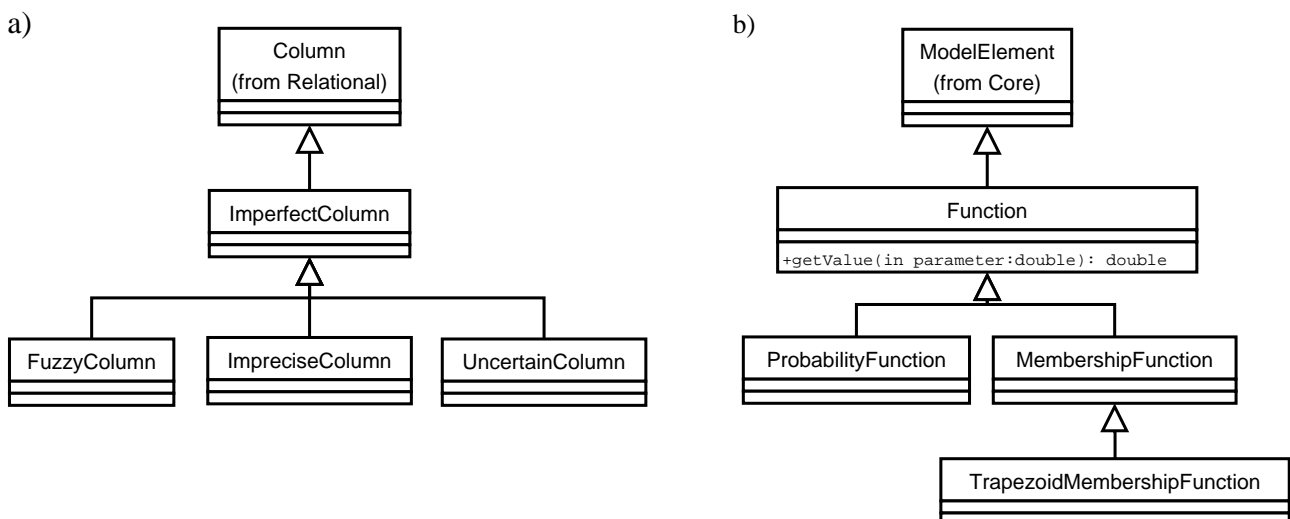


Abbildung 10: OO-Erweiterung des Relational-Pakets um Imperfektion

Wie aus Abbildung 10a ersichtlich ist, bildet die *Column*-Metaklasse aus dem *Relational*-Paket die Basis für die Erweiterungen. Von dieser Klasse erben die spezialisierten Metaklassen, die um Konzepte für den Umgang mit Imperfektion erweitert wurden. Grundlage für die speziellen Klassen für imperfekte Spalten bildet eine allgemeine Metaklasse *ImperfectColumn*, von der dann die Klassen *FuzzyColumn*,

ImpreciseColumn und *UncertainColumn* erben, die die jeweilige Art von Imperfektion repräsentieren können.

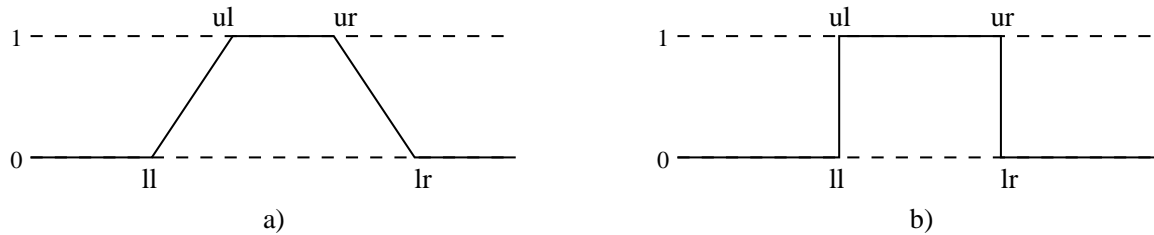


Abbildung 11: TrapezoidMembershipFunction a) unscharf (fuzzy) — b) scharf

Die jeweils benötigten Zugehörigkeits- und Wahrscheinlichkeitsfunktionen werden in einer eigenen Klassenhierarchie modelliert. Abbildung 10b zeigt, wie vom grundlegenden *ModelElement* eine neue Metaklasse *Function* erbt, die dann die Oberklasse für die speziell benötigten Funktionen, wie *MembershipFunction* oder *ProbabilityFunction* ist. Diese Klassen stellen erst einmal das allgemeine Konzept der Zugehörigkeits- beziehungsweise Wahrscheinlichkeitsfunktionen dar. Um das Modell zu verdeutlichen, wurde dann noch eine konkrete Zugehörigkeitsfunktion, nämlich die *TrapezoidMembershipFunction* modelliert. Abbildung 11 zeigt die speziellen Eigenschaften dieser Funktion: vier Eckpunkte beschreiben die Werte ab wann der Funktionswert nicht mehr 0 ist, in welchem Bereich er 1 ist und ab wann er wieder 0 ist. Die *TrapezoidMembershipFunction* ist zur Modellierung von Zugehörigkeitsfunktionen für klassische Mengen (scharfe Abgrenzung) und für Fuzzy-Mengen geeignet.

3.3.3 Wertung der vorgestellten Erweiterungen

In Abschnitt 2.2 wurden die verwendeten Erweiterungstechniken vorgestellt und festgestellt, dass die objektorientierten Techniken nicht geeignet sind, um direkt von einem CWM-konformen Tools verstanden zu werden. Aber auch der Einsatz von TaggedValues und Stereotypes für die Beschreibung von Zugehörigkeitsfunktionen ist nicht ideal [8]. Es besteht die Hoffnung, dass es unter Einsatz der neuen UML2 Erweiterungstechniken möglich ist, eine flexible, das heißt auch automatisierbare, Erweiterung zu modellieren, die zwar noch nicht von den aktuellen Tools verstanden wird, aber die vermutlich mit Tools, die irgendwann einmal „CWM 2“ verstehen, interpretiert wird.

In [10] wird mit dem Kontextmodell nur eine Modellierung der Imperfektion für relationale Datenbanken beschrieben, in [8] wurde somit auch nur das *Relational*-Paket des CWM erweitert. Ich werde deshalb in den weiteren Kapiteln, wenn ich ein Profil zur Beschreibung imperfekter Informationen für das Common Warehouse Metamodel entwickle, untersuchen, welche Möglichkeiten es gibt, dieses Profil allgemeiner zu halten. Mit dem Ziel, es nicht auf ein bestimmtes Datenmodell zu beschränken, sondern eine grundlegende Möglichkeit zu bieten, bei der Beschreibung von Metadaten zu verdeutlichen, dass ein Datum mit Imperfektion behaftet sein kann, unabhängig von dem Datenmodell mit dem es gespeichert wird.

4 Weitergehende CWM-Erweiterung zur Beschreibung von Imperfektion

Wie schon in Abschnitt 3.3.3 erwähnt, ist die von Alexander Haag vorgestellte Erweiterung des Common Warehouse Metamodel [8] sehr restriktiv auf relationale Datenbasisschema beschränkt. Ziel dieses Abschnittes ist es, weitere Möglichkeiten einer Erweiterung aufzuzeigen, die vor allem auch generischer einsetzbar sind.

Da meiner Meinung nach sich in zukünftigen Versionen der Erweiterungsmechanismus des Common Warehouse Metamodel wieder dem der UML Version 2.0 angleichen wird (siehe Abschnitt 2.3.4), werde ich die weitergehendere Erweiterung schon jetzt mit dieser neuen Technik modellieren.

Aus Abschnitt 3.1 ist bekannt, dass es dreierlei Arten von Imperfektion gibt. Deshalb werde ich das Problem, eine generische Erweiterung zu modellieren, aufteilen und zuerst in Abschnitt 4.2.1 untersuchen, welche Metaklassen des CWM für eine generische Beschreibung von unsicheren Informationen erweitert werden müssen, bevor in Abschnitt 4.2.2 dann diese Erweiterung vorgestellt wird. Daraufhin wird in Abschnitt 4.3.1 das CWM auf Metaklassen untersucht, die einer Erweiterung zur Darstellung von unscharfen Informationen dienen. Das modellierte Profil wird dann anschließend in Abschnitt 4.3.2 vorgestellt. Abschließend wird noch eine Modellierungsidee für eine Erweiterung zur Beschreibung ungenauer Informationen präsentiert, aber nicht weiter vertiefend untersucht (Abschnitt 4.4).

4.1 Vorüberlegungen

Bevor nun die genauen Metaklassen gesucht werden, die erweitert werden müssen, wird nun zuerst in diesem Abschnitt erläutert, warum man sich bei der Suche nach geeigneten Metaklassen auf das *Core-Package* beschränken kann und auch muss.

Die in [8] vorgestellte Erweiterung, erweitert das *Relational*-Paket um Imperfektion. Dies hat zur Folge, dass damit nur beschrieben werden kann, wie Imperfektion in relationalen Datenbanken dargestellt werden kann. In dieser Studienarbeit wird deshalb versucht, einen Ansatzpunkt im CWM zu bestimmen, der als Basispunkt einer Erweiterung dient und generischer ist als die *Column*-Klasse aus dem *Relational*-Paket.

Wie aus Abschnitt 2.1 bekannt ist, ist das CWM in mehrere Schichten eingeteilt. Eine Schicht besteht wiederum aus mehreren Paketen. Auf der *Resource*-Schicht liegen fünf Pakete zur Beschreibung verschiedener Datenmodelle, unter anderem auch das *Relational*-Paket zur Beschreibung relationaler Datenbasisschemata. Würde eine Erweiterung des CWM um Imperfektion auf einer Metaklasse einer dieser Pakete aufbauen, dann stände sie auch nur bei der Beschreibung dieses speziellen Datenmodells zur Verfügung. Deshalb darf eine allgemeine Erweiterung des CWM um Imperfektion nicht eines dieser Pakete als Basis haben, sondern muss auf einer oder mehreren Klassen einer niederen Schicht aufbauen.

Auch aus einem anderen Blickwinkel betrachtet, bietet sich eine der Klassen aus der *Resource*-Schicht nicht als Basis für eine Erweiterung an. Die *Resource*-Schicht stellt Möglichkeiten zur Beschreibung von Datenmodellen. Die Eigenschaft einer Information imperfekt zu sein, ist aber keine spezielle Eigenschaft, die an ein Datenmodell gebunden ist. Die Information ist imperfekt unabhängig davon mit welchem Datenmodell sie gespeichert wird. Es ist also wünschenswert, dass eine Erweiterung des CWM um Imperfektion, eine allgemeine Beschreibungsmöglichkeit für Imperfektion bietet. Da diese Beschreibung vom Datenmodell unabhängig sein soll, kann als Basisklasse auch keine Metaklasse aus der *Resource*-Schicht verwendet werden.

Wie aus Abbildung 12 ersichtlich ist, gibt es zu allen grundlegenden Klassen eines Paketes aus der *Resource*-Schicht in den restlichen Paketen der Schicht eine korrespondierende Klasse. So entspricht

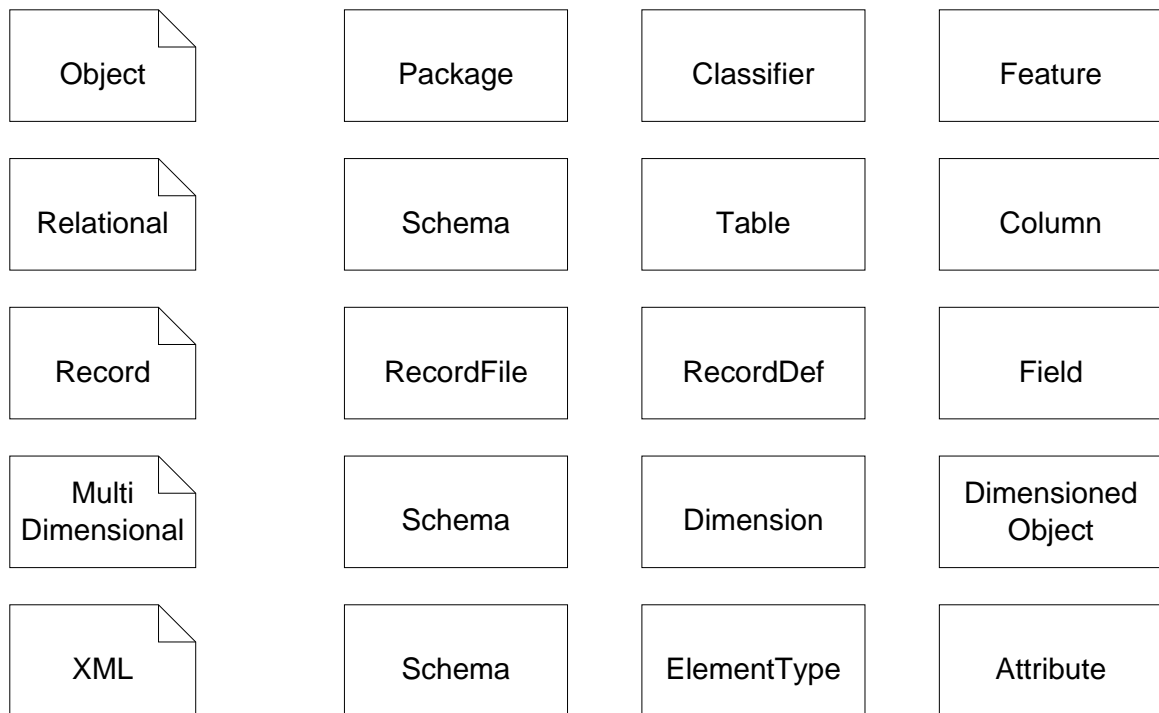


Abbildung 12: Zusammenhang der Pakete auf der Resource-Schicht

eine *Relational::Column* zum Beispiel einem *XML::Attribute*. Erreicht wird diese „Classifier equivalence“ [20, Seite 151] dadurch, dass allen eine gemeinsame Oberklasse zugrunde liegt.

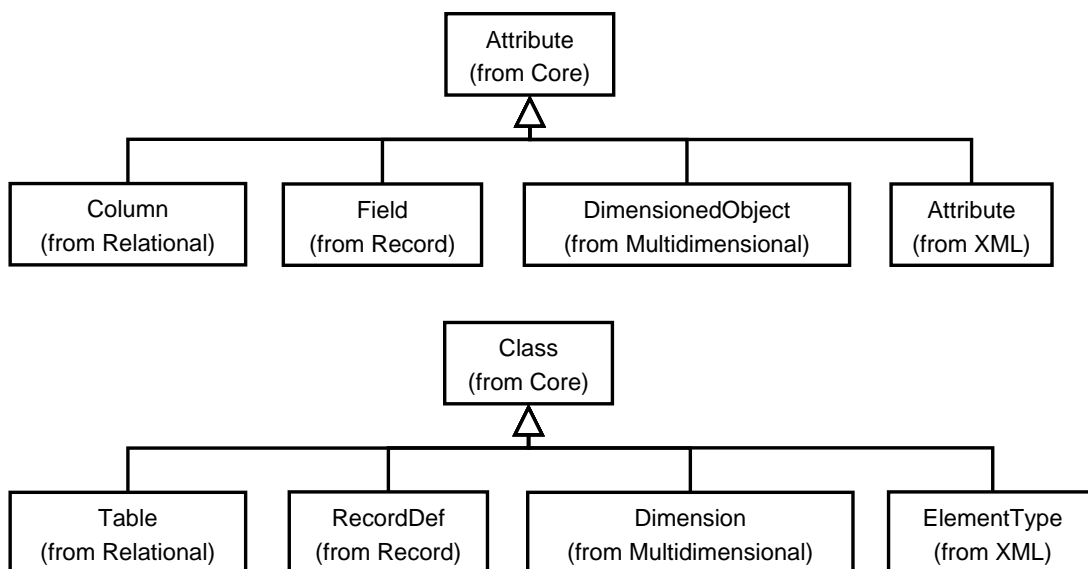


Abbildung 13: Spezialisierungen von Core::Attribute und Core::Class

Untersucht man das CWM nun daraufhin, von welchen Metaklassen die relevanten Klassen der *Resource*-Schicht abhängen, erkennt man, dass fast alle direkten Elternklassen im *Core*-Paket liegen. Beispielhaft ist in Abbildung 13 gezeigt, welches die Elternklassen von den Metaklassen *Relational::Column* und *Relational::Table* und deren äquivalenten Klassen auf der *Resource*-Schicht sind. Dabei sind die Klassen *Attribute* und *Class* aus dem *Core*-Paket als spezialisierteste Elternklassen zu erkennen.

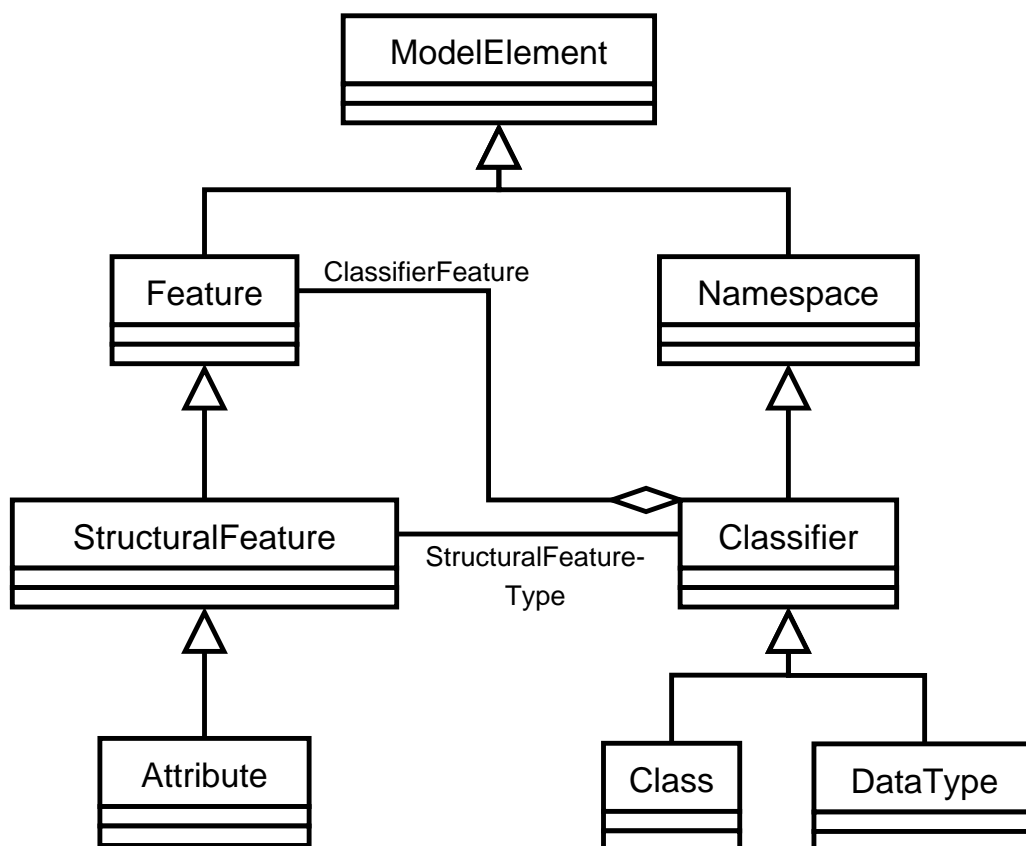


Abbildung 14: Ausschnitt CWM::Core-Paket

Deshalb kann man sich bei der Suche nach einem Erweiterungsansatz auf das *Core*-Paket des Common Warehouse Metamodel beschränken.

4.2 Erweiterung des CWM zur Beschreibung unsicherer Informationen

4.2.1 Ansatzpunkte einer Erweiterung

Die Beschreibung unsicherer Informationen geschieht durch eine Beschreibung der darzustellenden Information ergänzt um einen Wahrscheinlichkeitsgrad, der angibt für wie wahrscheinlich man den Wahrheitsgehalt der Information hält (Abschnitt 3.1.1). Eine unsichere Staumeldung besteht somit zum Beispiel aus den Informationen wo und wie lang ein Stau ist, zusammen mit einer Angabe, die den Wahrheitsgehalt der Meldung beschreibt. Wenn die Meldung von der Polizei bekannt gegeben wurde, so wird dieser Wert höher ausfallen als bei einem erstmalig auftretenden freiwilligen Staumelder.

Das Kernstück des Core-Paketes bietet Strukturierungsmöglichkeiten zur Beschreibung von Objekten aus einer Realwelt, zur Darstellung von Informationen. Abbildung 14 verdeutlicht wie im CWM strukturierte Objekte durch *Classifier* repräsentiert werden. Ein *Classifier* besteht aus mehreren *ClassifierFeatures*, die die individuellen Eigenschaften von Bestandteilen einer Sache beschreiben. Dies, wieder am Beispiel eines Staus verdeutlicht, bedeutet, dass ein Stau durch einen *Classifier* modelliert wird und vermutlich aus den Features „Beginn“, „Ende“ und „Staulänge“ bestehen würde.

Es gibt nun zwei Anwendungsszenarien, welche Informationen bei einer strukturierten und zusammengesetzten Sache unsicher sein können. Als erstes kann man der gesamten Sache mißtrauen. Wieder am Beispiel betrachtet, wird die gesamte Staumeldung als gesamtes als unsicher betrachtet. Zum zweiten

muss man vielleicht auch nur ein einzelnes *ClassifierFeature* als unsicher betrachten. Beispielsweise wird nur die gemeldete Staulänge als unsicher angesehen, während man der Meldung, dass überhaupt ein Stau vorliegt, voll vertraut.

Will man nun modellieren, dass die Information über den Stau mit Unsicherheit behaftet ist, müsste man wie oben beschrieben, der Information über den Stau noch einen Wahrscheinlichkeitsgrad zuweisen. Dies könnte zum Beispiel einfach durch ein weiteres Feature, das dann den Wahrscheinlichkeitsgrad beschreibt, geschehen. Auch die Modellierung, dass nur das Staulängenattribut unsicher ist, könnte über ein weiteres Feature der Staumeldung geschehen, solange deutlich wird, dass dieses weitere Feature den Wahrscheinlichkeitsgrad des Staulängenattributes beschreibt.

Welche Metaklassen des CWM müssen nun um einen Stereotypen zur Beschreibung unsicherer Informationen erweitert werden? Zum einen die Klasse *Core::Class*, damit man einer zusammengesetzten Struktur im gesamten einen Unsicherheitswert zuordnen kann. Desweiteren muss man auch ein einzelnes *Core::StructuralFeature* als unsicher bezeichnen können. Nach Abbildung 14 wird einem *StructuralFeature* über die Assoziation *StructuralFeatureType* ein *Classifier* als Typ zugeordnet, was dann wiederum eine zusammengesetzte Struktur (*Class*) oder ein einfacher Datentyp sein kann. Wenn man nun den *Core::Classifier* erweitert, um unsichere Informationen kennzeichnen zu können, so wäre diese Erweiterung in beiden Anwendungsszenarien einsetzbar. Eine *Class* kann als direkte Unterklasse von *Classifier* als unsicher beschrieben werden. Ebenso ist es möglich ein einzelnes Feature einer *Class* als unsicher zu kennzeichnen, indem über die Assoziation *StructuralFeatureType* des Features ein unsicherer *Classifier* ausgewählt wird. Es bietet sich also an, die Metaklasse *Core::Classifier* zur Beschreibung von unsicheren Informationen zu erweitern.

Die bisher getätigten Überlegungen werden nun an einem Beispiel zusammenfassend dargelegt: Die Informationen über Staus seien in einer relationalen Datenbank darzustellen. Dazu gibt es eine Tabelle „Stau“, die vier Spalten besitzt „Autobahnnummer“, „Beginn“, „Ende“ und „Länge“. Nun möchte man zusätzlich ausdrücken, dass man zum einen die gesamte Staumeldung als unsicher betrachtet und des weiteren auch schon der gemeldeten Staulänge alleine nicht vertraut. Im CWM wird eine relationale Tabelle mit einer *Relational::Table* modelliert, die eine Unterklasse von *Core::Classifier* ist. Ist die Classifier-Klasse für die Beschreibung von Unsicherheit erweitert, so kann dann auch die gesamte Tabellenzeile als unsicher gekennzeichnet werden. Die Spalten der Stautabelle werden als *Relational::Columns* modelliert. Den *Relational::Columns* wird über *StructuralFeatureType* ein Datentyp zugewiesen. Ist dieser Datentyp, eine Unterklasse von *Classifier*, unsicher, so wird auch deutlich, dass in der Tabellenspalte unsichere Informationen stehen.

4.2.2 Modellierung

In diesem Abschnitt wird nun begonnen ein Profil zu entwickeln, so dass das Common Warehouse Metamodel um Möglichkeiten zur Beschreibung von imperfekten Informationen erweitert wird. Zuerst werden in diesem Abschnitt Stereotypen zur Kennzeichnung von unsicheren Informationen vorgestellt.

Aus dem vorherigen Abschnitt ist die Idee der Modellierung unsicherer Informationen ja schon bekannt. Eine unsichere *Class* wird um ein weiteres Feature erweitert, das den Wahrscheinlichkeitsgrad der gesamten *Class* beschreibt. Wird *Core::DataType* als unsicher markiert, so muss es in der Klasse, die das Attribut, zu dem der Datentyp gehört, besitzt, eben auch ein weiteres Feature geben, das nun den Wahrscheinlichkeitsgrad des Attributes beschreibt.

Abbildung 15 zeigt nun den Ausschnitt aus dem Profil *Imperfection*, der die Erweiterung zur Beschreibung unsicherer Informationen enthält. Wie nicht anders zu erwarten, wird die aus dem vorherigen Abschnitt herausgedeutete Klasse — *Core::Classifier* — als `<<metaclass>>` gekennzeichnet ins Profil importiert. (Die zu erweiternden Klassen des Metamodell werden im Profil mit `<<metaclass>>`

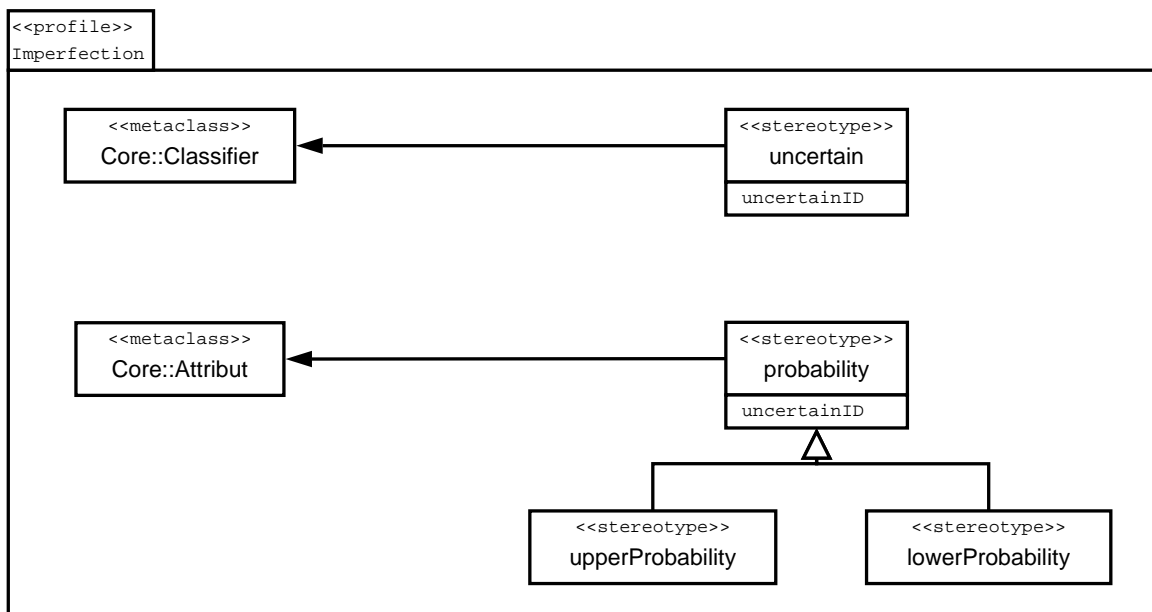


Abbildung 15: Auszug aus dem Profil Imperfection: Stereotyp <<uncertain>>

gekennzeichnet.) Der Stereotyp <<uncertain>> kann nun eingesetzt werden, um diese Klasse als unsicher zu markieren. Da der Erweiterungspfeil nicht mit {required} ausgezeichnet ist, steht es dem Anwender bei der Verwendung des Profils frei, ob er die Metaklasse *Classifier* mit dem Stereotyp kennzeichnet oder nicht, also ob er unsichere Informationen beschreiben will oder nicht.

Oben wurde beschrieben, dass, falls eine *Class* oder auch ein *DataType* mit <<uncertain>> gekennzeichnet wird, es ein weiteres Feature geben muss, das den Wahrscheinlichkeitsgrad der Unsicherheit aufnimmt. Genau dazu dient nun der Stereotyp <<probability>>. Da ein *Core::Attribute* auch nur ein Feature ist, das aber mit einem Initialwert belegt werden kann [20, Seite 87], erweitert der Stereotyp <<probability>> nicht die Metaklasse *StructuralFeature*, sondern *Attribute*, um eventuell auf die explizite Angabe eines Wahrscheinlichkeitsgrades verzichten zu können und das Feature mit einem Standardwert zu belegen (siehe Abbildung 14).

Wie erkennt man nun aber, falls mehrere Features als unsicher gekennzeichnet sind, welches Attribut nun den Wahrscheinlichkeitsgrad von welchem Feature beschreibt? Dazu dient das zusätzliche Attribut *uncertainID* in den Stereotypen. Es muss für jede Metaklasse, die mit dem Stereotyp <<uncertain>> markiert ist, auf einen eindeutigen Wert gesetzt werden. Desweiteren muss es für alle Klassen, die als unsicher gekennzeichnet sind ein Attribut geben, welches mit dem Stereotyp <<probability>> ausgezeichnet ist und den gleichen Wert für *uncertainID* zugewiesen bekommen hat, so dass der Zusammenhang zwischen den beiden Metaklassen erkennbar wird.

Abbildung 16 zeigt nun, wie die Stereotypen eingesetzt werden können, um die beispielhafte Stautabelle aus dem vorherigen Abschnitt zu modellieren und deutlich machen zu können, dass die Informationen in der gesamten Tabelle als unsicher angesehen werden. Wieder wird die schon bekannte Stautabelle beschrieben. Diesmal jedoch wird die Table mit <<uncertain>> als unsicher gekennzeichnet. Da es für alle mit <<uncertain>> markierten Klasse auch eine Unterklasse von *Core::Attribute* geben muss, die den Wahrscheinlichkeitsgrad aufnimmt und mit dem Stereotyp <<probability>> ausgezeichnet ist, wurde der Stautabelle eine weitere Spalte zur Aufnahme dieses Wahrscheinlichkeitswertes hinzugefügt.

Da zur Zeit im OVID-Projekt im Rahmen einer Diplomarbeit erste Ansätze untersucht werden, unsichere Informationen durch ein Wahrscheinlichkeitsintervall anstatt eines einzelnen Wahrscheinlichkeitswertes zu beschreiben, wurde das Profil auch gleich um Stereotypen zur Beschreibung von Wahr-

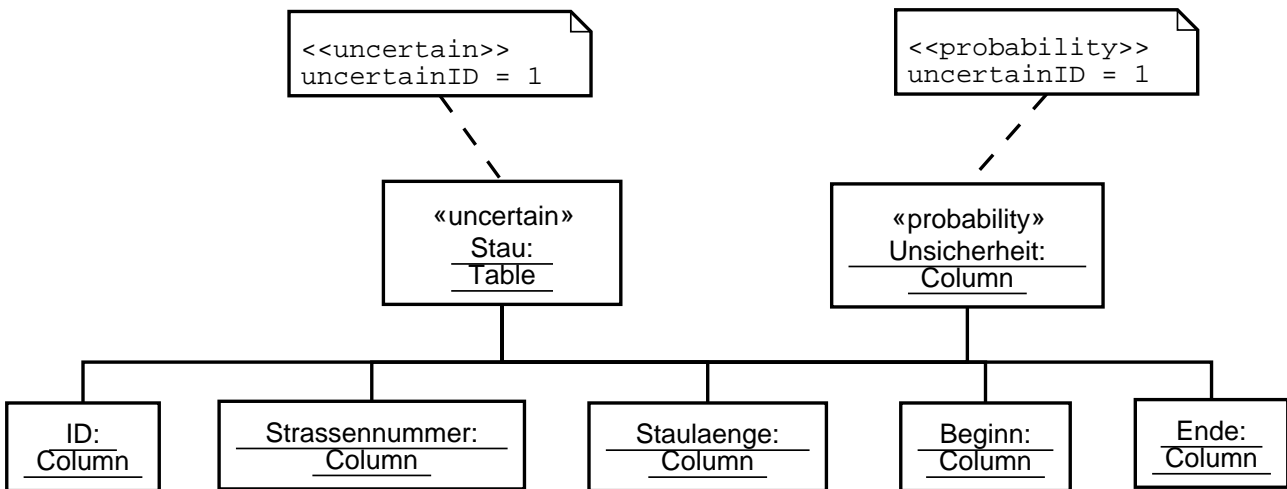


Abbildung 16: Beispiel: unsichere Stautabelle

scheinlichkeitsintervallen erweitert. Anstatt des bisherigen Stereotypen `<<probability>>`, müssen dann die neuen Stereotypen `<<lowerProbability>>` und `<<upperProbability>>` verwendet werden. Diese Stereotypen kennzeichnen ein Attribut nun, ob es die untere Schranke (`<<lowerProbability>>`) oder die obere Schranke (`<<upperProbability>>`) eines Wahrscheinlichkeitsintervalls aufnimmt. Wieder wird der Zusammenhang zwischen der unsicheren *Class* oder *Attribute* und dem das Wahrscheinlichkeitsintervall beschreibenden *Attributes* über das zusätzliche Attribut *uncertainID* in den Stereotypen hergestellt.

4.3 Erweiterung des CWM zur Beschreibung unscharfer Informationen

4.3.1 Ansatzpunkte einer Erweiterung

Zu Beginn der Suche nach einer geeigneten Metaklasse, die zur Beschreibung unscharfer Informationen erweitert werden kann, wird wieder das Beispiel der Stautabelle betrachtet. Sie soll diesmal jedoch nicht als unsichere Information aufgefasst werden. Auch unscharfe Informationen stecken in dieser Tabelle. Will man nämlich die „Staulänge“ in Klassen („kurz“, „mittellang“ und „lang“) einteilen, liegen unscharfe Informationen vor. Zu jeder Staulänge müsste man nun die Zugehörigkeitsgrade zu den einzelnen Klassen angeben können. Dazu bräuchte man eine linguistische Variable, die für das Attribut Staulänge die Terme „kurz“, „mittellang“ und „langer Stau“ beschreibt.

Aus den vorherigen Abschnitten ist bekannt, dass ein generischer Ansatzpunkt zur Erweiterung eines einzelnen Features die Metaklasse *Attribute* aus dem *Core*-Paket ist. Dies entspricht auch dem Vorgehen in [8], dort wird nämlich die Metaklasse *Relational::Column* zur Beschreibung unscharfer Informationen erweitert. Und *Relational::Column* ist, wie inzwischen bekannt sein dürfte, eine Unterklasse von *Core::Attribute*, somit stellt *Core::Attribute* einen generischeren Ansatzpunkt einer Erweiterung dar. Es bietet sich also unter Umständen an, die Klasse *Core::Attribute* zu erweitern.

Die Tatsache, dass die Information, die ein Attribut aufnimmt, unscharf ist, lässt sich auch ausdrücken, indem für das Attribut ein Datentyp verwendet wird, der mit Imperfektion umgehen kann. Der Ausschnitt aus dem CWM::Core-Paket in Abbildung 14 zeigt, dass jedem Attribut über die Assoziation *StructuralFeatureType* ein *Classifier* zugeordnet wird. (Die Klasse *Attribute* erbt die Assoziation mit der *Classifier*-Klasse von ihrer Oberklasse *StructuralFeature*.) Insbesondere kann so jedem Attribut ein *Data Type*, was eine Unterklasse von *Classifier* ist, zugeordnet werden. Die Metaklasse *Core::DataType*

besitzt keinerlei eigene Attribute und Funktionen. Somit stellt sie einen Platzhalter für weiter zu spezifizierende Unterklassen dar, die es dann ermöglichen für spezielle Datenmodelle deren Datentypen zu beschreiben. Zum Beispiel bietet die Unterklasse *SQLDataType* aus dem *Relational*-Paket ein Attribut *typeName* zur Auswahl eines standardisierten Datentyps aus den relationalen Datenbanksystemen. Könnte man nun einen Datentyp als unscharf kennzeichnen, so wäre dann auch ersichtlich, dass das zugehörige Attribut unscharfe Informationen repräsentiert.

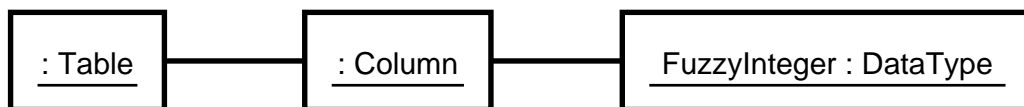


Abbildung 17: Idee der Modellierung eines unscharfen Datentyps

Auch jetzt ist es mit dem CWM schon eingeschränkt möglich, einen Datentyp zu kennzeichnen, dass er zur Beschreibung von unsicheren Informationen dient. Abbildung 17 zeigt wie diese Kennzeichnung aussehen könnte. Dazu muss man bei der Beschreibung eines Datenbasisschemas für das Objekt eines unscharfen Datentypes einfach einen aussagekräftigen Namen wählen, der auf die Unschärfe hinweist. Einen Datentyp, der zum Beispiel eine unscharfe Menge über den Integers beschreibt, würde man dann als *FuzzyInteger* bezeichnen. Solange im CWM aber noch kein präziser spezifizierter Datentyp zur Beschreibung von Imperfektion existiert, ist es so leider nicht möglich weitere Informationen über die Eigenschaften dieses *FuzzyIntegers* zu beschreiben. Es ist also nicht möglich anzugeben, ob eine linguistische Variable hinter diesem Datentyp steht und mit welchen Termen und Zugehörigkeitsfunktionen diese genauer spezifiziert werden kann.

Hier kann nun eine Erweiterung des CWM ansetzen. Die Metaklasse *Core::DataType* wird um einen Stereotyp <<fuzzy>> erweitert, der dann eine Beschreibung von linguistischen Variablen zur Darstellung von unscharfen Informationen ermöglicht. Dieser Ansatz erweitert alle vorhandenen Datentypen, um eine Beschreibungsmöglichkeit von unscharfen Informationen und führt somit neue Datentypen ein.

Beim Ansatzpunkt *Core::Attribute* wurde nach einer möglichst allgemeinen Oberklasse gesucht, die als Grundlage für eine Erweiterung dient. Warum wird dann nicht auch jetzt bei diesem Ansatz die Metaklasse *Core::Classifier* erweitert, obwohl sie doch generischer als *Core::DataType* ist?

Prinzipiell wäre vermutlich auch *Classifier* als Erweiterungsgrundlage möglich und damit untersuchenswert. Da aber für den Einsatz im Rahmen des OVID-Projektes, eine Erweiterung der Metaklasse *DataType* ausreicht, wurde auf eine weitergehende Untersuchung dieser Möglichkeit verzichtet. Falls es in der Zukunft nötig wäre, diesen Stereotype zur Beschreibung von unscharfen Informationen auch auf einen *Classifier* anzuwenden, müsste eben zuerst einführend analysiert werden, ob und mit welchem Aufwand diese Erweiterung einfach auf *Classifier* übertragen werden könnte.

Es stehen also zwei Metaklassen aus dem *Core-Package* heraus, die sich für eine Erweiterung zur Beschreibung von unscharfen Informationen anbieten: *Core::Attribute* und *Core::DataType*.

Bewertung und Auswahl Beide Metaklassen — *Core::Attribute* und *Core::DataType* — scheinen sich als Basisklassen für die Modellierung des Stereotyps <<fuzzy>> in einem Profil für die Beschreibung von imperfekten Informationen anzubieten. Die Variante über den *DataType* entspricht eher dem erwarteten Vorgehen zur Beschreibung der Eigenschaften eines Attributwertes. Über den Datentyp wird beschrieben, dass der Attributwert zum Beispiel vom Typ Integer ist. Ebenso betrachtet man nun den um Imperfektion erweiterten Datentyp als einen besonderen Datentyp, der eingesetzt wird wie andere Datentypen zur Beschreibung der Eigenschaften eines Attributwertes. Die zweite Variante,

eine Erweiterung der Metaklasse *Attribute*, trennt die Beschreibung der Imperfektion von der Definition der weiteren Eigenschaften eines Attributes, ordnet aber im Gegensatz dazu die Information über die Unschärfe direkt dem Attribut zu.

Die Aussagekraft beider Varianten ist gleich. Ich werde mich deshalb in dieser Studienarbeit auf eine Basisklasse beschränken. Im weiteren Verlauf dieser Arbeit werde ich nun die Metaklasse *Core::DataType* als Basisklasse des Stereotyps <<fuzzy>> verwenden. Im folgenden Abschnitt wird diese Metaklasse jetzt um Merkmale zur Beschreibung von unscharfen Informationen erweitert.

4.3.2 Erste Modellierungsmöglichkeit

In diesem Abschnitt wird nun das Profil *Imperfection* um Möglichkeiten zur Beschreibung unscharfer Informationen ergänzt. Dazu wird die Metaklasse *Core::DataType* um den Stereotyp <<fuzzy>> erweitert.

Zur Beschreibung von imperfekten Informationen wird auf die in Abschnitt 3.1.2 vorgestellten linguistischen Variablen zurückgegriffen. Der Stereotyp <<fuzzy>> soll die Metaklasse *DataType* um Möglichkeiten zur Definition von linguistischen Variablen erweitern. So wird es dem Anwender möglich bei der Modellierung nicht nur anzugeben, von welchem Datentyp ein Attribut ist, sondern er kann auch eine linguistische Variable für diesen Datentyp definieren.

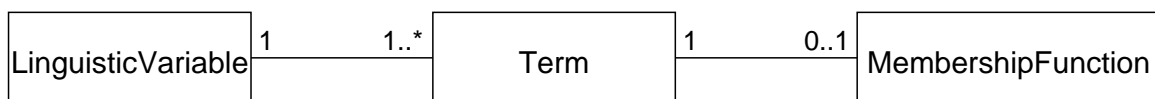


Abbildung 18: Linguistische Variable

Eine linguistische Variable besteht aus mehreren Termen. Ein Term beschreibt verbal eine Fuzzy-Menge, die formal mit jeweils einer charakteristischen Funktion spezifiziert wird (siehe Abbildung 18). Die charakteristische Funktion einer Fuzzy-Menge bestimmt zu jedem Element einer Grundmenge die Zugehörigkeit zu einem Term.

Die Kardinalität 0..1 an der Assoziation zwischen *Term* und *MembershipFunction* lässt die Möglichkeit offen, auch noch keine Funktion anzugeben. So ist man auch für Anwendungsszenarien gerüstet, in denen die Zugehörigkeitsfunktionen bei der Definition der Terme noch unbekannt sind. Zur Modellierung des Stereotyps <<fuzzy>> wird nun diese Struktur einer linguistischen Variablen verwendet. Der Stereotyp ergänzt ein *DataType* um eine Menge von Termen. Einem Term wird ein Name und eine Zugehörigkeitsfunktion zugeordnet. Zur Beschreibung der Zugehörigkeitsfunktion wird auf das *Expression*-Paket des CWM zurückgegriffen [20, Seite 99ff]. Im folgenden Abschnitt 4.3.3 wird eine weitere Modellierungsmöglichkeit vorgestellt, die komplett ohne die Angabe der Zugehörigkeitsfunktionen auskommt.

Das *Expression*-Paket erlaubt auf zweierlei Arten die Beschreibung von Ausdrücken und somit auch von Funktionen. Die erste Beschreibungsvariante, die sogenannte *black box expression*, ist eine einfache textuelle, an eine Programmiersprache angelehnte Formulierung des Ausdruckes, die um einen Hinweis ergänzt werden kann, auf welcher (Programmier-)sprache die textuelle Beschreibung basiert. Die zweite Variante, die *white box expression*, erlaubt die komplette Beschreibung des Syntaxbaumes des Ausdruckes. Dazu muss der Ausdruck zuerst in eine Folge von möglicherweise vernesteten Funktionsaufrufen überführt werden, um dann anschließend als Syntaxbaum mit den Klassen aus dem *Expression*-Paket dargestellt zu werden. Diese Darstellungsform erlaubt die Beschreibung von Ausdrücken unabhängig von irgendeiner Programmiersprache.

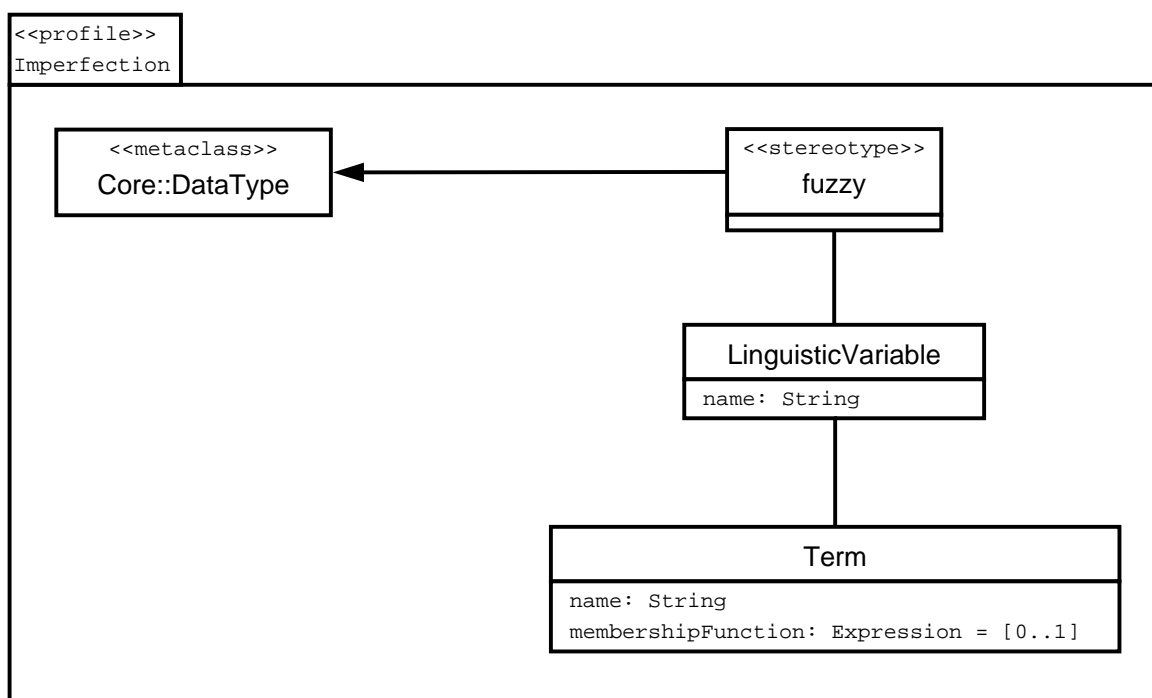


Abbildung 19: Auszug aus dem Profil Imperfection: Stereotyp <<fuzzy>>

Je nach Einsatzzweck, ob man eher eine einfache Lösung oder einen hohen Grad an Kompatibilität erreichen will, kann man sich nun für eine *black box* oder *white box expression* entscheiden.

Abbildung 19 zeigt nun den gesamten Stereotypen. Die Metaklasse *Core::DataType* wird um den Stereotyp <<fuzzy>> erweitert. Da der Erweiterungspfeil nicht mit {required} markiert ist, steht es dem Anwender des Profils frei, einen Datentyp als fuzzy zu kennzeichnen oder nicht. Der Stereotyp <<fuzzy>> ergänzt nun einen Datentyp um das Konzept der linguistischen Variable. Wie oben beschrieben, kann zu einem Datentyp, der als fuzzy gekennzeichnet ist, eine Menge von Termen definiert werden, die wiederum durch einen Namen (**name**) und eine Zugehörigkeitsfunktion (**membershipFunction**) genauer spezifiziert werden. Da **Expression** die Art der Beschreibung — *black box* oder *white box expression* — nicht einschränkt, steht es dem Profilbenutzer offen, abhängig vom Einsatzzweck, die Zugehörigkeitsfunktion textuell oder als Syntaxbaum zu beschreiben.

Eine beispielhafte Anwendung des Profils zur Beschreibung der Stautabelle mit unscharfem Staulängenattribut zeigt Abbildung 20. Der Datentyp des Attributes *Staulaenge* ist mit <<fuzzy>> gekennzeichnet und wird um die linguistische Variable „Staulaenge“ erweitert, die aus den Termen „kurz“, „mittellang“ und „lang“ besteht. Die Zugehörigkeitsfunktionen werden der Einfachheit halber als TrapezoidMembershipFunctions (siehe Abschnitt 3.3.2) beschrieben. Abbildung 21 verdeutlicht den grafischen Verlauf der Zugehörigkeitsgrade der einzelnen Terme in Abhängigkeit der Staulänge.

4.3.3 Zweite Modellierungsmöglichkeit

Im vorherigen Abschnitt wurde schon angesprochen, dass es Anwendungsszenarien geben kann, in denen die Zugehörigkeitsfunktionen zu den Termen einer linguistischen Variable unbekannt sind. Jetzt wird eine Erweiterung des CWM vorgestellt, die einsetzbar ist, falls eben die Zugehörigkeitsfunktionen unbekannt sind, aber zu jedem Element trotzdem abgeschätzt werden kann, mit welchem Zugehörigkeitsgrad es zu den jeweiligen Termen gehört. Ein beispielhaftes Anwendungsszenario ist, dass eine gemeldete Staulänge nicht nach einer strengen Funktion den Termen „kurz“, „mittellang“, „langer

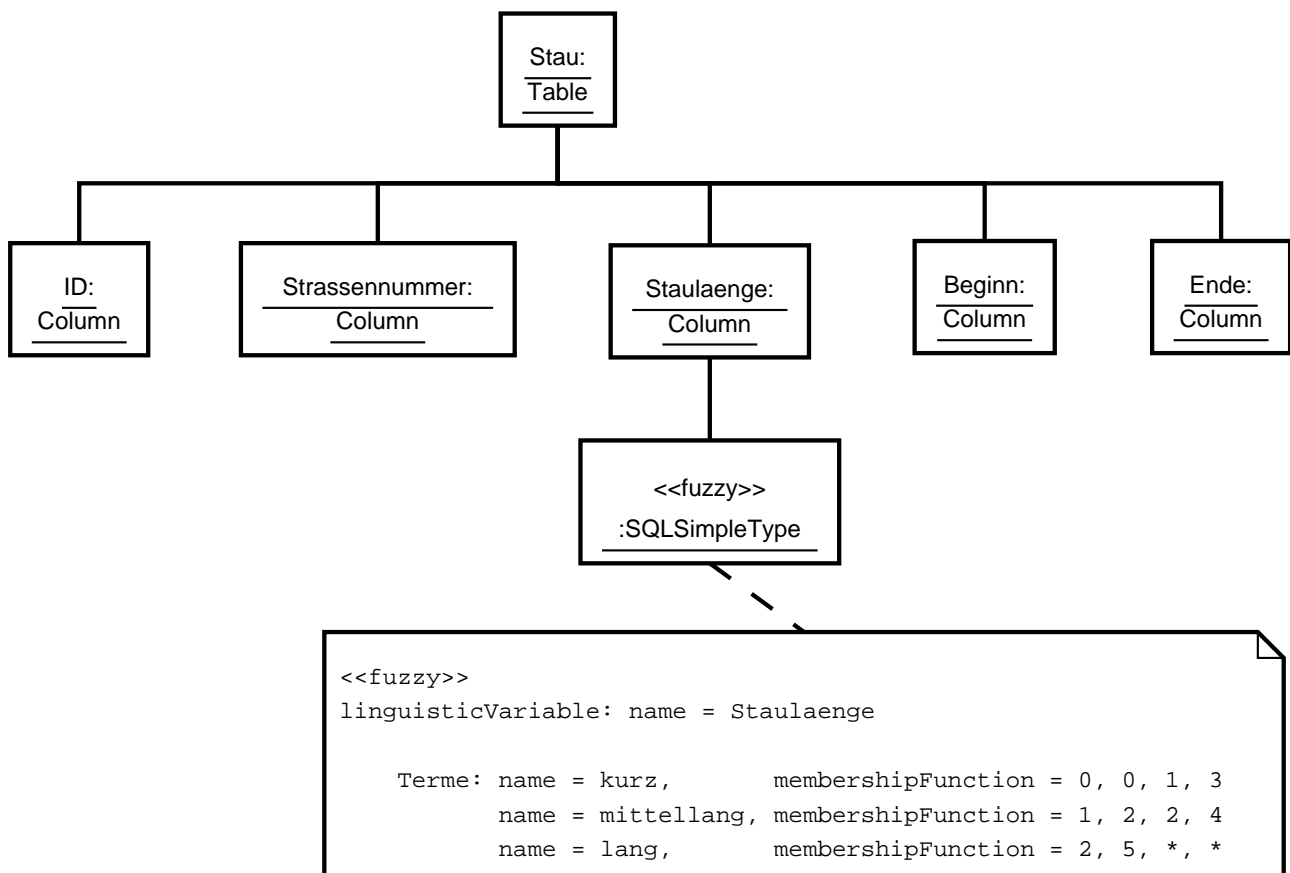


Abbildung 20: Beispiel: Stautabelle mit unscharfem Attribut

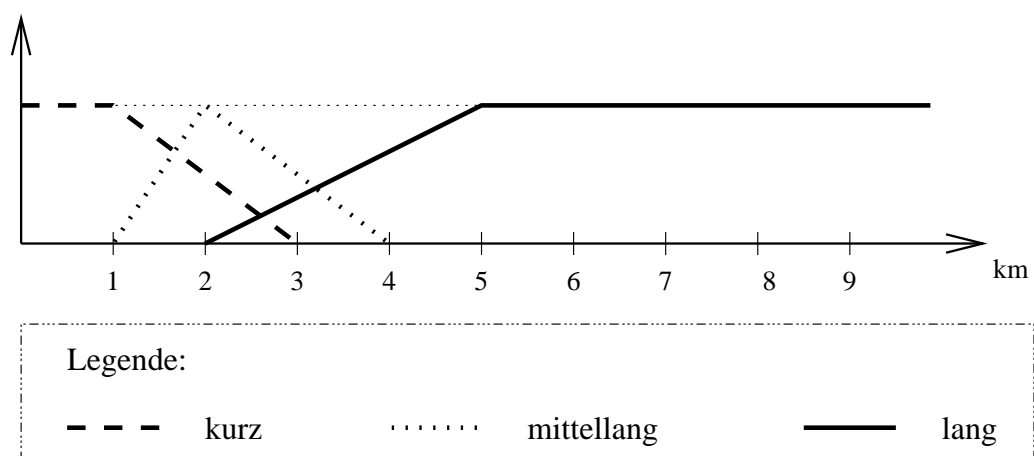


Abbildung 21: Zugehörigkeitsfunktionen der linguistischen Variable „Staulaenge“

Stau“ zugeordnet wird, sondern die Zugehörigkeitsgrade einfach dem gesunden Menschenverstand nach bestimmt werden.

Eine weitere Motivation für eine zweite Modellierungsmöglichkeit zur Beschreibung unscharfer Informationen ist die Möglichkeit, dass in einer Datenbank vermutlich gar nicht die Originalwerte gespeichert sind, sondern nur die Zugehörigkeitsgrade zu den einzelnen Termen. Dies bedeutet, es gibt zu jedem Term ein Attribut, das den Zugehörigkeitsgrad aufnimmt. Auch diesem Szenario muss eine Modellierung eines Profils zur Beschreibung unscharfer Informationen gerecht werden. Mit dem Profil muss der Zusammenhang zwischen den einzelnen Termen und einer linguistischen Variable formulierbar sein.

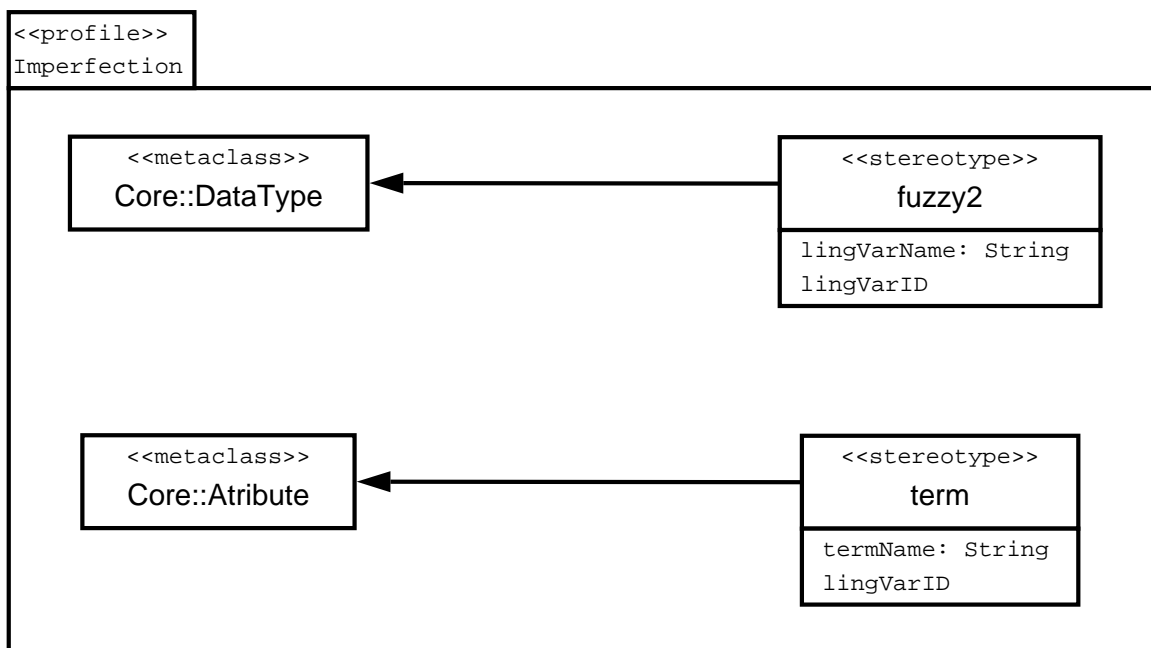


Abbildung 22: Auszug aus dem Profil Imperfection: Stereotyp <<fuzzy2>>

Abbildung 22 zeigt nun den Ausschnitt aus dem Imperfection-Profil, das den neuen Stereotyp enthält, der zur Beschreibung von linguistischen Variablen ohne Zugehörigkeitsfunktionen geeignet ist. Die Idee gleicht dem Ansatz zur Beschreibung von unsicheren Informationen (Abschnitt 4.2.2). Wird ein Datentyp als <<fuzzy2>> markiert, bedeutet es, dass es in der zugehörigen *Class* weitere Attribute geben muss, die die Zugehörigkeitsgrade zu den einzelnen Termen aufnehmen. Die Kenntlichmachung dieser Terme geschieht durch den weiteren Stereotyp <<term>>. Ein Attribut, das mit <<term>> gekennzeichnet ist, wird um ein Merkmal erweitert, das den Namen des Terms angibt (*termName*) und einen Verweis auf die linguistische Variable, zu der der Term gehört (*lingVarID*).

Wird nun ein Datentyp mit <<fuzzy2>> als unscharfe Information gekennzeichnet, so muss es so viele weitere Attribute geben, die mit <<term>> markiert sind, wie es Terme gibt, die die linguistische Variable zu dem Attribut beschreiben. Der Zusammenhang zwischen dem unscharfen Attribut und den Termen wird über die gleiche *lingVarID* deutlich gemacht. Die mit <<term>> markierten Attribute können nun die jeweiligen Zugehörigkeitsgrade aufnehmen.

Betrachtet man das Szenario, dass in der Datenbank keine Originalwerte gespeichert werden, sondern nur die Zugehörigkeitsgrade zu den einzelnen Termen, so besteht auch nicht die Möglichkeit diesem Attribut einen mit <<fuzzy2>> gekennzeichneten Datentyp zuzuweisen. Wie schon beschrieben, ermöglicht das Profil über den Stereotyp <<term>> die Attribute zu kennzeichnen, die die Zugehörigkeitsgrade zu dem Term aufnehmen. Falls ein Attribut die Originalwerte speichern soll und sein Datentyp somit mit <<fuzzy2>> markiert ist, konnte die Zuordnung der Terme zu einer linguistischen

Variable mit dem Attribut `lingVarID` hergestellt werden. In dem nun betrachteten Fall existiert aber kein Attribut für die Originalwerte, es gibt somit auch keinen mit `<<fuzzy2>>` ausgezeichneten Datentyp für diese linguistische Variable. Deshalb wird im Stereotyp `<<term>>` durch ein zusätzliches Attribut `lingVarName` die Zuordnung zu einer linguistischen Variable ermöglicht.

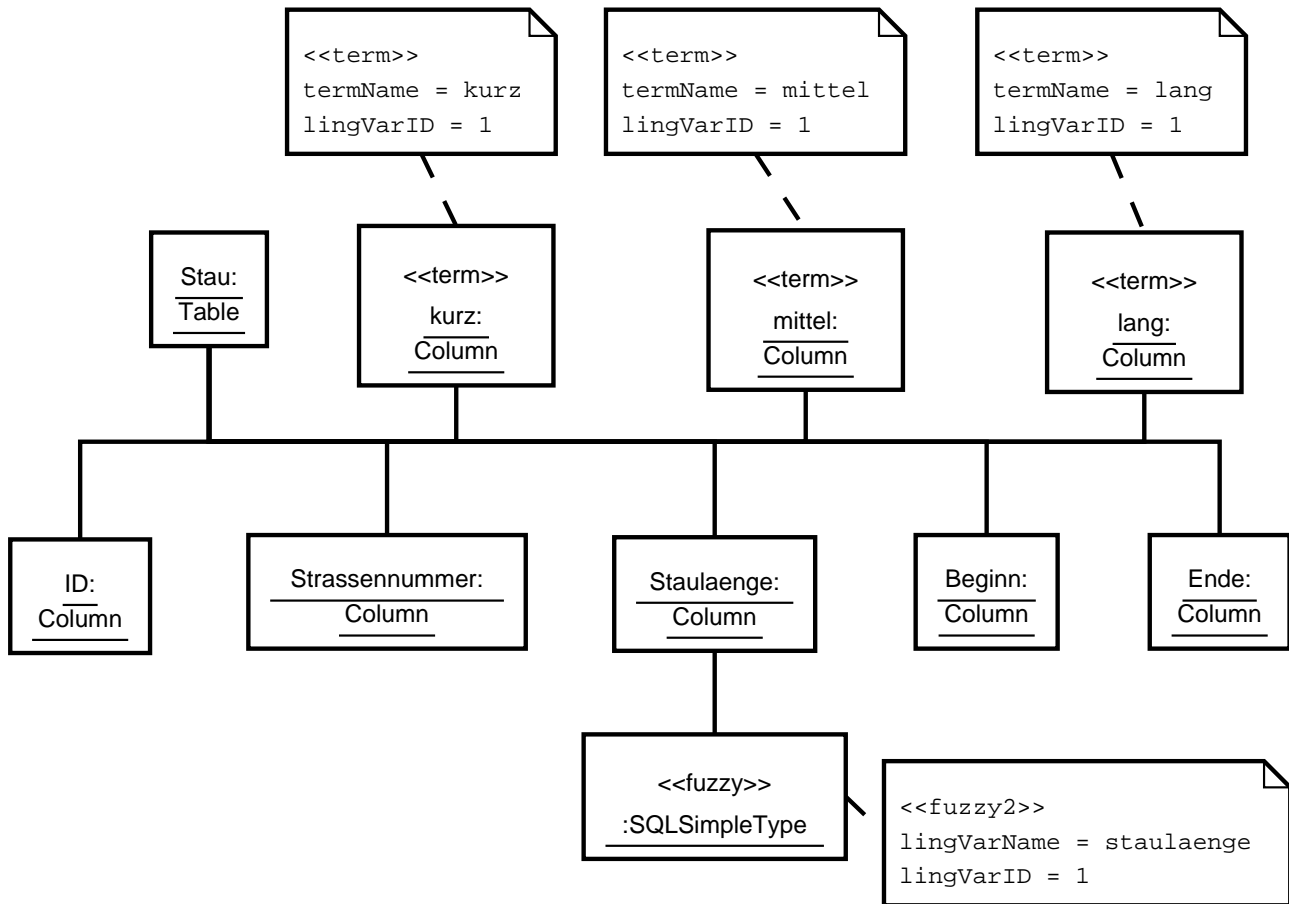


Abbildung 23: Beispiel: Stautabelle mit unscharfer Staulänge ohne Zugehörigkeitsfunktionen

Auch diese Stereotypen werden nun anhand eines Beispiels verdeutlicht. Die Staulänge wird nun auch mit einer linguistischen Variable modelliert, aber da die Zugehörigkeitsfunktionen der Terme nicht bekannt sind, werden in den mit `<<term>>` gekennzeichneten Spalten, nur die Zugehörigkeitsgrade gespeichert.

4.4 Was ist mit ungenauen Informationen

Um den Aufwand in dieser Arbeit in Grenzen zu halten, werde ich keine genaue Betrachtung eines Stereotyps zur Beschreibung von ungenauen Informationen tätigen. Ich werde hier nur kurz eine Modellierungsidee vorstellen:

Aus Abschnitt 3.2.1 ist bekannt, dass eine mögliche Art ungenaue Informationen in relationalen Datenbanken darzustellen, das Kontextmodell mit *scharfen Kontexten* ist. In dem darauf folgenden Abschnitt wurde dann auch erläutert, wie man scharfe Kontexte mit Hilfe von linguistischen Variablen ausdrücken kann, dazu dürfen die Zugehörigkeitsfunktionen der Terme nur keine Fuzzy-Mengen, sondern nur noch traditionelle klassische Mengen beschreiben. Eine Idee einen Stereotyp `<<imprecise>>` zu modellieren, die ich aber in dieser Arbeit nicht mehr weiter verfolgen werde, wäre folglich ein ähnlicher Ansatz wie beim Stereotyp `<<fuzzy>>`: die Erweiterung eines Datentyps um eine Menge von

Termen, die hier im Zusammenhang mit dem Kontextmodell dann Kontexte heißen würden und deren Zugehörigkeitsfunktionen auf die Beschreibung klassischer Mengen beschränkt sein müssten.

4.5 Zusammenfassung

Ein Ziel dieser Studienarbeit, eine generische Erweiterung des Common Warehouse Metamodel zur Beschreibung von imperfekten Informationen zu entwickeln, ist nun erreicht. Nachdem in den ersten Kapiteln dieser Arbeit die Grundlagen über das CWM, die neuen UML2-Profile, der Zusammenhang zwischen dem CWM und der neuen UML2 und Imperfektion in Informationssystemen gelegt wurden, ist in diesem Kapitel dann die geforderte Erweiterung vorgestellt worden.

Es ist nun möglich generisch mit dem CWM unscharfe und unsichere Informationen in Datenbanken zu beschreiben. Desweiteren wurde auch eine Idee zur Modellierung einer Erweiterung für ungenaue Informationen vorgestellt.

Nun gilt es diese Erweiterung in einem Tool zu implementieren und zu evaluieren.

5 Evaluierung

Nachdem im vorherigen Kapitel ein Profil zur Beschreibung von imperfekten Informationen im Rahmen des Common Warehouse Metamodel vorgestellt wurde, ist der nächste Schritt nun diese entwickelte Erweiterung an einem Beispiel umzusetzen. Eine beispielhafte Datenbank soll mit Hilfe des CWM, das zur Beschreibung von imperfekten Informationen erweitert worden ist, beschrieben werden. Da alle MOF-konformen Metamodelle und somit auch CWM- und UML-Modelle mit XML, genauer gesagt XMI, beschrieben werden können (Abschnitt 2.3.2), war es das Ziel letztendlich zwei XML-Dateien zu erhalten, wovon die eine das Profil *Imperfection* und die zweite das Beispielszenario beschreibt. Auf dieser Basis sollte es dann möglich sein, das Visualisierungstool zur Darstellung imperfekter Informationen aus Oliver Forsters Studienarbeit [6] um einen CWM-Importmechanismus zu erweitern.

In diesem Kapitel wird nun aber zuerst dargelegt, welche Schwierigkeiten bei der Beschreibung eines beispielhaften Datenbankszenarios mit dem CWM, das um ein Profil für die Darstellung imperfekter Informationen erweitert sein sollte, aufgetreten sind und wie sie umgegangen worden sind. Im darauf folgenden Abschnitt 6 wird auch dann noch kurz das Visualisierungstool und der entwickelte CWM-Importmechanismus vorgestellt.

5.1 Schwierigkeit

Der einfachste Weg das Profil *Imperfection* und das Beispielszenario zu modellieren, ist der Einsatz eines Modellierungstool, das einen XMI konformen Export der Modelle unterstützt. Da der Profilingmechanismus erst mit der UML Version 2.0 eingeführt worden ist, musste man sich auf die Tools beschränken, die schon UML2 konform sind.

Untersucht wurden daraufhin die Community Edition von Gentlewares *Poseidon for UML* [7] und die *EMF-based UML 2.0 Metamodel Implementation* aus dem Eclipse UML2 Project [5]. Die kostenlose Community Edition von Poseidon for UML hat sich dann aber sehr schnell als untauglich herausgestellt, da der angebliche XMI Export nur eine graphische Beschreibung des Modells darstellt und keine strukturelle Beschreibung des Modells.

Die Eclipse UML 2.0 Metamodel Implementation ist eigentlich eine Programmierschnittstelle zur Programmierung von UML2 Modellierungswerkzeugen. Dennoch war es, zwar nicht komfortabel, aber noch relativ einfach möglich, sehr einfache UML2 Profile damit zu modellieren und diese Profile dann in einem einfachen UML2 Modell einzusetzen. Auch der anschließende Export in eine XMI-konforme XML-Datei war problemlos möglich.

In dieser Studienarbeit wollte man kein UML2-Profil und Modell entwickeln, sondern ein Profil mit Stereotypen für Metaklassen des CWM und ein CWM-Modell. Folgendes Problem trat jedoch bei dem Einsatz des Eclipse UML2 Framework auf. Es war nur möglich Stereotypen für UML2 Metaklassen zu definieren und eben nicht für Metaklassen aus dem Common Warehouse Metamodel. Nachdem nun auch dieses „Modellierungswerkzeug“ nicht für den gewünschten Einsatzzweck geeignet war, musste man andere Möglichkeiten untersuchen, wie das modellierte Profil noch umgesetzt werden könnte.

5.2 Lösungsansätze

Letztendlich wollte man zwei XMI-konforme Dateien erhalten, die das Profil und das CWM-Modell beschreiben. Der naheliegendste Lösungsweg, diese Dateien eben selber von Hand zu schreiben, wurde nicht näher in Betracht gezogen, weil der dafür aufzubringende Aufwand in keinem Verhältnis mehr zum Nutzen gestanden wäre. Man hätte einen enormen Aufwand daran setzen müssen zu verstehen, wie UML2 Profile mit XMI beschrieben werden, daraus hätte man dann ableiten müssen, wie „CWM

Profile“ zu beschreiben sind. Und dann müsste man desweiteren noch verstehen, nach welchen Regeln ein UML2 Profil in einem UML2 Modell XMI konform wieder auftaucht.

Deshalb hat man sich entschieden, das Profil nicht exakt, UML2 konform umzusetzen, sondern es so weit wie möglich mit den aus dem CWM vorhandenen Mitteln zu simulieren. In Abschnitt 2.2 wurden die nun verwendeten Techniken, Einsatz von CWM Stereotypen und TaggedValues sowie modellierte Erweiterungen, schon ausführlich vorgestellt. Die folgenden Unterabschnitte zeigen nun kurz, wie die Simulation des CWM Profils *Imperfection* mit den jeweiligen Techniken möglich wäre und für welche Technik ich mich nun bei der Umsetzung entschieden habe.

Die Umsetzungsmöglichkeiten werden am Beispiel des Stereotyps `<<uncertain>>`, der nochmal in Abbildung 24 gezeigt ist, verdeutlicht.

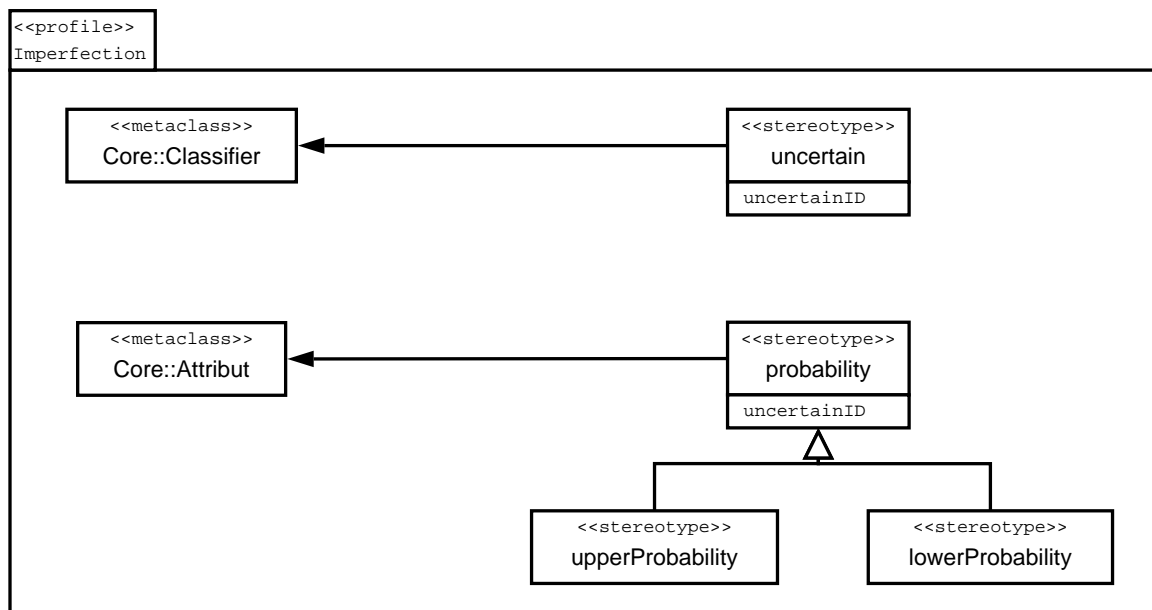


Abbildung 24: Stereotyp `<<uncertain>>`

5.2.1 Simulation mit standardisierten CWM 1.0 Stereotypen

Aus Abbildung 24 erkennt man, dass die Metaklassen *Classifier* und *Attribute* um ein weiteres Attribut `uncertainID` erweitert werden, falls sie mit dem Stereotyp `<<uncertain>>` markiert sind. Die Tatsache, dass eine Metaklasse des CWM ein weiteres zusätzliches, im Metamodell noch nicht vorhandenes Attribut erhält, kann man mit den vorhandenen TaggedValues sehr einfach erreichen. Wird im CWM Modell eine Instanz der Metaklasse *Classifier* gewünscht, die das zusätzliche Attribut `uncertainID` erhalten soll, so wird an diese Instanz einfach ein TaggedValue mit dem Tag `uncertainID` und dem gewünschten Wert angehängt.

Die Funktionalität des Attributes `uncertainID` — Herstellen eines Zusammenhangs zwischen dem unsicheren Typ und einem weiteren Attribut, das den Wahrscheinlichkeitsgrad aufnimmt — wird so zwar weitgehend simuliert, leider geht die Information verloren, dass das zusätzliche Attribut `uncertainID` nur bei den Metaklassen *Classifier* und *Attribut* zur Verfügung steht, soweit sie mit den Stereotypen `<<uncertain>>` beziehungsweise `<<probability>>` ausgezeichnet sind.

Um den Zusammenhang zwischen den Stereotypen und der `uncertainID` herzustellen, muss man also den TaggedValue in einem CWM Stereotyp kapseln. Den aus dem CWM schon bekannten Stereotypen können beliebig viele TaggedValues zugeordnet werden. Wird dann ein Modellelement mit diesem

Stereotyp ausgezeichnet, so erhält es über den Stereotyp auch den TaggedValue. Leider erreicht man auch so nicht das von den UML2 Profilen erwartete Verhalten. In einem UML2 Modell können bei jedem Einsatz des Stereotyps die zusätzlichen Attribute mit eigenen Werten belegt werden. Bei CWM Stereotypen, wird der Wert des zugeordneten TaggedValue jedoch bei der Definition des Stereotyps festgelegt. Wird dieser Stereotyp nun mehrmals für unterschiedliche Modellelemente verwendet, erhalten alle den gleichen Wert für den TaggedValue `uncertainID`.

Die Kombination aus den beiden Möglichkeiten, kommt dem Konzept der UML2 Profile am nächsten, obwohl auch hier der Zusammenhang zwischen dem Stereotyp und dem zusätzlichen Attribut nicht deutlich wird. Dazu wird der *Classifier* zum einen um den Stereotyp `<<uncertain>>`, der aber keine TaggedValues erhält, und zum zweiten um einen TaggedValue `uncertainID` ergänzt.

5.2.2 Simulation durch neue Unterklassen von bestehenden CWM 1.0 Metaklassen

Ein Stereotyp erweitert beispielsweise eine Metaklasse um weitere Attribute, ähnliche Semantik könnte erreicht werden, falls man eine eigene Unterklasse dieser Metaklasse definiert, die dann als Eigenschaft dieses zusätzliche Attribut erhält. Ganz vergleichbar sind diese Konzepte nicht, ein Stereotyp ist für alle angegebenen Basisklassen und deren Unterklassen einsetzbar, während eine Unterklasse einen eigenen Zweig in der Vererbungshierarchie eröffnet, was aber für unseren Einsatzzweck ausreichend ist. Im Common Warehouse Metamodel sind modellierte Erweiterungen sogar ausdrücklich erlaubt (Abschnitt 2.2), deshalb werden nun auch diese Techniken betrachtet, um das Profil zur Beschreibung imperfekter Informationen umzusetzen.

Es wurde schon erwähnt, dass eine neue Unterklasse nicht mehr so flexibel einsetzbar ist, wie ein Stereotyp, deshalb muss man sich schon jetzt eine Metaklasse herausdeuten, die man spezialisieren will, damit sie den Stereotyp `<<uncertain>>` darstellt. Da in allen Beispielen immer auf das relationale Datenmodell Bezug genommen worden ist, wird nun auch hier die Metaklasse *Relational::SQLSimpleType* ausgewählt, um sie zur Beschreibung von Unsicherheit zu erweitern. Wie oben erwähnt modelliert man nun eine Unterklasse *UncertainSimpleType* von *SQLSimpleType*, die ein weiteres Attribut `uncertainID` erhält.

Nun ist es möglich zu verdeutlichen, welche Spalten einer Tabelle nur von einem normalen SQL Typ sind, und welchen Spalten ein unsicherer Datentyp zugeordnet wird. Sobald dann der Typ *UncertainSimpleType* verwendet wird, steht auch das benötigte Attribut `uncertainID` zur Verfügung. Analog verfährt man mit den anderen Stereotypen des Profils *Imperfection*. Umgesetzt werden kann diese modellierte Erweiterung, indem die zusammen mit dem CWM veröffentlichte DTD um die neuen Konstrukte erweitert wird. Die wesentlichen Ausschnitte aus dieser veränderten Standard-DTD kann man im Anhang A einsehen.

Die Umsetzung des Profils mit Hilfe neuer modellierter Unterklassen, setzt am besten den Zusammenhang zwischen den Stereotypen im Profil und deren Eigenschaften um, leider geht dabei sehr viel Flexibilität, die die Verwendung von Stereotypen mit sich bringen würde, verloren.

5.2.3 Auswahl einer Lösungsmöglichkeit

Wie schon erwähnt, beziehen sich alle Beispiele in dieser Studienarbeit auf das relationale Datenmodell. Auch die Entwicklung eines Werkzeuges, welches ein Szenario, das mit dem erweiterten CWM beschrieben ist, verarbeiten kann, wird sich deshalb auf relationale Datenbanken beschränken. Aus diesem Grund ist der Verlust der Flexibilität beim Einsatz von neuen modellierten Unterklassen des CWM zu verschmerzen. Da diese Variante dabei dem eigentlich umzusetzenden UML2-Profil am nächsten kommt, wird auch nur diese Variante in einem beispielhaften Werkzeug implementiert, das eine Datenbankbindung für das Visualisierungstool aus [6] realisiert.

6 Konzeption und Implementierung eines Importmechanismus in ein Analyse-Werkzeug

Die Studienarbeit von Oliver Forster [6] beschäftigt sich mit der Frage, wie imperfekte Informationen visualisiert werden können. Um die getätigten Überlegungen verdeutlichen zu können, wurde auch ein beispielhaftes Analyse-Werkzeug konzipiert und implementiert. Dieses Werkzeug wird im Rahmen dieser Studienarbeit erweitert, so dass auch Daten aus einer Datenbank importiert werden können, welche mit dem Common Warehouse Metamodel beschrieben sind. Da das Visualisierungstool besonders die graphische Darstellung imperfekter Informationen unterstützt, musste es auch mit dem CWM möglich sein, die Art der Imperfektion zu kennzeichnen. Dies stellt nun den Zusammenhang zu dieser Studienarbeit dar, in der eine generische Erweiterung des CWM zur Beschreibung imperfekter Informationen vorgestellt wird.

Dieses Kapitel beschreibt das Konzept zur Implementierung eines Importmechanismus für das Visualisierungswerkzeug.

6.1 Das Visualisierungswerkzeug

Das Analyse-Werkzeug, das in [6] erstellt wurde, kann imperfekte Informationen mit unterschiedlichen Techniken visualisieren. Alle drei Arten von imperfekten Informationen — unsichere, ungenaue und unscharfe — können als zweidimensionale Balkendiagrammen dargestellt werden. Zusätzlich lassen sich unscharfe Informationen auch mit der ThemeRiver-Technik visualisieren. Weitere Informationen zu diesen beiden Techniken und auch zu weiteren untersuchten Visualisierungstechniken kann man in der Originalarbeit nachlesen.

Das Visualisierungstool gliedert sich in zwei Hauptbestandteile. Zum einen das Paket *de.uka.ipd.ovid.-visual.visualizer.information*, das Klassen zur Repräsentation imperfekter Informationen enthält. Und des weiteren das Paket *de.uka.ipd.ovid.visual.visualizer.layout*, das imperfekte Informationen, aus dem Paket *information*, graphisch darstellt. Bisher werden die Klassen aus dem Paket *information* durch das Auslesen von Textdateien mit Inhalten gefüllt, dazu existieren im selben Paket noch zusätzlich mehrere Loader-Klassen, die allesamt das Interface *IDataLoader* implementieren. Diese Schnittstelle beschreibt eine Methode `loadData`, die eine imperfekte Information aus dem *information* Paket zurückgibt.

6.2 Konzeption eines CWM-Importmechanismus

Die oben vorgestellte strikte Trennung der Visualisierungslogik von der Repräsentation der imperfekten Informationen, erlaubt eine einfache Ergänzung des Visualisierungstool um einen Importmechanismus, der auf einem erweiterten CWM basiert. Diese Ergänzung enthält weitere Loader-Klassen, die das *IDataLoader* Interface implementieren, aber die Daten gemäß der CWM Beschreibung laden.

Eine weitere grundsätzliche Konzeptionsentscheidung betraf den Umfang der zu unterstützenden Datenmodelle. Prinzipiell lassen sich mit dem CWM, und auch mit der in dieser Studienarbeit vorgestellten Erweiterung des CWM zur Beschreibung imperfekter Informationen, Szenarien mit mehreren Datenmodellen beschreiben. Im Rahmen dieser Studienarbeit ist es für Demonstrationszwecke jedoch ausreichend, erstmal nur relationale Datenbanken als Quellen für den Datenimport des Visualisierungswerkzeuges zu unterstützen.

Das Visualisierungswerkzeug unterstützt nur die Darstellung von Fließkommazahlen — entweder Zugehörigkeitsgrade zu Termen einer linguistischen Variable oder auch Datenwerte. Deshalb ist es ausreichend nur Tabellenspalten zu berücksichtigen, die Daten in einem darstellbaren Format aufweisen, primär erstmal nur Spalten vom Typ `NUMBER`.

Da das Visualisierungswerkzeug imperfekte Informationen vorallem als zweidimensionale Balkendiagramme darstellt, muss man auswählen aus welcher Spalte die Datenpunkte gewählt werden und wie die Achse beschriftet werden soll. Die Ideallösung würde dem Benutzer diese Auswahl überlassen. Da der Importmechanismus aber möglichst einfach gehalten werden soll, wurde entschieden, dass die Spalten jeweils über dem Primärschlüssel aufgetragen werden.

6.3 Implementierung

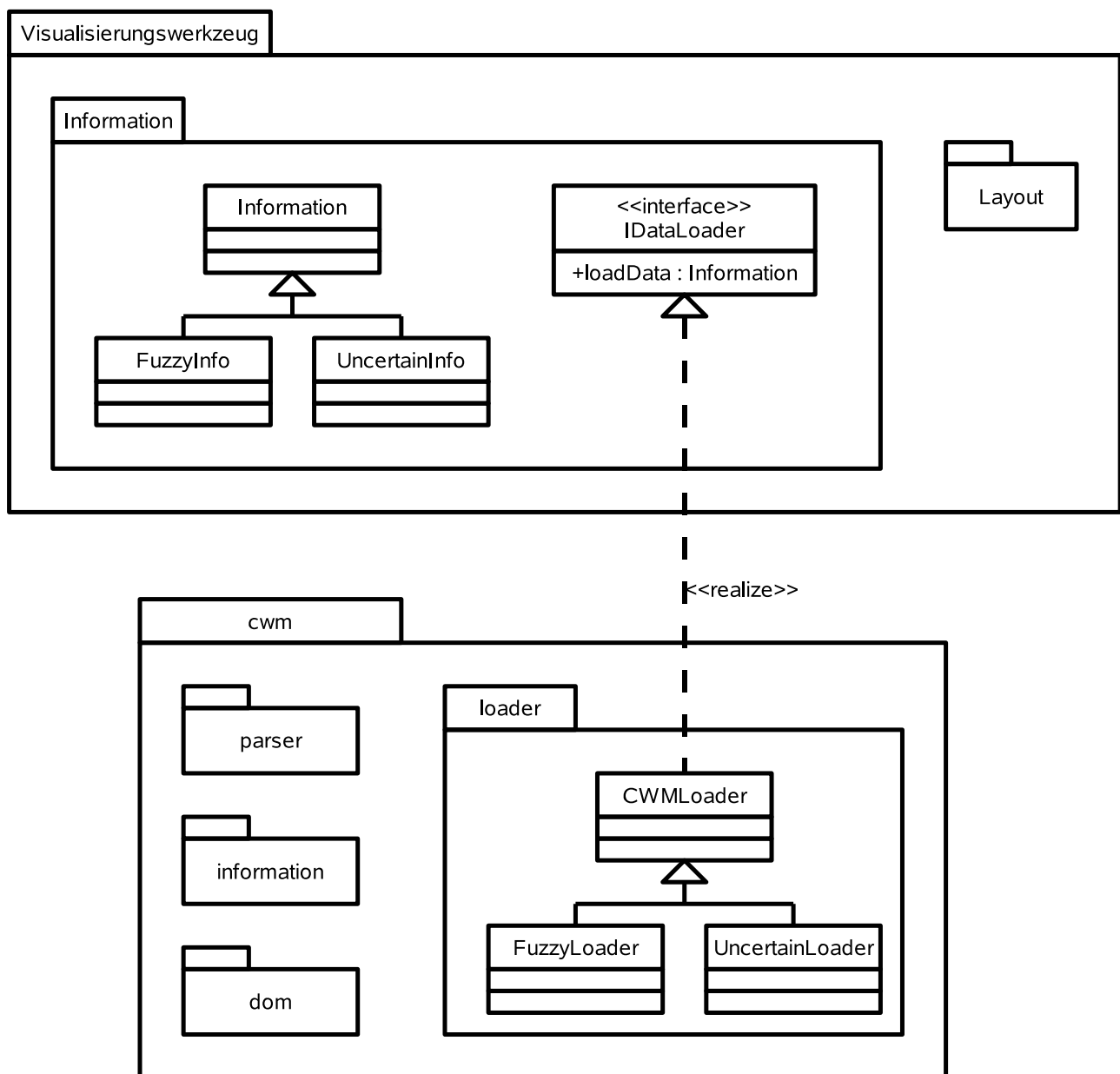


Abbildung 25: Ausschnitt aus dem Klassendiagramm des erweiterten Visualisierungswerkzeuges

Die Erweiterung des Visualisierungswerkzeuges, um eine CWM-konforme Datenbankschnittstelle, gliedert sich in fünf Pakete. In *de.uka.ipd.ovid.cwm* ist neben einer Klasse um das erweiterte Visualisierungswerkzeuges zu starten, eine Unterklasse des Originalhauptfensters zu finden, die dieses um

einen neuen Menüpunkt (CWM - LoadData) erweitert. Die neuen Loader-Klassen sind in *de.uka.ipd.ovid.cwm.loader* zu finden. Diese Klassen implementieren das *IDataLoader* Interface und lesen die imperfekten Informationen unter der Verwendung der JDBC-Schnittstelle aus einer Datenbank aus. Sie benötigen dazu jedoch Metainformationen darüber welche Tabellenspalten zum Beispiel zum Laden einer unsicheren Information gebraucht werden. Diese Metainformationen stehen in Klassen aus *de.uka.ipd.ovid.cwm.information*. Die Hauptarbeit leistet jedoch das Paket *de.uka.ipd.ovid.cwm.parser*. Die Klasse *CWMParser* liest eine CWM-konforme XMI-Datei ein und kann alle darin beschriebenen Tabellen extrahieren. Ein *TableParser* bestimmt zu jeder Tabelle nun den Primärschlüssel. Desweiteren kann er aus der XML-Datei Metainformationen über die Art der Imperfektion in dieser Tabelle und wie diese in der Tabelle repräsentiert ist, bestimmen. Das Paket *de.uka.ipd.ovid.cwm.dom* enthält zwei Implementierungen von Schnittstellen des *org.w3c.dom* Interfaces, die in der verwendeten DOM-Implementierung nicht vorhanden waren. Zum einen eine Implementierung der `getElementById`-Methode des *Document*-Interfaces und zum zweiten eine Implementierung des *org.w3c.dom.traversal.NodeIterator*.

7 Zusammenfassung und Ausblick

In dieser Studienarbeit wurde das Common Warehouse Metamodel erweitert, um imperfekte Informationen beschreiben zu können. Dazu wurde zuerst untersucht, mit welchen Techniken das CWM erweitert werden kann. Neben denen in der Spezifikation vorgestellten Techniken (*Stereotypes*, *TaggedValues* und der modellierten Erweiterung), wurde vor allem der neue in der UML2 Infrastructure beschriebene *Profiling*-Mechanismus genauer betrachtet. Nachdem dieser neue Mechanismus vorgestellt wurde, wurde aufgezeigt, dass eine zukünftige Version des CWM mit dieser Technik erweitert werden kann. Aus diesem Grunde habe ich mich in dieser Studienarbeit dazu entschlossen, das CWM schon jetzt mit dieser neuen Technik zu erweitern.

Ebenso wurde in dieser Arbeit dargelegt, wie imperfekte Informationen in Datenbanken dargestellt werden können. Darauf aufbauend wurde dann ein CWM-Profil zur Beschreibung imperfekter Informationen modelliert. Das Hauptaugenmerk während der Modellierung richtete sich darauf, eine generische Erweiterung des CWM zu entwickeln. Das entstandene Profil ist somit nicht auf ein spezielles Datenmodell beschränkt, sondern ist zur Beschreibung imperfekter Informationen geeignet, unabhängig davon in welchem Datenmodell sie repräsentiert werden.

Abschließend wurde ein Importmechanismus für ein existierendes Analyse-Werkzeug konzipiert und implementiert, der Gebrauch vom dem vorgestellten CWM-Profil macht.

In dieser Studienarbeit wird die Behauptung vertreten, dass neue Versionen des Common Warehouse Metamodel mit dem Konzept der Profile, das in der UML 2.0 Infrastructure vorgestellt wird, erweitert werden können. In Abschnitt 2.3.4 wurde diese Behauptung ausführlich begründet. Aber erst eine Veröffentlichung einer Version 2.0 des CWM durch die Object Management Group wird zeigen, ob man damit auch wirklich richtig lag. Wenn dann in Zukunft eine neue Version des CWM spezifiziert ist, kann man auch das in dieser Studienarbeit vorgestellte Profil zur Beschreibung imperfekter Informationen auf Übereinstimmung mit dieser neuen CWM Version untersuchen.

Bisher erweitern die Stereotypen `<<fuzzy>>` und `<<fuzzy2>>` die CWM-Metaklasse *Core::DataType*, was im Rahmen dieser Studienarbeit völlig ausreichend war. Für zukünftige Fragestellung kann es aber auf jeden Fall von Bedeutung sein, ob diese Erweiterung nicht auch an der Oberklasse von *DataType*, nämlich *Core::Classifier*, ansetzen könnte.

Sobald die neue CWM Version spezifiziert ist, muss man die in dieser Studienarbeit vorgestellte CWM-basierte Datenbankschnittstelle für das graphische Analysewerkzeug an diese neue Version anpassen. Der bisher gewählte Weg, über die Simulation des Profils mit CWM 1.0 konformen Mitteln — *Stereotypes* und *TaggedValues*, sowie der modellierten Erweiterung — wird dann nicht mehr nötig sein.

Bisher kann die Datenbankschnittstelle des Visualisierungswerkzeuges nur relationale Datenbanken über die JDBC-Schnittstelle ansprechen. Zusammen mit der CWM Version 2.0, wird es dem Profil möglich, imperfekte Informationen zu beschreiben, die in einem beliebigen Datenmodell repräsentiert sind. Somit bietet es sich an, in Zukunft das Visualisierungswerkzeug um weitere Schnittstellen zu ergänzen, damit zum Beispiel auch eine XML-Datenbank angesprochen werden kann.

A Erweiterung der CWM-DTD

```

<!-- ===== CWMEXT:FuzzyDataType ===== -->
<!ELEMENT CWMEXT:TrapezoidMembershipFunction EMPTY>
<!ATTLIST CWMEXT:TrapezoidMembershipFunction
  ll CDATA #IMPLIED
  ul CDATA #IMPLIED
  ur CDATA #IMPLIED
  lr CDATA #IMPLIED>

<!ELEMENT CWMEXT:FuzzyTerm (
  CWM:Expression |
  CWMEXT:TrapezoidMembershipFunction ) >
<!ATTLIST CWMEXT:FuzzyTerm
  name CDATA #IMPLIED >

<!ELEMENT CWMEXT:LinguisticVariable (CWMEXT:FuzzyTerm*)>
<!ATTLIST CWMEXT:LinguisticVariable
  name CDATA #IMPLIED >

<!ENTITY % CWMEXT:FuzzyDataTypeFeatures '(%CWMRDB:SQLSimpleTypeFeatures;* |
  CWMEXT:LinguisticVariable)''>
<!ENTITY % CWMEXT:FuzzyDataTypeAtts '%CWMRDB:SQLSimpleTypeAtts;''>
<!ELEMENT CWMEXT:FuzzyDataType (%CWMEXT:FuzzyDataTypeFeatures;)>
<!ATTLIST CWMEXT:FuzzyDataType %CWMEXT:FuzzyDataTypeAtts;>

<!-- ===== CWMEXT:Fuzzy2DataType ===== -->
<!ENTITY % CWMEXT:Fuzzy2DataTypeAtts '%CWMRDB:SQLSimpleTypeAtts;
  lingVarName CDATA #IMPLIED
  lingVarID CDATA #REQUIRED''>
<!ELEMENT CWMEXT:Fuzzy2DataType (%CWMRDB:SQLSimpleTypeFeatures;)*>
<!ATTLIST CWMEXT:Fuzzy2DataType %CWMEXT:Fuzzy2DataTypeAtts;>

<!-- ===== CWMEXT:UncertainDataType ===== -->
<!ENTITY % CWMEXT:UncertainDataTypeAtts '%CWMRDB:SQLSimpleTypeAtts;
  uncertainID CDATA #REQUIRED''>
<!ELEMENT CWMEXT:UncertainDataType (%CWMRDB:SQLSimpleTypeFeatures;)*>
<!ATTLIST CWMEXT:UncertainDataType %CWMEXT:UncertainDataTypeAtts;>

```

Abbildung 26: Simulation der Stereotypen für imperfekte Datentypen

```
<!-- ===== CWMEXT:ProbabilityColumn ===== -->
<!ENTITY % CWMEXT:ProbabilityColumnAtts '%CWMRDB:ColumnAtts;
    uncertainID CDATA #REQUIRED'>
<!ELEMENT CWMEXT:ProbabilityColumn (%CWMRDB:ColumnFeatures;)*>
<!ATTLIST CWMEXT:ProbabilityColumn %CWMEXT:ProbabilityColumnAtts;>

<!-- ===== CWMEXT:TermColumn ===== -->
<!ENTITY % CWMEXT:TermColumnAtts '%CWMRDB:ColumnAtts;
    termName CDATA #IMPLIED
    lingVarID CDATA #IMPLIED
    lingVarName CDATA #IMPLIED'>
<!ELEMENT CWMEXT:TermColumn (%CWMRDB:ColumnFeatures;)*>
<!ATTLIST CWMEXT:TermColumn %CWMEXT:TermColumnAtts;>
```

Abbildung 27: Simulation der Stereotypen <<probability>> und <<term>>

Literatur

- [1] Stau bremsst volkswirtschaftliches Wachstum. <http://www.presseportal.de/>, März 2005.
- [2] Marc Born, Eckhardt Holz, and Olaf Kath. *Softwareentwicklung mit UML2*. Addison-Wesley, 2004.
- [3] BMVBW Verkehrsnachrichten. Heft 12/2000.
- [4] Bastian Chlond, Wilko Manz, and Dirk Zumkeller. Stagnation der Verkehrsnachfrage — Sättigung oder Episode? *Internationales Verkehrswesen*, pages 396–403, 2002. Heft 9/2002.
- [5] Eclipse UML2 Project. <http://www.eclipse.org/uml2/>.
- [6] Oliver Forster. Visualisierung imperfekter Informationen in einem Analyse-Werkzeug. Studienarbeit, Universität Karlsruhe (TH), Fakultät für Informatik, Dezember 2004.
- [7] Gentleware Poseidon for UML. <http://www.gentleware.com/>.
- [8] Alexander Haag. Konzeption und Umsetzung einer Erweiterung des Common Warehouse Metamodel (CWM) zur Beschreibung von imperfekten Daten. Studienarbeit, Universität Karlsruhe (TH), Fakultät für Informatik, Juli 2004.
- [9] Erik Koop. Datenbankunterstützung für imperfekte Daten im Verkehrsumfeld. Diplomarbeit, Universität Karlsruhe (TH), Fakultät für Informatik, Juni 2004.
- [10] Andreas Merkel. Konzeption und Umsetzung einer Schemaerweiterung durch Kontexte unter Berücksichtigung von Aggregierungsanfragen. Studienarbeit, Universität Karlsruhe (TH), Fakultät für Informatik, Juni 2004.
- [11] Object Management Group Inc. *OMG Unified Modeling Language Specification Version 1.3*, März 2000. <http://www.omg.org/cgi-bin/doc?formal/00-03-01>.
- [12] Object Management Group Inc. *Common Warehouse Metamodel (CWM) Specification*, Februar 2001. <http://www.omg.org/cgi-bin/doc?ad/2001-02-01>.
- [13] Object Management Group Inc. *Common Warehouse Metamodel (CWM) Specification Volume 2. Extensions*, Februar 2001. <http://www.omg.org/cgi-bin/doc?ad/2001-02-02>.
- [14] Object Management Group Inc. *OMG Unified Modeling Language Specification Version 1.4*, September 2001. <http://www.omg.org/cgi-bin/doc?formal/01-09-67>.
- [15] Object Management Group Inc. *Unified Modelling Language: Superstructure, Version 2.0*, 2003. <http://www.omg.org/cgi-bin/doc?ptc/2003-08-02>.
- [16] Object Management Group Inc. *Meta Object Facility (MOF) 2.0 Core Specification*, April 2004. <http://www.omg.org/cgi-bin/doc?ptc/2003-10-04>.
- [17] Object Management Group Inc. *UML 2.0 Infrastructure Specification*, April 2004. <http://www.omg.org/cgi-bin/doc?ptc/2003-09-15>.
- [18] Object Management Group (OMG). <http://www.omg.org/>.
- [19] Stärkung der Selbstorganisationsfähigkeit im Verkehr durch I+K-gestützte Dienste (OVID). <http://www.ovid.uni-karlsruhe.de/>.

- [20] John Poole, Dan Chang, Douglas Tolbert, and David Mellor. *Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration*. OMG Press, John Wiley and Sons, Inc., New York, 2002.
- [21] S. Schnittger and B. Chlond. Skriptum zur Vorlesung Verkehrstechnik und -telematik, November 2003. Institut für Verkehrswesen, Universität Karlsruhe.
- [22] René Witte. *Architektur von Fuzzy-Informationssystemen*. Dissertation, Universität Karlsruhe (TH), 2002.
- [23] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard Thomas Snodgrass, V.S. Subrahmanian, and Roberto Zicari. *Advanced Database Systems*. Morgan Kaufmann Publishers Inc., 1997.