Becker, Jörg; Janiesch, Christian; Knackstedt, Ralf; Kramer, Stephan; Seidel, Stefan

**Working Paper**

# Konfigurative Referenzmodellierung mit dem H2-Toolset

Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 114

**Provided in Cooperation with:**
University of Münster, Department of Information Systems

*Suggested Citation:* Becker, Jörg; Janiesch, Christian; Knackstedt, Ralf; Kramer, Stephan; Seidel, Stefan (2006) : Konfigurative Referenzmodellierung mit dem H2-Toolset, Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 114, Westfälische Wilhelms-Universität Münster, Institut für Wirtschaftsinformatik, Münster

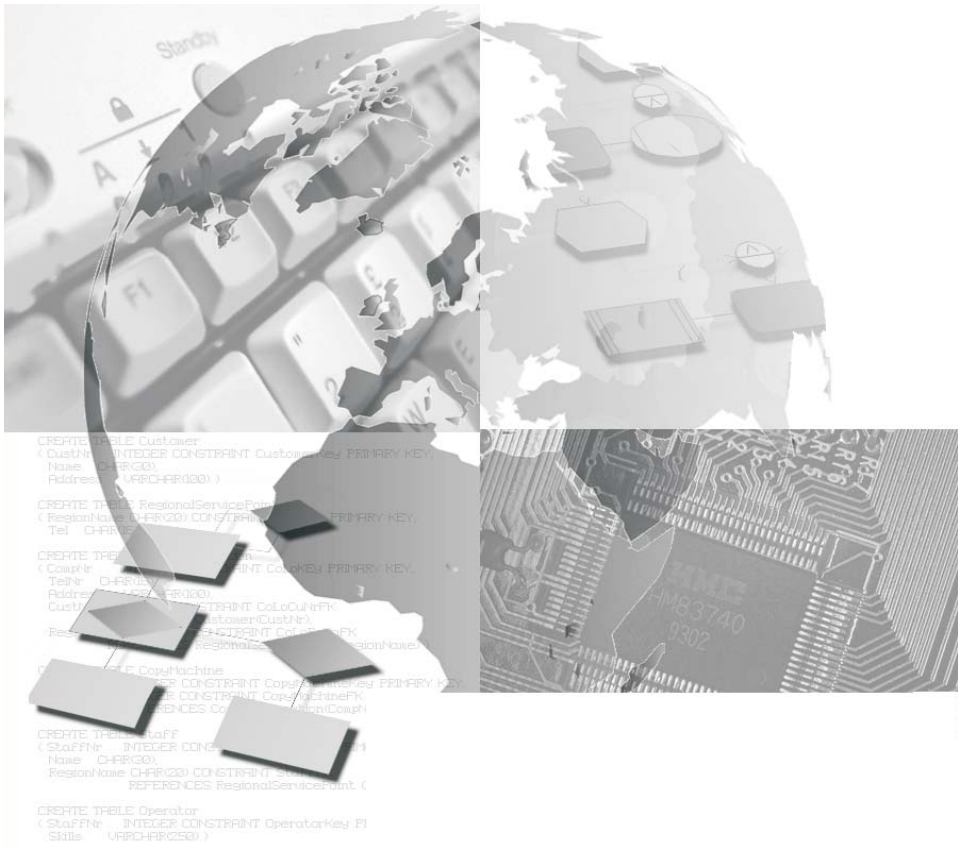
This Version is available at:
https://hdl.handle.net/10419/59564

Westfälische
Wilhelms-Universität
Münster

WIRTSCHAFTS
INFORMATIK

Institut für Wirtschaftsinformatik

Arbeitsberichte



Arbeitsbericht Nr. 114

# Konfigurative Referenzmodellierung mit dem H2-Toolset

Jörg Becker, Christian Janiesch, Ralf Knackstedt,
Stephan Kramer, Stefan Seidel

*unter Mitarbeit des Projektseminars COIN.*

**Arbeitsberichte des Instituts für Wirtschaftsinformatik**
Herausgeber: Prof. Dr. J. Becker, Prof. Dr. H. L. Grob, Prof. Dr. S. Klein,
Prof. Dr. H. Kuchen, Prof. Dr. U. Müller-Funk, Prof. Dr. G. Vossen

Arbeitsbericht Nr. 114

# Konfigurative Referenzmodellierung mit dem H2-Toolset

Jörg Becker, Christian Janiesch, Ralf Knackstedt, Stephan Kramer, Stefan Seidel

*unter Mitarbeit des Projektseminars COIN:*

Boris Beumers, Björn Böhmer, Tamer El-Hawari, Stefanie Filius, Stefan Fleischer, Stefan Schellhammer, Alexander Simons und Martin Pellengahr

**ISSN 1438-3985**

# Inhaltsverzeichnis

1 Grundlagen konfigurativer Referenzmodellierung ..... 1

2 Konfigurative Referenzmodellierung mit dem H2-Toolset ..... 4
- 2.1 Konfiguration von Referenzmodellen ..... 4
- 2.2 Basiskomponenten des H2-Toolsets ..... 9
  - 2.2.1 Übersicht ..... 9
  - 2.2.2 Grundlagen des Meta-Meta-Modells des H2-Toolset ..... 10
  - 2.2.3 Meta-Modellierung mit dem H2-Toolset ..... 12
  - 2.2.4 Modellierung mit dem H2-Toolset ..... 19
- 2.3 Neue und ergänzte Komponenten des H2-Toolsets ..... 24

3 Fachkonzept, DV-Konzept und Implementierung der einzelnen Funktionen ..... 28
- 3.1 Erstellung und Verwaltung von Referenzmodellen ..... 28
  - 3.1.1 Fachkonzept ..... 28
  - 3.1.2 DV-Konzept und Implementierung ..... 29
- 3.2 Benutzer- und Rechteverwaltung ..... 36
  - 3.2.1 Fachkonzept ..... 36
  - 3.2.2 DV-Konzept und Implementierung ..... 39
- 3.3 Erstellung und Verwaltung von Konfigurationsparametern ..... 47
  - 3.3.1 Fachkonzept ..... 47
  - 3.3.2 DV-Konzept und Implementierung ..... 49
- 3.4 Erstellung und Verwaltung von Termen ..... 62
  - 3.4.1 Fachkonzept ..... 62
  - 3.4.2 DV-Konzept und Implementierung ..... 63
- 3.5 Erstellung und Verwaltung der Selektionsmechanismen ..... 69
  - 3.5.1 Fachkonzept ..... 69
  - 3.5.2 DV-Konzept und Implementierung ..... 71
- 3.6 Erstellung und Verwaltung der Bezeichnungsvariation ..... 82
  - 3.6.1 Fachkonzept ..... 82
  - 3.6.2 DV-Konzept und Implementierung ..... 83
- 3.7 Darstellungsvariation der Symbole ..... 89
  - 3.7.1 Fachkonzept ..... 89
  - 3.7.2 DV-Konzept und Implementierung ..... 90
- 3.8 Konfiguration und Zuordnung von Konfigurationsparametern zu Benutzern ..... 94
  - 3.8.1 Fachkonzept ..... 94
  - 3.8.2 DV-Konzept und Implementierung ..... 95
- 3.9 Erstellung und Verwaltung von Kantentypen ..... 97
  - 3.9.1 Fachkonzept ..... 97
  - 3.9.2 DV-Konzept und Implementierung ..... 99

4 Status quo und Entwicklungsperspektiven konfigurativer Referenzmodellierung ..... 104

Literaturverzeichnis ..... 106

Anhang: Implementierungstechnische Details ..... 108

# Abbildungsverzeichnis


Abb. 1: Aufbau des Arbeitsberichts ..... 2
Abb. 2: Workflow der konfigurativen Referenzmodellierung im H2-Toolset ..... 5
Abb. 3: Perspektivenwahl zur Konfiguration von Referenzmodellen ..... 6
Abb. 4: Kern-Repository des H2-Toolset ..... 10
Abb. 5: Sprachdefinition „Mini H2“ ..... 12
Abb. 6: Darstellung von Hierarchiestufen ..... 16
Abb. 7: Erzeugung von Objektkopien ..... 18
Abb. 8: Erzeugung von Objekten ..... 19
Abb. 9: Erzeugen von Objektausprägungen ..... 21
Abb. 10: Erzeugen von Objektausprägungen mit kompletter Struktur ..... 22
Abb. 11: Löschen von Objektausprägungen (ganze Hierarchiestufe) ..... 24
Abb. 12: Softwarearchitektur des H2-Toolsets ..... 26
Abb. 13: ER-Diagramm der Referenzmodellkomponenten ..... 29
Abb. 14: Klassendiagramm des *RefModels* und seiner Komponenten ..... 31
Abb. 15: Klassendiagramm der Klasse *RefModel* ..... 31
Abb. 16: Struktur des Datenbank- Explorer ..... 32
Abb. 17: Kontextmenü eines Referenzmodells ..... 33
Abb. 18: Hinzufügen eines Referenzmodells ..... 35
Abb. 19: ER-Diagramm der Benutzer- und Rechteverwaltung ..... 37
Abb. 20: Code-Fragment zur Aktivierung von GUI-Befehlen gemäß zugehöriger Funktionsrechte ..... 42
Abb. 21: Klassendiagramm der Benutzerverwaltung ..... 43
Abb. 22: Benutzerverwaltung im Datenbank-Explorer ..... 44
Abb. 23: Zuordnung von Funktionsrechten zu Benutzergruppen ..... 45
Abb. 24: Zuweisung von Benutzergruppen an Benutzer in Referenzmodellen ..... 46
Abb. 25: Ändern der Benutzerattribute ..... 46
Abb. 26: Struktur der Konfigurationsparameter ..... 47
Abb. 27: Workflow zur Verwendung von Konfigurationsparametern ..... 50
Abb. 28: ER-Diagramm der Konfigurationsparameter ..... 51
Abb. 29: Struktur der Ontologien ..... 53
Abb. 30: Metamodell der Konfigurationsparameter und Ontologien ..... 55
Abb. 31: Beispiel der Erweiterung der Konfigurationssprache ..... 57
Abb. 32: Automatisch generierte Ontologien im Modell-Editor für Konfigurationsparameter ..... 58
Abb. 33: Vorselektion von Konfigurationsparameter für ein Referenzmodell ..... 59

Abb. 34: Konfigurationsparameter-Zuweisung zu Benutzern ..... 60
Abb. 35: Auswahl der Konfigurationsparameter zur Konfiguration ..... 61
Abb. 36: Gegenüberstellung von One-of-Ausdruck und Or-Ausdruck ..... 62
Abb. 37: Erzeugung des Syntaxbaumes eines Termtextes..... 64
Abb. 38: Termeditor..... 66
Abb. 39: ER-Diagramm der Konfigurationsmechanismen..... 70
Abb. 40: Klassendiagramm der Konfigurationsmechanismen ..... 71
Abb. 41: Abhängigkeiten der zu selektierenden Konstrukte ..... 74
Abb. 42: Zuweisen von Termen..... 77
Abb. 43: Terme im Configuration Customizer ..... 77
Abb. 44: Termauswahl..... 78
Abb. 45: Sprachselektion und Modellselektion ..... 79
Abb. 46: Kontextselektion, Objekttypselektion und kontextabhängige Objekttypselektion ..... 80
Abb. 47: Elementselektion..... 81
Abb. 48: ER-Diagramm der Bezeichnungsvariation ..... 83
Abb. 49: Algorithmus des Variationsmechanismus..... 86
Abb. 50: Basiswortkatalog der Bezeichnungsvariation ..... 87
Abb. 51: Zuweisung von Termen zu Synonymen..... 87
Abb. 52: Zuweisung von Synonymgruppen zu Modellelemente..... 88
Abb. 53: Auswirkung der Bezeichnungsvariation ..... 89
Abb. 54: ER-Modell für die Darstellungsvariation der Symbole ..... 90
Abb. 55: Darstellungsvariation ..... 92
Abb. 56: Anlegen eines Image-Sets..... 93
Abb. 57: Perspektivenwechsel bei einem Referenzmodell..... 95
Abb. 58: ER-Diagramm der Zuweisung von Konfigurationsparameterdefinitionen zu Benutzern ..... 96
Abb. 59: ER-Diagramm der Kanten..... 98
Abb. 60: Klassendiagramm der Kanten ..... 99
Abb. 61: Nutzung von Kantentypen ..... 101
Abb. 62: Anlegen von Kanten..... 103
Abb. 63: ER-Modell der neuen Komponenten ..... 110
Abb. 64: Gesamt ER-Modell ..... 111
Abb. 65: H2 Klassendiagramm..... 112

# Tabellenverzeichnis

Tab. 1: Beispielhafter morphologischer Kasten ..... 5
Tab. 2: Beispielhafte Entscheidungstabelle ..... 7
Tab. 3: Tabelle ContextRules ..... 13
Tab. 4: Tabelle Context ..... 14
Tab. 5: Tabelle ObjectDefinition ..... 16
Tab. 6: Tabelle ObjectOccurence ..... 17
Tab. 7: Relationstabelle ActionRule ..... 19
Tab. 8: Tabelle Refmodel ..... 30
Tab. 9: Tabellen der Referenzmodellkomponenten ..... 30
Tab. 10: Funktionsrechte im H2-Toolset ..... 37
Tab. 11: Tabelle UserGroup ..... 39
Tab. 12: Tabelle UserList ..... 40
Tab. 13: Tabelle Userfunction ..... 41
Tab. 14: Tabelle UserGroupRefmodelRelation ..... 41
Tab. 15: Berücksichtigung von Ontologien in Programmbausteinen ..... 49
Tab. 16: Tabelle ConfParam ..... 51
Tab. 17: Tabelle Confparamdefinition ..... 52
Tab. 18: Tabelle Confparamtype ..... 53
Tab. 19: Daten der Tabelle Confparamtype ..... 53
Tab. 20: Tabelle Rules ..... 54
Tab. 21: Klassen für die Konfigurationsparameter ..... 55
Tab. 22: Tabelle Term ..... 63
Tab. 23: Buttons des Termeditors ..... 67
Tab. 24: Tabelle Termlanguagerelation ..... 72
Tab. 25: Tabelle Termmodelrelation ..... 72
Tab. 26: Tabelle Termobjecttypedefinition ..... 72
Tab. 27: Tabelle Termcontextrelation ..... 72
Tab. 28: Tabelle Termcontextrulerelation ..... 72
Tab. 29: Tabelle Termobjectdefinitionrelation ..... 73
Tab. 30: Tabelle Termobjectoccurencerelation ..... 73
Tab. 31: Tabelle Synonym ..... 84
Tab. 32: Tabelle SynonymObjectDefinitionRelation ..... 84
Tab. 33: Tabelle Termsynonymrelation ..... 85
Tab. 34: Tabelle Image ..... 90

Tab. 35: Tabelle ImageSet ..... 91
Tab. 36: Tabelle Imagesetimageobjecttypedefinitionrelation ..... 91
Tab. 37: Tabelle Termimagesetrelation ..... 91
Tab. 38: Tabelle Userconfparamrelation ..... 96
Tab. 39: Tabelle EdgeTypeContextRuleRelation ..... 100
Tab. 40: Tabelle Edge ..... 100
Tab. 41: Tabelle EdgeType ..... 100
Tab. 42: Cachezugriff von Methoden ..... 109
Tab. 43: Divergenzen zwischen Entitytypen und Tabellennamen ..... 109

# 1 Grundlagen konfigurativer Referenzmodellierung

Die Spezifikation und Gestaltung von Informationssystemen ist eine komplexe und unübersichtliche Aufgabe. Sachverhalte sind so zu beschreiben und strukturieren, dass sowohl die eingesetzten Anwendungssysteme als auch die Organisationsabläufe adäquat abgebildet werden. Eine probate Möglichkeit hierfür stellt der Einsatz von Informationsmodellen dar.[1] Ein Informationsmodell ist als das Ergebnis einer Konstruktionsleistung eines Modellierers definiert, der für die Anwendungs- und Organisationsgestaltung Informationen über darzustellende Elemente eines Systems zu einer Zeit unter Rückgriff auf eine Modellierungssprache als relevant deklariert.[2]

Eine spezielle Klasse von Informationsmodellen stellen Referenz-Informationsmodelle dar.[3] Diese werden nicht für einen konkreten Anwendungskontext erstellt, sondern sind generalisierte Ausgangsmodelle für eine Klasse abstrakter Anwendungsgebiete. Mit dem Einsatz von Referenz-Informationsmodellen als Ausgangslösungen für die Entwicklung projektspezifischer Modelle ist auf Anwenderseite nicht zuletzt die Erwartung der Erhöhung der Wirtschaftlichkeit der eigenen Informationsmodellierung verbunden.[4] Als Vergleichsbasis zur Beurteilung eigener Modelle und bedarfsgerecht zu modifizierende Ausgangslösung, die die Übernahme von Modellteilen und Begriffssystemen erlaubt, stellen Referenzmodelle insbesondere durch das Prinzip der Wiederverwendung den Transfer betriebswirtschaftlichen Know-hows in Aussicht. Den durch den Einsatz von Referenz-Informationsmodellen erwarteten Einsparungen stehen jedoch nicht unerhebliche Adaptions- und Pflegekosten gegenüber, die es zu kompensieren gilt.[5] Der Referenzmodellersteller steht aus diesem Grund vor dem Dilemma, dass die Akzeptanz eines Referenzmodells einerseits in hohem Maße vom Anpassungsbedarf an die spezifischen Bedürfnissen des Kunden abhängt, andererseits jedoch auf nur einzelne Unternehmensspezifika zugeschnittene Spezialmodelle das Absatzrisiko aufgrund eines zu geringen Adressatenkreises erhöhen.[6] Einen Ausweg aus diesem Dilemma stellen *konfigurierbare* Referenzmodelle in Aussicht,[7] die mit Hilfe von Konfigurationsmechanismen die regelbasierte Ableitung von Referenzmodellvarianten in Abhängigkeit von den Anwendungskontext beschreibenden Konfigurationsparametern vorsehen. Somit kann der Anpassungsaufwand gering gehalten und der Zugang zu dem in Referenzmodellen enthaltenen Wissen erleichtert werden.[8] Um Referenzmodellierungstechniken wirtschaftlich effizient

[1] Vgl. Becker et al. (2002), S. 25.
[2] Vgl. Schütte (1998), S. 63.
[3] Vgl. vom Brocke (2003), S. 34 ff.
[4] Vgl. z. B. Becker et al. (2002), S. 26 sowie Becker, Knackstedt (2003), S. 416.
[5] Vgl. Becker, Delfmann, Rieke (2004), S. 2.
[6] Vgl. Becker et al. (2002), S. 26.
[7] Vgl. Becker et al. (2002), S. 25.
[8] Vgl. Becker et al. (2002), S. 26 sowie Becker, Knackstedt (2003), S. 415.

und praktisch einsetzen zu können, bedarf es neben der methodischen Unterstützung der Techniken auch eine Werkzeugunterstützung, um die hohe Eigenkomplexität sowie den mit der Nutzung der konfigurativen Modelle verbundenen Wartungsaufwand zu reduzieren.[9]

Die Modellvarianten des Referenzmodells werden ex ante durch den Modellierer festgelegt. Um die Anwendung der Konfigurationsmechanismen möglichst effizient zu gestalten, ist eine Differenzierung der Konfigurationsmechanismen in unterschiedliche Granularitätsgrade sinnvoll. Es wird daher in diesem Arbeitsbericht die konfigurative Referenzmodellierung nach Becker et al. zu Grunde gelegt.[10]

Der Aufbau des Arbeitsberichts gliedert sich wie folgt:

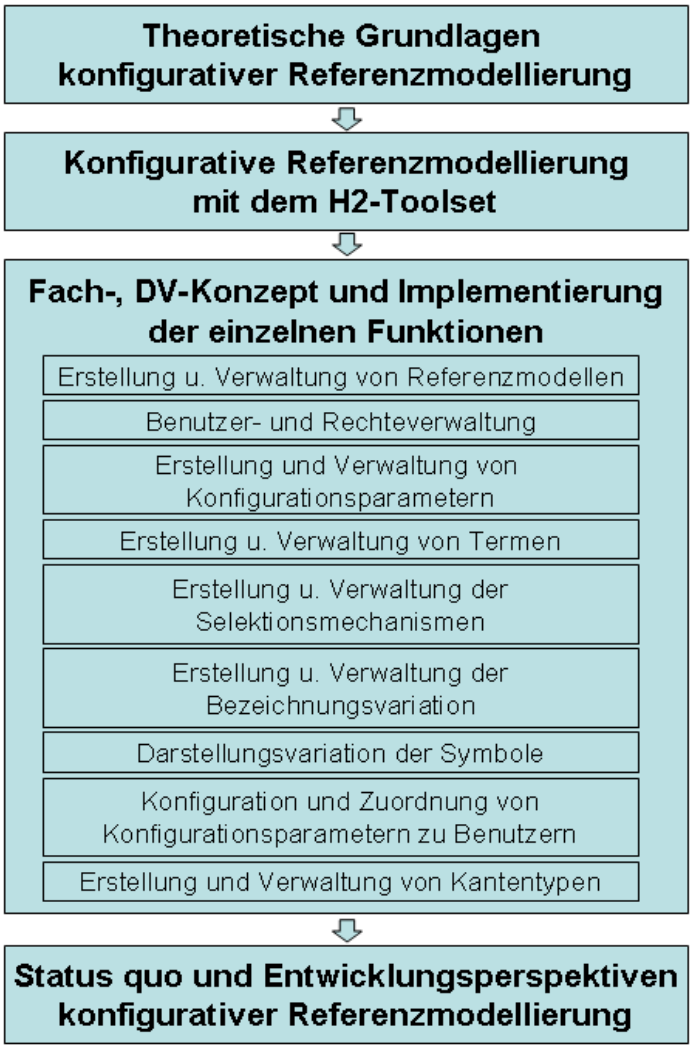



**Abb. 1:** Aufbau des Arbeitsberichts

[9] Vgl. Becker et al. (2002) und Becker, Knackstedt (2004).
[10] Vgl. Becker, Delfmann, Knackstedt (2004), S. 254.

In Kapitel 2 wird zunächst der grundlegende Referenzmodellierungsprozess beschrieben. Weiterhin werden die Grundlagen des H2-Toolsets dargestellt: Nach einer Übersicht des Kern-Repositories wird die Meta-Modellierung sowie die Modellierung mit dem H2-Toolset erläutert und auf die notwendigen Änderungen für die konfigurative Referenzmodellierung eingegangen.

Kapitel 3 beinhaltet die Erweiterung des H2-Toolsets um die Mechanismen der konfigurativen Referenzmodellierung. Dazu wird zunächst jeweils der theoretische Hintergrund auf fachkonzeptioneller Basis vermittelt, bevor eine Transformation in ein DV-Konzept und die Implementierung erläutert wird. Neben den fünf Konfigurationsmechanismen wird an entsprechender Stelle auch auf notwendige Modifikationen und Erweiterungen des ursprünglichen Toolsets eingegangen.

In Kapitel 4 finden eine Zusammenfassung der vorgenommenen Änderungen und eine Einordnung in den betriebswirtschaftlichen Gesamtkontext statt. Insbesondere wird auf die Möglichkeiten der praktischen Anwendung eingegangen. Ein Ausblick auf notwendige Verbesserungen und weiteren Forschungsbedarf bildet den Abschluss dieses Kapitels.

Im Anhang werden implementierungstechnische Details erläutert, insbesondere im Zusammenhang mit der Umstellung des Toolsets von .NET 1.1 auf 2.0.

# 2 Konfigurative Referenzmodellierung mit dem H2-Toolset

Die am Lehrstuhl für Wirtschaftsinformatik und Informationsmanagement der Westfälischen Wilhelms-Universität Münster entwickelte Software H2 ist ein Meta-Modellierungswerkzeug, mit dem sich hierarchisch strukturierte Modellierungssprachen definieren und diesen Sprachdefinitionen entsprechende Modelle konstruieren lassen.[11] Das Ziel des Projektseminars COIN (COnfigurative INformation Modeling) bestand darin, das H2-Toolset um Methoden der konfigurativen Referenzmodellierung zu erweitern. An der Realisation dieses Projekts waren neun studentische Teilnehmer sowie drei Betreuer im Zeitraum von Oktober 2005 bis Mai 2006 beteiligt. Diese Arbeit dokumentiert die durchgeführten Arbeitsschritte.

Die Aufgabenstellung des Seminars gliedert die Dokumentation nach den zur Erweiterung des Toolsets notwendigen Funktionen. Zunächst werden die Mechanismen der konfigurativen Referenzmodellierung fachkonzeptionell vorgestellt, um eine theoretische Grundlage für die implementierten Erweiterungen zu schaffen.

Ausgangs- und Bezugspunkt dieser Dokumentation ist ein im Folgenden vorgestellter Workflow zur Referenzmodellierung, der die Grundlage für die praktische Erweiterung der Software darstellt. Um die Verständlichkeit dieser Erweiterungen zu gewährleisten, wird der Workflow sukzessive anhand der zugrunde liegenden Literatur hergeleitet, um so Nutzen- und Anwendungspotenziale der konfigurativen Referenzmodellierung – und somit auch einer werkzeugunterstützten Nutzung dieser – zu motivieren.

Im folgenden Kapitel werden kurz die Vorarbeiten am H2-Toolset vorgestellt. Neben der Darstellung von Definitionen und Grundfunktionalität wird die Erzeugung von Objekten näher beschrieben. Dies beinhaltet sowohl die Beschreibung der Metamodellierungs- als auch der Modellierungskomponente. Die Darstellung ist fachkonzeptionell, enthält aber auch Aspekte des DV-Konzepts und der eigentlich Implementierung. Kapitel 2 wird abgeschlossen durch die Darstellung der notwendigen konzeptionellen Erweiterungen für die Implementierung der Mechanismen der konfigurativen Referenzmodellierung.

## 2.1 Konfiguration von Referenzmodellen

Konfigurierbare Referenzmodelle enthalten Regeln, nach denen sie an spezifische Bedürfnisse eines Unternehmens anzupassen sind. Diese Regeln ermöglichen eine automatische Adaption über verschiedene Mechanismen, die den erforderlichen Anpassungsaufwand minimie-

[11] Vgl. Becker et al. (2005) und Becker et al. (2006).

ren. Die zur konfigurativen Referenzmodellierung notwendigen Arbeitsschritte lassen sich grob anhand eines Workflows zusammenfassen (vgl. Abb. 2).

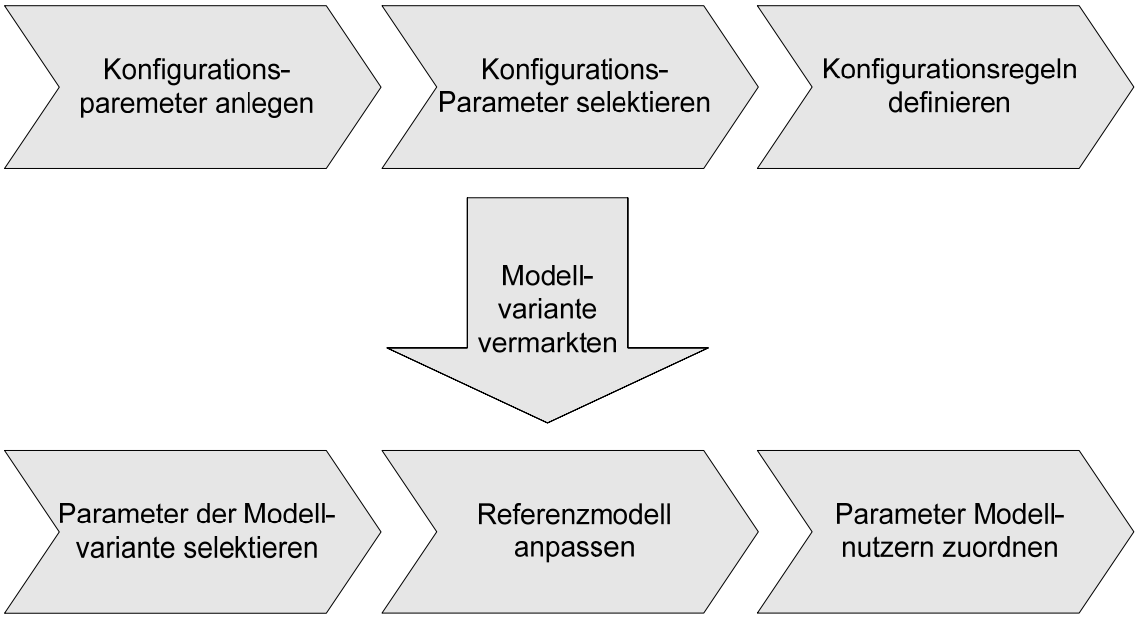



**Abb. 2:** Workflow der konfigurativen Referenzmodellierung im H2-Toolset

Die im Folgenden beschriebenen, den Workflow charakterisierenden Prozesse gliedern die weiteren Inhalte dieses Kapitels.

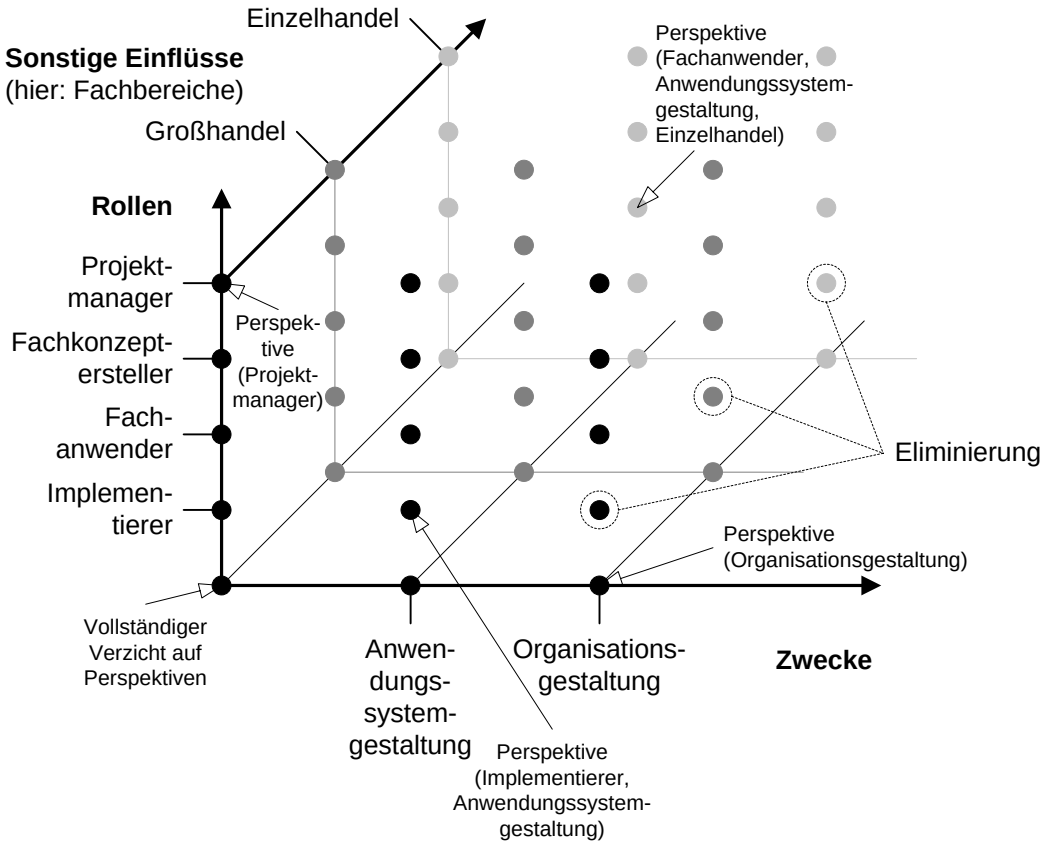
*Konfigurationsparameter anlegen*

Unter Konfigurationsparametern werden jene Eigenschaften von Unternehmen bzw. Modellierern verstanden, anhand derer ein Referenzmodell konfiguriert werden kann. Es eignen sich *Unternehmensmerkmale* sowie ihre verschiedenen Ausprägungen oder *Perspektiven*. Erstere können in Form morphologischer Kästen konkretisiert werden, die in den Zeilen einem Unternehmensmerkmal seine möglichen Ausprägungen zuordnen (vgl. Tab. 1).

| **Unternehmensmerkmal** | **Unternehmensmerkmalsausprägung** | | |
|---|---|---|---|
| **Geschäftsart** | Lagergeschäft | Streckengeschäft | Zentralregulierung |
| **Bezugsraum** | international | | national |
| **Träger der Nutzung** | Investitionsgüterhandel | | Konsumgüterhandel |
| **Berichtszyklus** | Wöchentlich | Monatlich | Jährlich |

**Tab. 1:** Beispielhafter morphologischer Kasten

Perspektiven als Konfigurationsparameter repräsentieren die verschiedenen Sichtweisen auf das Gesamtmodell, die sich aus den unterschiedlichen Verwendungskontexten des Referenzmodells ergeben.[12] Beispiele hierfür sind verschiedene *Rollen* wie Fachkonzeptersteller oder Implementierer, *Zwecke* wie Organisations- oder Anwendungssystemgestaltung oder auch *sonstige weitere Einflüsse* (Vgl. Abb. 3).




Quelle: Becker et al. (2002), S. 40

**Abb. 3:** Perspektivenwahl zur Konfiguration von Referenzmodellen

Andere Konfigurationsparameter (Context Driver) liefert bspw. die Standard Core Components Technical Specification (CCTS).[13]

Eine Software zur Konfiguration von Referenzmodellen muss das Anlegen und Pflegen von Konfigurationsparametern ermöglichen. Neben diesen inhaltlichen Komponenten erscheint auch die Integration von Ausschluss- und Bedingungsbeziehungen zwischen den Konfigurationsparametern sinnvoll. Schließen sich einzelne Konfigurationsparameter aus oder bedingen einander, dürfen sie bei der Konfiguration nicht bzw. nur gemeinsam selektiert werden. Ab-

[12] Vgl. Becker et al. (2002), S. 28.
[13] Vgl. Crawford (2003).

schnitt 3.3 stellt die Erweiterung des Toolsets um diese Funktionalitäten detailliert dar, Kap. 3.3.2 beschreibt die Implementierung dieser Funktion.

*Konfigurationsparameter selektieren*

Im Anschluss an die Definition genereller Konfigurationsparameter sind für das Referenzmodell relevante Parameter zu selektieren. Somit muss es dem Modellierer möglich sein, eine Menge von Konfigurationsparametern anzulegen, um aus dieser wiederum eine Teilmenge auszuwählen. Diese Teilmenge enthält jene Konfigurationsparameter, welche er für die Konfiguration des Referenzmodells als entscheidend erachtet. Die Implementierung dieser als „Preselection“ bezeichneten Funktionalität ist in Kap. 3.3.2 beschrieben.

*Konfigurationsregeln definieren*

Die für das Referenzmodell als relevant deklarierten Konfigurationsparameterausprägungen müssen mit den Modellelementen in Beziehung gesetzt werden, um eine Konfiguration des Modells zu ermöglichen. Einzelne Modellelemente sind somit nur für bestimmte Kombinationen von Konfigurationsparametern sichtbar.

| **Bedingungen** | | | **Aktionen** | | | | | |
|---|---|---|---|---|---|---|---|---|
| Geschäftsart | Bezugsraum | Berichtszyklus | Artikelgruppe Warengruppen | Artikelgruppe Herkunftsland | Artikelgruppe Aktionsart | Wertansatz nicht aggregiert | Taggruppe Woche | Taggruppe Monat |
| [Lagergeschäft \| Lagergeschäft + Streckengeschäft] | [international\| international + regional] | [Woche] | | | | | | |
| | | [Monat] | | | | | | |
| | | [Woche+Monat] | | | | | | |
| | [regional] | [Woche] | | | | | | |
| | | [Monat] | | | | | | |
| | | [Woche+Monat] | | | | | | |
| [Aktionsgeschäft \| Lagergeschäft + Aktionsgeschäft \| Lagergeschäft + Aktionsgeschäft + Streckengeschäft \| Aktionsgeschäft + Streckengeschäft] | [international \| international + regional] | [Woche] | | | | | | |
| | | [Monat] | | | | | | |
| | | [Woche+Monat] | | | | | | |
| | [regional] | [Woche] | | | | | | |
| | | [Monat] | | | | | | |
| | | [Woche+Monat] | | | | | | |

Quelle: Becker, Knackstedt (2004), S. 44

**Tab. 2:** Beispielhafte Entscheidungstabelle

Die Gültigkeit verschiedener Modellelemente in Abhängigkeit unterschiedlicher Konfigurationsparameterausprägungskombinationen kann mit Hilfe von Entscheidungstabellen dargestellt werden (vgl. Tab. 2). Mit dem Term-Editor in Abschnitt 3.4 und dem Configuration Customizer in Abschnitt 3.5 und 3.6 wird die Implementierung und Nutzung der für die Konfiguration zentralen Funktionen der Erstellung von Termen (welche die Gültigkeit von Modellelementen in Abhängigkeit bestimmter Konfigurationsparameter festlegen) und der Zuweisung von Termen zu Modellelementen vorgestellt.

*Modellvariante vermarkten*

Der ökonomische Nutzen konfigurativer Referenzmodellierung besteht u. a. in der Adressierung eines möglichst großen Marktes zum Vertrieb des Modells. Hat der Referenzmodellersteller ein Modell sowie geeignete Konfigurationsparameter angelegt, kann er durch Selektion jener und entsprechender Konfiguration des Modells eine Modellvariante erstellen und vermarkten. Kap. 3.3.2 hat die Konfiguration von Referenzmodellen zum Gegenstand und zeigt somit die Möglichkeit der Konzeption von Modellvarianten auf.

*Parameter der Modellvariante selektieren*

Die Modellvarianten genügen in der Regel noch keinen unternehmensspezifischen Informationsräumen. Erwirbt ein Unternehmen eine Modellvariante, bildet diese das Unternehmen zunächst nur grob ab – eine weitergehende Adaption der Variante ist notwendig. Hierzu kann ein Modellierer des Unternehmens weitere unternehmensspezifische Konfigurationsparameter selektieren, um das Modell noch weiter anzupassen. Dieser Prozess unterscheidet sich in seiner Implementierung nicht vom vorangegangenen – schließlich gilt es dieselbe Funktion zu erfüllen.

*Referenzmodell anpassen*

Der Referenzmodellersteller kann mit Hilfe von Konfigurationsparametern den Adaptionsaufwand gering halten – sämtliche unternehmensindividuelle Gegebenheiten lassen sich durch deren Anlage und Selektion jedoch nicht berücksichtigen. Der Modellierer muss in der Lage sein, nachträglich, d. h. nach Auswahl relevanter Konfigurationsparameter, weitere Änderungen am Modell vorzunehmen.

*Parameter Modellnutzern zuordnen*

Im letzten Prozess des Workflows können verschiedenen Anwendern Konfigurationsparameter zugeordnet werden. Auf diese Weise kann auf die verschiedenen Bedürfnisse und Sichtweisen einzelner Modellnutzer Rücksicht genommen werden. Kap. 3.3.2 und 3.8 konkretisieren diese Funktionalität des Toolsets.

## 2.2 Basiskomponenten des H2-Toolsets

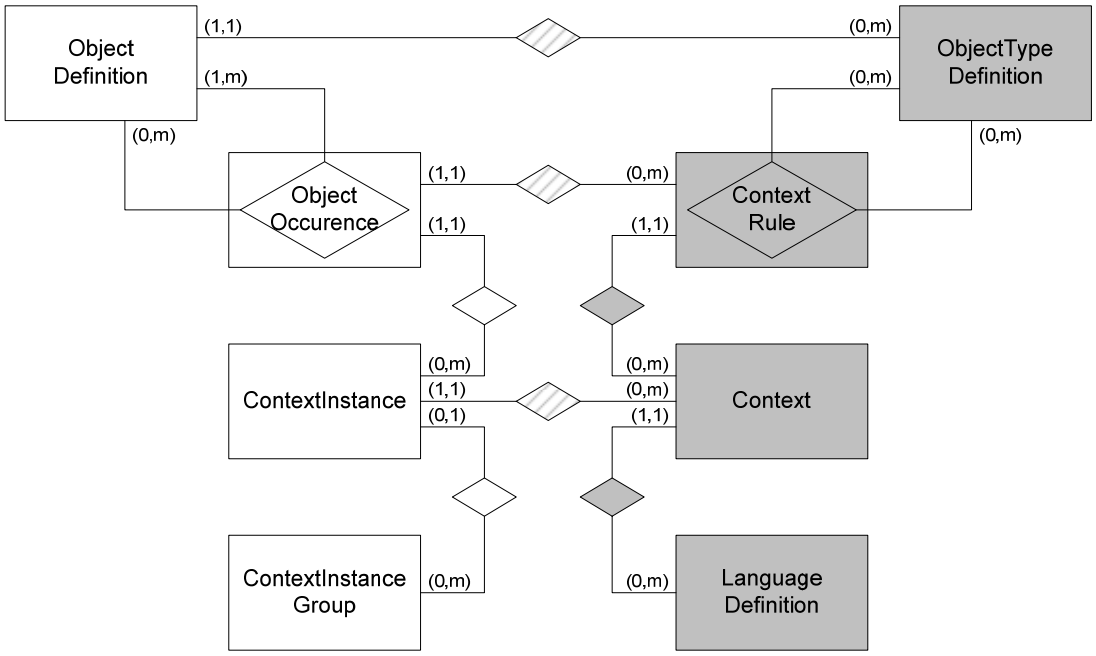
### 2.2.1 Übersicht

Das H2-Toolset ermöglicht die Erstellung neuer hierarchischer Modellierungssprachen bzw. die Modifikation existierender Sprachen. Mit dem Language-Editor werden Objekttypdefinitionen (ObjectTypeDefinition), deren Zuordnung zu Kontexten (Context) und Modellierungssprachen (LanguageDefinition) sowie die Definition von Verknüpfungsregeln zwischen Objekttypdefinitionen (ContextRules) vorgenommen. Ferner werden im Language-Editor Attributgruppen definiert und diese Objekttypdefinitionen zugeordnet. Die Erstellung einer Projektdatenbank erfordert zwingend die Angabe einer Sprachdefinition, der die Elemente der Modellierungssprache sowie die in dieser Sprache erlaubten Verknüpfungen definiert. Eine Kopie des ContextType wird beim Erstellen in der jeweiligen Modelldatenbank gespeichert. Jede Operation auf Objekten in den Editoren wird auf Übereinstimmung mit den Regeln geprüft werden. Gegebenenfalls wird die Operation verweigert.

Entsprechend der Sprachdefinition werden für ein konkretes Modell Objekte angelegt. Die Konstrukte entsprechen der Sprachdefinition. Objektdefinitionen (ObjectDefinition) basieren immer auf einer Objekttypdefinition. Die Beziehung zweier konkreter Objektdefinitionen wird über ihre Objektausprägung (ObjectOccurrence) definiert, die jeweils Teil einer Kontextinstanz (ContextInstance) sind. Kontextinstanzen können zu einer Kontextinstanzgruppe (ContextInstanceGroup) zusammengefasst werden.

Abb. 4 zeigt die Kernbestandteile des Repositories, die im folgenden Kapitel am Beispiel der Modellierungssprache MetaMIS[14] näher erläutert werden.

[14] Vgl. Holten (2003).




Quelle: Becker et al. (2006), S. 8

**Abb. 4:** Kern-Repository des H2-Toolset

### 2.2.2 Grundlagen des Meta-Meta-Modells des H2-Toolset

Eine ObjectTypeDefinition repräsentiert ein Objekt der Modellierungssprache. Dabei kann es sich sowohl um ein reguläres Modellierungskonstrukt handeln (bspw. Dimension, Bezugsobjekt, Dimensionsgruppe, Dimensionsausschnitt, Dimensionsausschnittskombination etc.) als auch um Objekte, die keinen direkten Bezug zur Modellierungssprache haben (wie bspw. Ordner).

Eine *ContextRule* definiert Verknüpfungsregeln zwischen genau zwei ObjectTypeDefinitions. Dabei ist zwischen Erzeuger-Regeln und anderen Regeln zu unterscheiden. Die Anwendung von Erzeuger-Regeln steuert die Erstellung von Objektdefinitionen.

Ein *Context* gruppiert mehrere ContextRules zu einem in sich geschlossenen Regelblock. Insofern beschreibt eine Kontext-Instanz eine konkrete Struktur aus Objekten (bspw. im Kontext „Dimension“ die Instanz „Dimension Zeit nach KW“). In einer Kontext-Instanz können Ausprägungen von Objekt-Definitionen unter einem Elternelement jeweils nur einmal vorkommen. Grundsätzlich können Objekte in verschiedenen Instanzen eines Kontextes jedoch mehrfach vorkommen (Bsp.: Der Tag „2004-01-12“ kommt in den Instanzen „Zeit nach KW“ und „Zeit nach Monat“ des Kontextes „Dimension“ vor). Für Kontexte können gewisse Regeln hinsichtlich des Aufbaus der Strukturen definiert werden. Einige dieser Regeln beziehen

sich jeweils auf so genannte ContextInstanceGroups. ContextInstanceGroups ermöglichen eine weitere logische Unterteilung von Kontexten auf Instanzebene. Beispielsweise können zwei Zeitdimensionen in einer ContextInstanceGroup zusammengefasst werden und es kann festgelegt werden, dass diese identische Blattmengen besitzen müssen.

Folgende Eigenschaften können für einzelne Kontexte festgelegt werden:

Es muss grundsätzlich unterschieden werden zwischen definierenden/nicht-definierenden und definierten/nicht-definierten Kontexten. Definierende Kontexte sind diejenigen, die die rechte Regelseite von ContextRules in definierten Kontexten näher beschreiben. Das heißt, der Root-ObjectType des definierenden Kontextes muss dem ObjectType der rechten Regelseite der entsprechenden Regel im definierten Kontext entsprechen (Bsp.: Dimensionsgruppe bestehen aus Dimensionen, welche im Kontext Dimension durch Dimensionsobjekte aufgespannt werden). Nicht-definierend sind dementsprechend diejenigen Kontexte, die keine Regeln in anderen Kontexten näher beschreiben; nicht-definiert sind die Kontexte, dessen Regeln nicht durch andere Kontexte näher beschrieben werden.

Eine *LanguageDefinition* repräsentiert eine Gruppierung mehrerer Kontexte, die eine konsistente Sprachdefinition darstellen.

Zusammenfassendes Beispiel: Eine Modellierungssprache besteht aus den ObjectTypeDefinitions Dimension (D) und Dimensionsobjekten (DO). Eine erste ContextRule legt fest, dass die Objekte des Typs D dem eindeutigen Root-Element zugeordnet werden. Auf diese Weise wird sichergestellt, dass Objekte des Typs D die Root-Elemente des Kontextes sind. Eine zweite ContextRule besagt, dass einem Objekt des Typs D beliebig viele DO’s zugeordnet werden können. Eine dritte ContextRule besagt, dass einem DO beliebig viele DO’s zugeordnet werden dürfen (zur Abbildung der hierarchischen Struktur der DO’s). Diese drei ContextRules werden zum Kontext „Dimension“ verknüpft. Ein ContextType „Mini H2“ bezeichnet eine Sprache, die beliebige Modelle (konstruiert aus Instanzen obiger ObjectTypeDefinitionss) unter Einhaltung der in den ContextRules abgebildeten Verknüpfungsregeln erzeugen kann. Abb. 5 zeigt ein einfaches Beispiel, welches zusätzlich Ordner enthält.

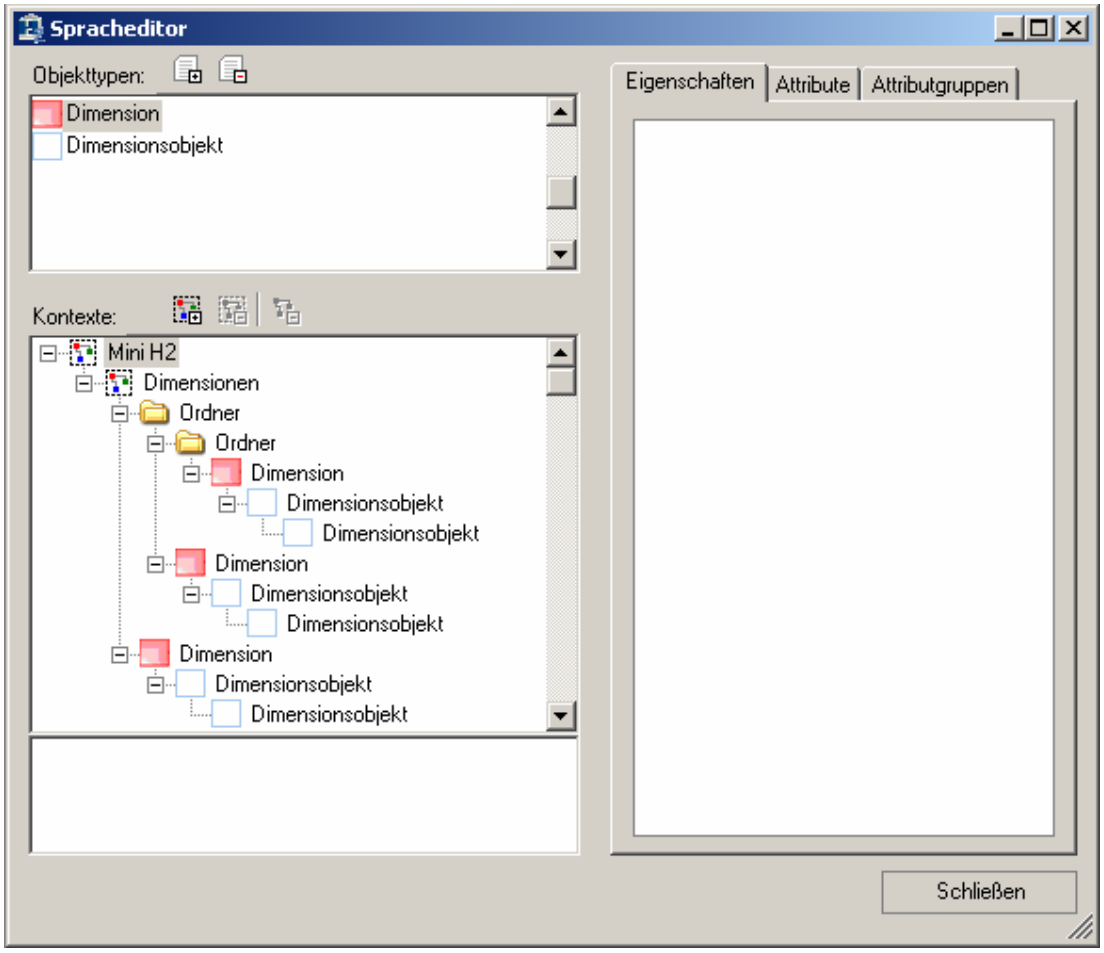



**Abb. 5:** Sprachdefinition „Mini H2“

### 2.2.3 Meta-Modellierung mit dem H2-Toolset

*Definition der ObjektTypeDefinitionss*

Ausgangspunkt für die Definition der Modellierungsprache ist die Definition der enthaltenen Sprachelemente bzw. ObjectTypeDefinitionss. Zu jeder ObjectTypeDefinition wird ein eindeutiger Name sowie ein Symbol gespeichert. Die ObjectTypes werden in einer unstrukturierten Liste gespeichert. Die nachträgliche Modifikationen oder Löschung von ObjectTypeDefinitions ist möglich, muss jedoch genau abgewägt werden, da sie Konsistenz der Sprache u.U. gefährdet. Die ObjectTypeDefinitionss werden im Konfigurationsrepository abgespeichert.

*Anlegen eines ContextType und Zuordnung von ObjektTypeDefinitions*

Mit der Zuweisung von ObjectTypeDefinitions zu Contexts wird definiert, welche Sprachelemente in dem jeweiligen ContextType zur Verfügung stehen. Im Konfigurationsrepository

können mehrere ContextTypes hinterlegt sein. Beim Erzeugen einer Sprache in einer Projektdatenbank ist mindestens ein ContextType notwendig.

*Definition von ContextRules*

Eine *ContextRule* definiert Verknüpfungsregeln zwischen genau zwei ObjectTypeDefinitions. Zur Speicherung einer Context Rule werden mehrere Attribute benötigt. Tab. 3 stellt diese Anhand der Beispielsprache MetaMIS dar. Tab. 4 enthält einer Übersicht der verwendeten Kontexte.

| **RuleID** | **OID1** | **OID2** | **Min/ Max** | | **Defined By Context** | **OnDrop** | **CID** | **Name** |
|---|---|---|---|---|---|---|---|---|
| 10 | ROOT | D | 0 | N | - | CD | A | |
| 20 | D | DO | 1 | N | - | CD | A | D-DRO Relation |
| 30 | DO | DO | 0 | N | - | CD | A | DRO-DRO Relation |
| 40 | ROOT | DG | 0 | N | - | CD | B | |
| 50 | DG | D | 1 | N | A | CO | B | DG-D Relation |
| 60 | ROOT | DS | 0 | N | - | CD | C | |
| 70 | DS | D | 1 | 1 | A | CDe / COe | C | DS-D Relation |
| 100 | ROOT | DSC | 0 | N | - | CD | D | |
| 110 | DSC | DS | 1 | N | C | CO | D | DSC-DS Relation |
| 120 | ROOT | IO | 0 | N | - | CD | F | |
| 130 | IO | DSC | 1 | 1 | D | CO | F | IO-DSC Relation |
| 140 | IO | RS | 1 | 1 | G | CO | F | IO-RS Relation |
| 150 | IO | FC | 1 | N | - | CO | F | IO-FC Relation |
| 160 | ROOT | FC | 0 | N | - | CD | E | |
| 170 | FC | FCE | 1 | 1 | - | CD | E | FC-FCE Relation |
| 180 | ROOT | RS | 0 | N | - | CD | G | |
| 190 | RS | R | 1 | N | - | CD | G | RS-R Relation |
| 200 | R | R | 0 | N | - | CD | G | R-R Relation |
| 210 | ROOT | F | 0 | N | - | CD | H | |
| 220 | F | IO | 1 | N | F | CO | H | F-IO Relation |
| 230 | F | F | 0 | N | - | CD | H | F-F Relation |
| 240 | ROOT | D | 0 | N | . | CO | I | |
| 250 | D | HL | 1 | 1 | - | CD | I | D-HL Relation |
| 260 | HL | HL | 0 | 1 | - | CD | I | HL-HL Relation |

**Tab. 3:** Tabelle ContextRules

| CID | Name | Instance-Leaves | Instance-Nodes | Area-Leaves | Area-Nodes | Binary-Tree | B-Tree |
|---|---|---|---|---|---|---|---|
| A | Dimension | disjunkt | disjunkt | Nicht disjunkt, total | Nicht disjunkt, partiell | | |
| B | DimensionGroup | Nicht disjunkt, total | Nicht disjunkt, total | | | | |
| C | DimensionScope | | | | | | |
| D | DimensionScope-Combination | | | | | | |
| E | FactCalculation | | | | | | |
| F | InformationObject | | | | | | |
| G | RatioSystem | | | | | | |
| H | Folder | | | | | | |
| I | HierarchyLevel | | | | | | |

**Tab. 4:** Tabelle Context

*Beschreibung der Tabellen*

RuleID ist Schlüssel der Tabelle ContextRule. Ein Eintrag in der Tabelle kann wie folgt interpretiert werden: Objekte vom Typ „OID1“ und Objekte vom Typ „OID2“ dürfen unter Beachtung der angegebenen Constraints miteinander verknüpft werden.

In den Spalten OID1 und OID2 werden die beiden ObjectTypes, deren Verknüpfung mit dieser Regel definiert wird, referenziert (als Fremdschlüssel zur Relation ObjectType, in den Beispielen vereinfachend über die Kürzel der ObjectTypes dargestellt). OID1 referenziert das Element, dem Unterelemente zugeordnet werden können. Um relevante Regeln der Verknüpfung eines Objekts zu identifizieren, müssen alle Regeln im aktuellen Kontext berücksichtigt werden, bei denen der Typ des Objektes in OID1 zu finden ist. Mithilfe der Attribute „Min“ und „Max“ wird die Kardinalität der Verknüpfung festgehalten. Die Angaben beziehen sich dabei auf vollständig modellierte und korrekte Objekte. Mithilfe dieser Kardinalität können unvollständig definierte Modellfragmente identifiziert werden. Im Attribut „DefinedByContext“ wird eine Referenz auf die Tabelle „Context“ abgespeichert, die die erlaubten Verknüpfungen der rechten Regelseite repräsentiert. Beispielsweise besagt das Attribut „DefinedBy-Context = A“ in der Regel mit ID 30, dass die Elemente, die unter dem Element vom Typ D hängen, nach den Regeln ID 10 und ID 20 konstruiert sein müssen. OnDrop steuert das Verhalten, wenn Objekte durch Drag & Drop anderen Objekten zugewiesen werden. Es werden drei Ausführungen unterschieden (je nachdem, ob ein neues Objekt definiert und seine Ausprägung angelegt werden soll oder ob nur eine Ausprägung angelegt werden muss oder ob eine Ausprägung angelegt werden und eine Struktur aus einem Kontext kopiert werden muss).

- CreateDefinition (CD): Notwendig, falls bei Zuweisung einer Struktur neue Objekte erzeugt werden sollen (bspw. der Fall bei Abbildung D und DO). In diesem Fall werden

neue Objektdefinitionen (mit entsprechenden Ausprägungen) erzeugt. In diesem Fall handelt es sich also um eine Erzeuger-Regel.

- CreateDefinitionExplicit (CDe): Notwendig, falls die Strukturen nach Zuweisung bearbeitet werden sollen. In diesem Fall muss die komplette Struktur bei Zuweisung unter der neuen RootID in der Relation ObjectOccurence abgelegt werden. Beispiel: Zuordnung einer D zu einer DS. Es findet ein Kopiervorgang statt. Die erzeugte ist gegenüber Änderungen im Originalkontext robust und kann nach belieben editiert werden.

- CreateOccurence (CO): Das Root-Element der zugewiesenen Struktur wird dem neuen Vaterelement zugewiesen (z. B. D an DG), da die Struktur nach Zuweisung nicht mehr editierbar sein muss.

- CreateOccurence (COe): Das Root-Element der zugewiesenen Struktur wird dem neuen Vaterelement zugewiesen (z. B. D an DS). Die Struktur soll nach Zuweisung im Gegensatz zu CDe nicht starr sein, sondern immer einen Ausschnitt den aktuellen Stand wiedergeben. D. h. entweder über Ausschluss oder durch explizite Nennung bestimmter Teile der Originalstruktur wird nur ein Teil der Kind-Knoten des angehängten Vater-Knoten dargestellt.

Das Attribut CID ist Fremdschlüssel aus der Tabelle „Context“.

Ein Kontext und eine ContextRule im Konfigurationsrepository kann geändert und (unwiederbringlich) gelöscht werden. Es ist zu prüfen, ob noch Modellelemente existieren, die auf dieser ContextRule basieren.

*Hierarchiestufen*

- Abbildung von Hierarchiestufen: Hierarchiestufen werden durch ein weiteres Attribut (Depth) in der Tabelle *ObjectOccurence* abgebildet, bei jedem Objekt wird also in einer zusätzlichen Spalte die Tiefe im jeweiligen „Kontext“ angegeben.

- Anzeige und Auflösung von Hierarchiestufen: Im Editor werden für alle Objekte Hierarchiestufen angezeigt. Können Hierarchiestufen nicht aufgelöst werden (keine entsprechende HL Zuweisung in der Tabelle *ObjectOccurence*) wird der numerische Wert angezeigt.

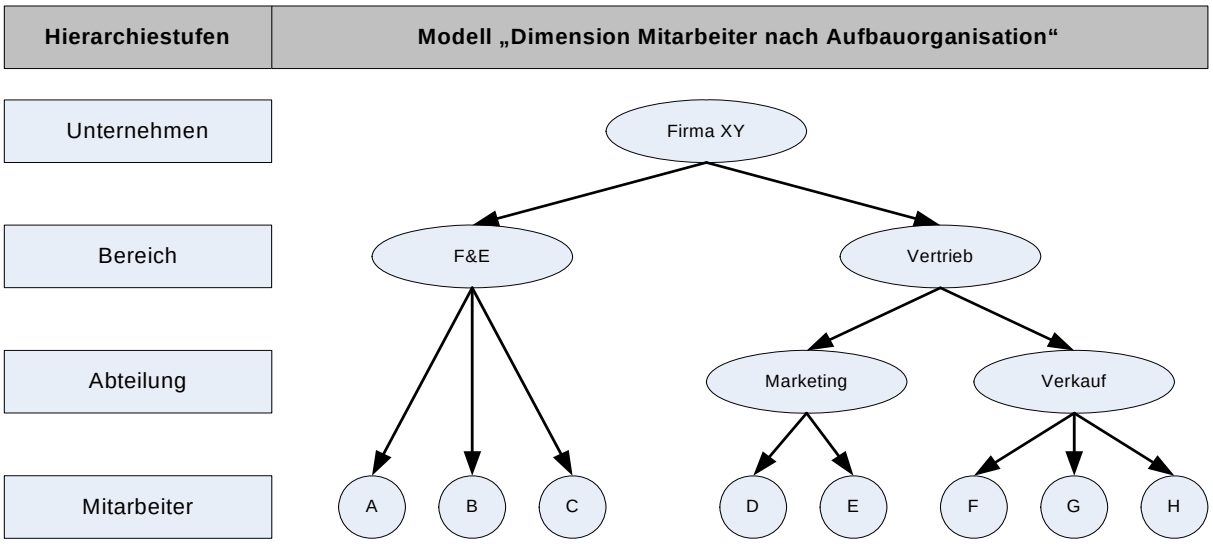



**Abb. 6:** Darstellung von Hierarchiestufen

Das obige Beispiel (Abb. 6) wird folgendermaßen in den Tabellen *ObjectDefinition* (Tab. 5) und *ObjectOccurrence* (Tab. 6) gespeichert:

| **OID** | **Name** | **OTID** |
|---|---|---|
| 0 | Mitarbeiter nach Abteilung | D |
| 10 | Firma XY | DO |
| 20 | F&E | DO |
| 30 | Vertrieb | DO |
| 40 | Marketing | DO |
| 50 | Verkauf | DO |
| 60 | A | DO |
| 70 | B | DO |
| 80 | C | DO |
| 90 | D | DO |
| 100 | E | DO |
| 110 | F | DO |
| 120 | G | DO |
| 130 | H | DO |
| 200 | Unternehmen | HL |
| 210 | Bereich | HL |
| 220 | Abteilung | HL |
| 230 | Mitarbeiter | HL |
| 500 | Dimensionsausschnitt „F&E“ | DS |

**Tab. 5:** Tabelle ObjectDefinition

| OID1 | OID2 | Depth | TrackingNo | Path | RuleID | ContextInstanceID |
|---|---|---|---|---|---|---|
| 0 | Null | 0 | | | | |
| 10 | 0 | 1 | | | | |
| 20 | 10 | 2 | | | | |
| 30 | 10 | 2 | | | | |
| 40 | 30 | 3 | | | | |
| 50 | 30 | 3 | | | | |
| 60 | 20 | 4 | | | | |
| 70 | 20 | 4 | | | | |
| 80 | 20 | 4 | | | | |
| 90 | 40 | 4 | | | | |
| 100 | 40 | 4 | | | | |
| 110 | 50 | 4 | | | | |
| 120 | 50 | 4 | | | | |
| 130 | 50 | 4 | | | | |
| 200 | Null | 1 | | | | |
| 210 | 200 | 2 | | | | |
| 220 | 210 | 3 | | | | |
| 230 | 220 | 4 | | | | |
| 500 | Null | 0 | | | | |
| 20 | 500 | 2 | | | | |
| 60 | 20 | 4 | | | | |
| 70 | 20 | 4 | | | | |
| 80 | 20 | 4 | | | | |
| 200 | Null | 1 | | | | |
| 210 | 200 | 2 | | | | |
| 220 | 210 | 3 | | | | |
| 230 | 220 | 4 | | | | |

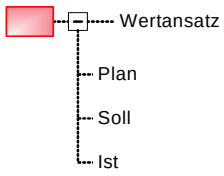
**Tab. 6:** Tabelle ObjectOccurence

Beim Zuweisen von Objekten an andere Objekte muss prinzipiell folgendes überprüft werden: Existiert ein Eintrag eines Objektes vom Typ „HierarchyLevel“ mit identischer RootID und Depth in der Tabelle *ObjectOccurence*, so ist dieser ebenfalls in der neu entstandenen Struktur mit neuer RootID.

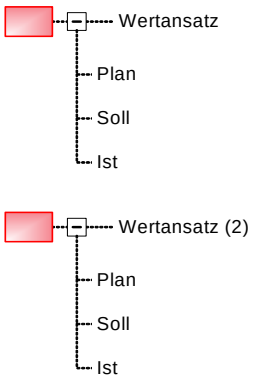
*Abbildung von „Varianten“ / Definitionskopien*

In jedem Editor können Varianten einer Objektausprägung erzeugt werden (unter Berücksichtung der diesem Objekt zugewiesenen Unterobjekte). Eine eigene Variantenverwaltung wird nicht realisiert. Stattdessen können Kopien existierender Objektausprägungen erzeugt werden (beschränkt auf den jeweiligen Editor des Objekttyps). Ergebnis ist eine neue Objektdefinition und die dazugehörige Objektausprägung.

**Modell(e) vor Änderung**




**Modell(e) nach Änderung**




**Tabellenschema vor Änderung**

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Wertansatz | D |
| 11 | Plan | DRO |
| 12 | Soll | DRO |
| 13 | Ist | DRO |
| 14 | Wertansatz | HL |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 10 | 10 | 1 |
| 14 | null | 10 | 1 |

**Tabellenschema nach Änderung**

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Wertansatz | D |
| 11 | Plan | DRO |
| 12 | Soll | DRO |
| 13 | Ist | DRO |
| 14 | Wertansatz | HL |
| 20 | Wertansatz (2) | D |
| 21 | Plan | DRO |
| 22 | Soll | DRO |
| 23 | Ist | DRO |
| 24 | Wertansatz | HL |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 10 | 10 | 1 |
| 14 | null | 10 | 1 |
| 20 | null | 20 | 0 |
| 21 | 20 | 20 | 1 |
| 22 | 20 | 20 | 1 |
| 23 | 20 | 20 | 1 |
| 24 | null | 20 | 1 |

**Abb. 7:** Erzeugung von Objektkopien

*Action Rules*

Zusätzlich werden ActionRules definiert, die als Relation zwischen ObjectTypes und ContextRules (z. B. bei der Neuanlage von Dimensionen) „eventbezogen“ greifen. Beispiele: Neuanlage einer Dimension: zuerst müssen die Hierarchiestufen bestimmt werden. Über ActionRuleLinkage können diese ActionRules verknüpft werden. Somit ist es möglich, logische Verknüpfungen (und, oder, xor,…) abzubilden. Beispiel: Bei DO muss die Regel 10 XOR 20 erfolgen (entweder das DO hängt unter einem DO XOR D).

Tupel in der Tabelle *ActionRule* steuern die Ausführung von ContextRules auf bestimmte Events. *ActionRuleLinkage* ermöglicht die Verknüpfung mehrerer ActionRules (durch Anwendung der boolschen Algebra).

| OTID | ContextRuleID | ActionRuleID | ActionRuleID | Action | Operator |
|---|---|---|---|---|---|
| D | 250 | | | OnCreate | |
| | | | | OnCreate | |
| | | 250 | 260 | OnCreate | XOR |

**Tab. 7:** Relationstabelle ActionRule

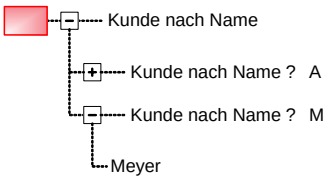
### 2.2.4 Modellierung mit dem H2-Toolset

*Anlegen von Objektdefinitionen*

Die Erzeugung von Objekten kann im *Model Editor* vorgenommen werden. Das Erzeugen von Objekten in einem anderen als dem dafür vorgesehenen Kontext wird prinzipiell unterbunden.

Beim Anlegen eines Objekts wird ein entsprechender Eintrag in der Tabelle *ObjectDefinition* vorgenommen. Anschließend wird eine Verknüpfung mit dem Vaterobjekt in der Tabelle *ObjectOccurence* angelegt (bei Root-Objekten erfolgt die Zuweisung des „Vaters“ durch den Eintrag „Null“).

**Erzeugtes Modell**




**Tabellschema nach Erzeugung**

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Kunde nach Name | D |
| 11 | Kunde nach Name -> A | DRO |
| 12 | Kunde nach Name -> M | DRO |
| 13 | Meyer | DRO |
| 14 | Nachname | HL |
| 15 | Kunde | HL |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 12 | 10 | 2 |
| 14 | null | 10 | 1 |
| 15 | 14 | 10 | 2 |

**Abb. 8:** Erzeugung von Objekten

*Ändern von Objektdefinitionen*

Wird bei einer Objektausprägung z. B. der Name geändert, so ist die entsprechende Objektdefinition in der Tabelle *ObjectDefinition* zu ändern (andere existierende Ausprägungen dieser Definition bekommen die Namensänderung durch Referenzierung auf die Definition automatisch mit).

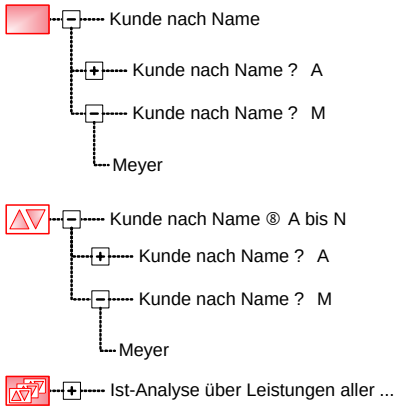
*Erzeugen von Objektausprägungen*

Die Ausprägung eines Objektes repräsentiert einen Eintrag in der Tabelle *ObjectOccurence* mit einem Verweis auf die Objektdefinition in *ObjectDefinition* (Datenbanktechnisch wird in diesen Fällen grundsätzlich eine Referenz erzeugt, die auf das Originalobjekt verweist). Eine Objektausprägung wird erzeugt, sobald Objekte erzeugt oder einem anderen Objekt zugeordnet werden (per Drag & Drop). Grundlage ist dabei das OnDrop-Attribut der entsprechenden ContextRule. Hat dieses den Wert CD (Create Definition), handelt es sich um die Erzeuger-Regel eines Objekt-Typen und eine entsprechende Objektdefinition muss erzeugt werden, sofern diese noch nicht existiert (unter Umständen kann schon eine existieren, da Objekte grundsätzlich mehrfach in einem Kontext vorkommen können).

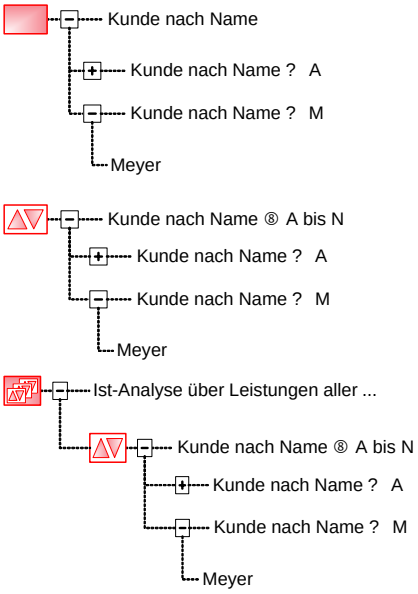
Konventionen für die folgenden Beispiele: Es existieren zwei Objekte „O1“ und „O2“ unterschiedlichen Typs, zumindest O2 sind dabei Subobjekte zugeordnet. Das Objekt O2 soll dem Objekt O1 zugeordnet werden (per Drag & Drop). Nun ist zu prüfen, welche Verknüpfungsregel für diese Aktion zutrifft (vgl. ContextRules). Die entsprechende Regel wird anhand des Kontextes ausgewählt, in den eingefügt werden soll. Es sind zwei Fälle zu unterscheiden:

- Fall 1: Die Verknüpfungsregel besagt, dass die untergeordnete Struktur des Objektes O2 nach Verknüpfung mit O1 nicht mehr geändert werden kann (über das Attribut OnDrop). Die Originalstruktur bzw. die Objektdefinition kann jedoch weiterhin geändert werden. In diesem Fall ist in der Tabelle *ObjectOccurence* lediglich ein Eintrag anzulegen, der das Objekt O2 mit dem Vaterobjekt O1 verknüpft. Diese Verknüpfung repräsentiert eine Objektausprägung von O2 im Kontext von O1. Der Rest der Hierarchie kann aus der Originalstruktur rekonstruiert werden (im folgenden Beispiel die Einträge 11, 12 und 13 mit der RootID 20 in der Tabelle *ObjectOccurence*).

**Modell(e) vor Änderung**




**Modell(e) nach Änderung**




**Tabellenschema vor Änderung**

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Kunde nach Name | D |
| 11 | Kunde nach Name --> A | DRO |
| 12 | Kunde nach Name --> M | DRO |
| 13 | Meyer | DRO |
| 14 | Nachname | HL |
| 15 | Kunde | HL |
| 20 | Kunde nach Name --> A bis N | DS |
| 30 | Ist-Analyse über Leistungen aller ... | DSC |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 12 | 10 | 2 |
| 14 | null | 10 | 1 |
| 15 | 14 | 10 | 2 |
| 20 | null | 20 | 0 |
| 11 | 20 | 20 | 1 |
| 12 | 20 | 20 | 1 |
| 13 | 12 | 20 | 2 |
| 14 | null | 20 | 1 |
| 15 | 14 | 20 | 2 |
| 30 | null | 30 | 0 |

**Tabellenschema nach Änderung**

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Kunde nach Name | D |
| 11 | Kunde nach Name --> A | DRO |
| 12 | Kunde nach Name --> M | DRO |
| 13 | Meyer | DRO |
| 14 | Nachname | HL |
| 15 | Kunde | HL |
| 20 | Kunde nach Name --> A bis N | DS |
| 30 | Ist-Analyse über Leistungen aller ... | DSC |

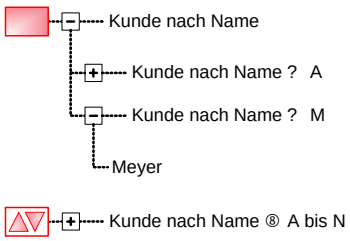
**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 12 | 10 | 2 |
| 14 | null | 10 | 1 |
| 15 | 14 | 10 | 2 |
| 20 | null | 20 | 0 |
| 11 | 20 | 20 | 1 |
| 12 | 20 | 20 | 1 |
| 13 | 12 | 20 | 2 |
| 14 | null | 20 | 1 |
| 15 | 14 | 20 | 2 |
| 30 | null | 30 | 0 |
| 20 | 30 | 30 | 1 |

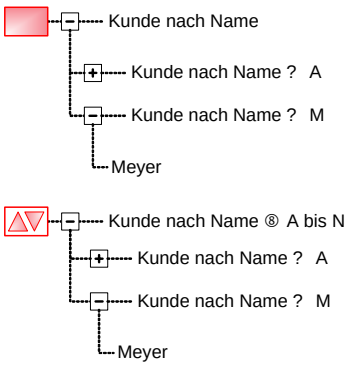
**Abb. 9:** Erzeugen von Objektausprägungen

- Fall 2: Legt die ContextRule fest, dass die Struktur nach Verknüpfung mit dem Objekt O1 weiterhin editiert werden darf (Ändern von Objektausprägungen) muss zwangsläufig die gesamte Hierarchie/Struktur von O2 im Kontext von O1 in der Tabelle *ObjectOccurence* abgespeichert werden, da sonst keine Möglichkeit besteht die Struktur zu rekonstruieren. Bei der Modellierungssprache MetaMIS trifft dieser Fall z. B. für Verknüpfung von Objekten der Typen DS und D zu.

**Modell(e) vor Änderung**




**Modell(e) nach Änderung**




**Tabellenschema vor Änderung**

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Kunde nach Name | D |
| 11 | Kunde nach Name --> A | DRO |
| 12 | Kunde nach Name --> M | DRO |
| 13 | Meyer | DRO |
| 14 | Nachname | HL |
| 15 | Kunde | HL |
| 20 | Kunde nach Name --> A bis N | DS |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 12 | 10 | 2 |
| 14 | null | 10 | 1 |
| 15 | 14 | 10 | 2 |
| 20 | null | 20 | 0 |

**Tabellenschema nach Änderung**

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Kunde nach Name | D |
| 11 | Kunde nach Name --> A | DRO |
| 12 | Kunde nach Name --> M | DRO |
| 13 | Meyer | DRO |
| 14 | Nachname | HL |
| 15 | Kunde | HL |
| 20 | Kunde nach Name --> A bis N | DS |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 12 | 10 | 2 |
| 14 | null | 10 | 1 |
| 15 | 14 | 10 | 2 |
| 20 | null | 20 | 0 |
| 11 | 20 | 20 | 1 |
| 12 | 20 | 20 | 1 |
| 13 | 12 | 20 | 2 |
| 14 | null | 20 | 1 |
| 15 | 14 | 20 | 2 |

**Abb. 10:** Erzeugen von Objektausprägungen mit kompletter Struktur

*Ändern/Löschen von Objektausprägungen*

Beim Löschen von Objektausprägungen sind zwei Fälle zu unterscheiden:

- Fall 1: Alle untergeordneten Objektausprägungen werden ebenfalls gelöscht. Dieser Vorgang entspricht dem Löschen eines kompletten Teilbaumes einer Struktur. Beispiel: Wird der Dimensionsausschnitt (vgl. Abb. 10 rechte Spalte) geändert (Löschen des Teilbaumes „Kunde nach Name → M“) sind in der Tabelle *ObjectOccurence* die Tupel (12,20, 20, 1) und (13, 12, 20, 2) zu löschen.

- Fall 2: In diesem Fall werden alle Objektausprägungen einer Hierarchiestufe entfernt, um die Konsistenz des entstehenden Fragments sicherzustellen. Der Löschvorgang erfolgt nach folgendem Schema:

- Ermittlung des zu betrachtenden Kontextes: Es werden nur Objektausprägungen bearbeitet, die eine identische RootID haben.
- Ermittlung der OID2 der zu löschenden Objektausprägung (wahlweise geht das auch über die Depth (alle Tupel mit der identischen Depth)).
- Löschen aller Objektausprägungen mit dieser OID2 und dieser RootID.
- Gleichzeitig werden alle direkt unter einer zu löschenden Objektausprägung hängenden Objektausprägungen neu verknüpft: Sie erhalten als neue OID2 die in Schritt 2 ermittelte OID2.

Grundsätzlich ist zu berücksichtigen, ob in einem definierenden oder einem definierten Kontext gelöscht wird. Im ersten Fall ist zu überprüfen, ob das zu löschende Element in vom betrachteten Kontext definierten Kontexten verwendet wird. Die entsprechende Beziehung im definierten Kontext muss ebenfalls gelöscht werden. Hierbei sind ebenfalls die zwei oben beschriebenen Fällte zu berücksichtigen. Wird in einem definierten Kontext gelöscht, kann die Beziehung ohne weitere Überprüfung entfernt werden.

Soll eine Beziehung gelöscht werden, der eine Erzeuger-Regel zugrunde liegt, ist sicherzustellen dass durch das Löschen eines Objektes dieses in keiner weiteren Beziehung vorkommt. Nur dann kann die Objektdefinition gelöscht werden.

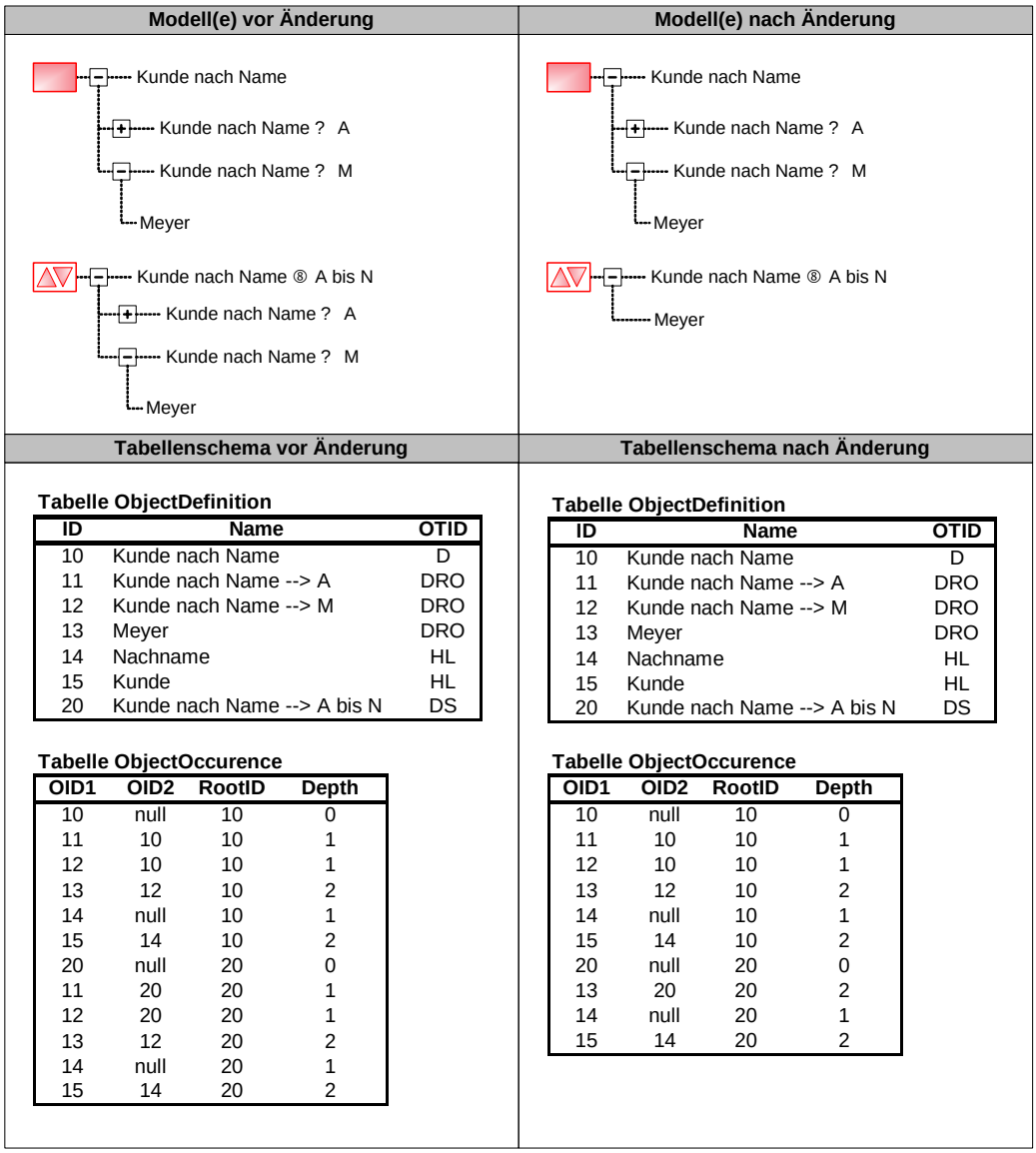



**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Kunde nach Name | D |
| 11 | Kunde nach Name --> A | DRO |
| 12 | Kunde nach Name --> M | DRO |
| 13 | Meyer | DRO |
| 14 | Nachname | HL |
| 15 | Kunde | HL |
| 20 | Kunde nach Name --> A bis N | DS |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 12 | 10 | 2 |
| 14 | null | 10 | 1 |
| 15 | 14 | 10 | 2 |
| 20 | null | 20 | 0 |
| 11 | 20 | 20 | 1 |
| 12 | 20 | 20 | 1 |
| 13 | 12 | 20 | 2 |
| 14 | null | 20 | 1 |
| 15 | 14 | 20 | 2 |

**Tabelle ObjectDefinition**

| ID | Name | OTID |
|---|---|---|
| 10 | Kunde nach Name | D |
| 11 | Kunde nach Name --> A | DRO |
| 12 | Kunde nach Name --> M | DRO |
| 13 | Meyer | DRO |
| 14 | Nachname | HL |
| 15 | Kunde | HL |
| 20 | Kunde nach Name --> A bis N | DS |

**Tabelle ObjectOccurence**

| OID1 | OID2 | RootID | Depth |
|---|---|---|---|
| 10 | null | 10 | 0 |
| 11 | 10 | 10 | 1 |
| 12 | 10 | 10 | 1 |
| 13 | 12 | 10 | 2 |
| 14 | null | 10 | 1 |
| 15 | 14 | 10 | 2 |
| 20 | null | 20 | 0 |
| 13 | 20 | 20 | 2 |
| 14 | null | 20 | 1 |
| 15 | 14 | 20 | 2 |

**Abb. 11:** Löschen von Objektausprägungen (ganze Hierarchiestufe)

## 2.3 Neue und ergänzte Komponenten des H2-Toolsets

Aufbauend auf der im vorangegangenen Kapitel erfolgten Erläuterung der Basisfunktionalität des H2-Toolset erfolgt die Dokumentation der dargestellten Komponenten zur werkzeuggestützten, konfigurativen Referenzmodellierung sowie einiger weiterer, im Rahmen des Seminars implementierter Funktionen. Diese werden späterer konkretisiert.

*Datenbank-Explorer*

Das H2-Toolset sah Modellierungsmöglichkeiten für Sprachen und Modelle vor. Da jedoch innerhalb eines Referenzmodells verschiedene Modelle in verschiedenen Sprachen angelegt

werden können, galt es, eine neue hierarchische Struktur im Datenbank-Explorer anzulegen. In einem Referenzmodell (als neuer Wurzelknoten der Baumstruktur) können im erweiterten Toolset nicht nur Modelle in verschiedenen Sprachen angelegt, sondern auch die verschiedenen Konfigurationsparameter modelliert werden. Abschnitt 3.1 stellt die Einführung von Referenzmodellen in das Menü des Datenbank-Explorers detailliert dar.

*Die Benutzerverwaltung*

Die bisherigen Erläuterungen zeigen, dass verschiedene Benutzergruppen an einem Referenzmodell arbeiten müssen, die jeweils sehr unterschiedliche Aufgaben verrichten. Es existieren beispielsweise Anwender, die Sachverhalte modellieren und andere, die bestehende Modelle konfigurierbar gestalten. Daraus entwickelt sich der Bedarf einer differenzierten Benutzerverwaltung. Einzelne Benutzer werden Benutzergruppen zugeordnet, an die einzelne Funktionsrechte gebunden sind. Die neue Benutzer- und Rechteverwaltung des H2-Toolsets wird in Abschnitt 3.2 erläutert.

*Die Konfigurationsmechanismen*

Es existiert eine Vielzahl von Mechanismen, nach denen ein Referenzmodell adaptiert werden kann:

- Modelltypselektion
- Elementtypsselektion
- Elementselektion
- Bezeichnungsvariation
- Darstellungsvariation

Die einzelnen Mechanismen werden über Terme ausgeführt. Diese Terme werden im Term-Editor (vgl. Kap.3.4) erstellt. Die Zuweisung der Terme zu Modellen/ Modellelementen erfolgt über den *Configuration Customizer* (vgl. Kap. 3.5 und 3.6). Durch eine Präfixnotation des *Term Editors* wurde die Bedienbarkeit des Toolsets einfach gehalten.

*Kantentypen*

Eine zusätzliche Erweiterung des Toolsets stellen die Kantentypen dar. Durch die Kantentypisierung ist es möglich, die Art einer Beziehung zwischen Modellelementen zu deklarieren (monetär, prozentual, o. ä.). Die Funktionalität und ihre Implementierung werden in Kap. 3.9 vorgestellt.

Ein Überblick über die implementierten Komponenten und deren Strukturierung ist Abb. 12 zu entnehmen. Auch wird in der Abbildung zwischen Komponenten unterschieden, welche im Zuge des Entwicklungsprozesses angepasst bzw. neu erstellt werden mussten. Zudem sind die Komponenten in Schichten angeordnet, wobei übergreifende Support-Komponenten vertikal angeordnet sind. Neben den für das H2-Toolset spezifischen Schichten existieren externe Komponenten, welche die Grundlage für das H2-Toolset bilden. Hier sind das .Net 2.0 Framework sowie ein beliebiges Datenbankmanagementsystem zu nennen.

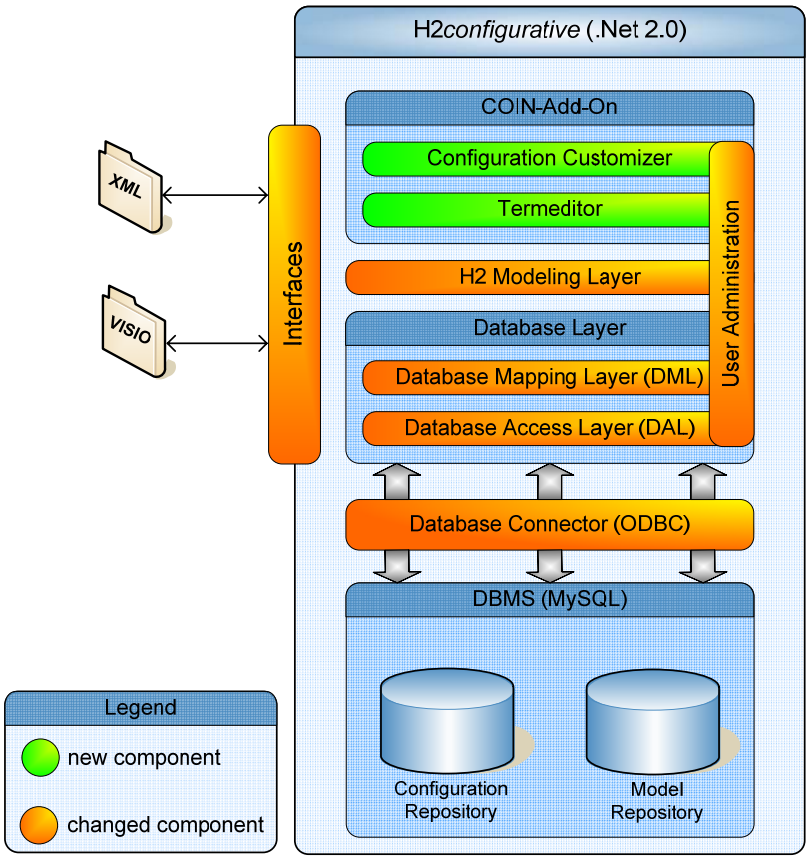



**Abb. 12:** Softwarearchitektur des H2-Toolsets

Aufbauend auf dem gewählten Datenbankmanagementsystem existieren Schichten, die den Transfer und die Umwandlung benötigter Daten in beide Richtungen ermöglichen. Die Database-Connector Schicht ermöglicht zunächst die Anbindung aller weiteren Schichten an die Datenbanken. Die Verwendung der ODBC-Technologie ermöglicht einen Austausch des verwendeten Datenbankmanagementsystems, vorausgesetzt eine entsprechende Connector-Implementierung ist verfügbar. In der Praxis beschränken jedoch SQL-Sprachspezifika diese Austauschbarkeit. Daher ist aktuell nur eine Verwendung von MySQL, MS-SQL oder ORACLE-Datenbanken möglich.

Als eine auf dem Database-Connector zugreifende Schicht beinhaltet der Database-Layer Methoden, um auf die Datenbankinhalte zuzugreifen und diese in objektorientierte Strukturen zu kapseln. Der Zugriff auf die Datenbank erfolgt im Database-Access-Layer, die Abbildung der relationalen Daten in programminterne Objekte wird im nachfolgenden Database-Mapping-Layer vorgenommen.

Die Geschäftslogik-Komponente in Form des H2-Modeling-Layer umfasst im Wesentlichen die bereits aus dem H2-Toolset bekannte Funktionalität zur Modellierung von Sprachen und Modellen. Zusätzlich wurde diese Schicht um Bestandteile zur Realisation der konfigurativen Referenzmodellierung erweitert. Der Termeditor als einer dieser Bestandteile ermöglicht das Editieren der zur Konfiguration benötigten Terme. Der „Configuration Customizer“ ermöglicht die Zuweisung der erstellten Terme zu Modell- und Metamodellelementen. Beide Komponenten sind integrale Bestandteile der im Rahmen des Projektseminars bereitgestellten Funktionalität.

Die XML-Schnittstelle wurde überarbeitet und basiert jetzt auf dem in .NET 2.0 verfügbaren XML-Serializer, welcher die Serialisierung beliebiger .Net Objekte ermöglicht. Die Benutzerverwaltung wurde grundlegend überarbeitet und an die Anforderungen des COIN-Add-Ons angepasst.

# 3 Fachkonzept, DV-Konzept und Implementierung der einzelnen Funktionen

Um die automatische Konfiguration von Referenzinformationsmodellen im H2-Toolset zu ermöglichen, wurde das H2-Toolset sowohl auf Fachkonzept, DV-Konzept- und Implementierungsebene verändert. Dieses Kapitel erörtert die vorgenommenen Änderungen und Erweiterungen. Die Unterkapitel sind anhand logischer Programmeinheiten unterteilt, deren Fach- und DV-Konzept sowie deren implementierungstechnische Details vorgestellt werden, um das fachliche und technische Verständnis der einzelnen Komponenten zu gewährleisten und zukünftige Erweiterungen zu erleichtern.

## 3.1 Erstellung und Verwaltung von Referenzmodellen

### 3.1.1 Fachkonzept

Der grundlegende Aufbau des H2-Toolsets muss an die fachkonzeptionellen Spezifika der konfigurativen Referenzmodellierung angepasst werden. Sprachen und die zugehörigen Modelle müssen im Kontext einer übergeordneten Instanz in Form eines Referenzmodells existieren. Daher ist es notwendig, eine Sprache immer eindeutig einem Referenzmodell zuzuordnen, Referenzmodelle werden als übergeordnete Gruppierungskomponente neu in die Toolset-Struktur aufgenommen. Grafisch wird dies durch einen neuen Knoten im Datenbank-Explorer realisiert, welcher direkt unter dem Datenbank-Knoten angesiedelt ist (vgl. Abb. 16).

Ebenfalls soll es möglich sein, eine Sprache in mehreren Referenzmodellen zu verwenden. Hierfür muss diese Sprache in das gewünschte Ziel-Referenzmodell importiert werden.

Neben der Anpassung der Modellierungsstuktur besteht die Notwendigkeit, die für die Konfiguration benötigten Komponenten, wie beispielsweise Konfigurationsparameter und Terme, in die Struktur zu integrieren. Konfigurationskomponenten werden direkt einem Referenzmodell zugeordnet. Für die Mehrfachverwendung steht analog zu den Sprachen eine Im- und Exportfunktion bereit. Zusätzlich bieten Referenzmodelle eine Differenzierung der Benutzerrechte, d. h. die Handlungsmöglichkeiten des Benutzers im Toolset variieren in Abhängigkeit des gewählten Referenzmodells (vgl. Kap. 3.2).

Eine Übersicht über die neu eingeführten Komponenten ist dem ERM in Abb. 13 zu entnehmen.

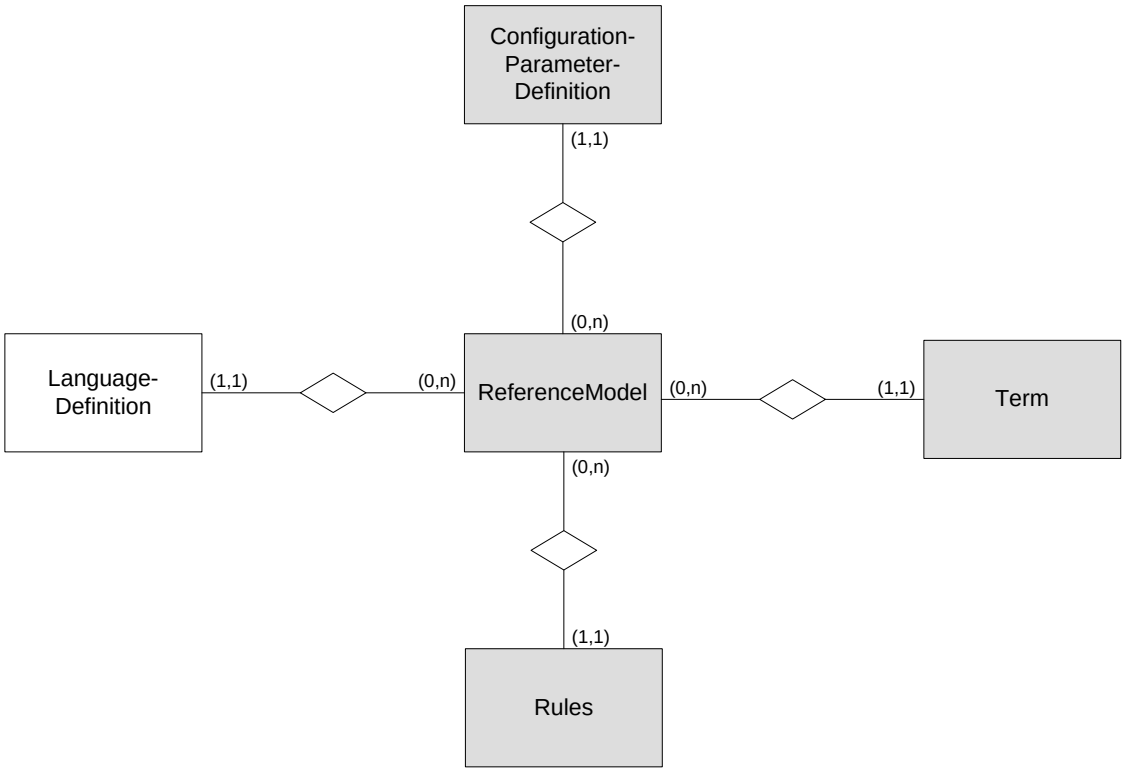



**Abb. 13:** ER-Diagramm der Referenzmodellkomponenten

Um Konflikte bzw. Unstimmigkeiten bei der Modellierung zu vermeiden, muss für den Benutzer immer transparent sein, in welchem Referenzmodell er gerade modelliert. Dies wird erreicht, indem immer nur ein Referenzmodell zur Bearbeitung aktiviert werden kann. Nur wenn ein Referenzmodell aktiviert ist, besteht für den Benutzer die Möglichkeit auf dessen Konfigurationsmechanismen, Sprachen und Modelle zuzugreifen. Aktivierte und deaktivierte Referenzmodelle sind grafisch leicht unterscheidbar und der Wechsel eines Referenzmodells erfolgt durch minimalen Aufwand.

### 3.1.2 DV-Konzept und Implementierung

Referenzmodelle werden auf Datenbankebene in einer neuen Tabelle *Refmodel* abgelegt. In dem Database-Mapping-Layer des Toolsets existiert für diese Tabelle die Klasse *RefModel*, um die Daten der angesprochenen Tabelle objektorientiert zu kapseln und die erforderlichen Methoden wie beispielsweise Laden, Löschen etc. zur Verfügung zu stellen. Ebenso hält diese Klasse verschiedene Collections-Objekte bereit, um Objekte der vom Referenzmodell abhängigen Komponenten (Sprachen, Terme etc.) aufzunehmen und somit einen Objektbaum mit dem Referenzmodell-Objekt als Wurzel aufzubauen. Der Objektbaum entspricht den relationalen Abhängigkeiten der Datenbank-Tabellen. Des Weiteren besitzt die Klasse *RefModel* einen statischen Cache, welcher die bereits einmal aus der Datenbank geladenen Referenzmo-

dell-Objektbäume aufnimmt und hierdurch die Anzahl der direkten Datenbankzugriffe verringert.

| Feld | Typ | NULL | Erläuterungen |
|---|---|---|---|
| id | Int(11) | Nein | Primärschlüssel |
| confParamModelId | Int(11) | Ja | |
| name | Varchar(50) | Ja | |

**Tab. 8:** Tabelle Refmodel

Die Tabelle *Refmodel* (ET: *ReferenceModel*) besteht aus den Spalten *id*, *confParamModelId* und *name*. Die *id* wird fortlaufend bei jeder Neuanlage bzw. jedem Import eines Referenzmodells erhöht. Anhand dieser *id* werden auf der Datenbankebene die jeweiligen Komponenten zugeordnet, indem diese bei den im Folgenden aufgelisteten Tabellen als Fremdschlüssel eingeht:

| Tabellenname |
|---|
| Confparam |
| Confparamdefinition |
| Language |
| Rules |
| Synonym |
| Term |
| UserConfParamRelation |
| UserGroupRefModelRelation |

**Tab. 9:** Tabellen der Referenzmodellkomponenten

Alle weiteren Tabellen wie beispielsweise *Model*, *Attribut* etc. haben einen indirekten Verweis auf ihr zugeordnetes Referenzmodell. Ein Modell kann z. B. über die Sprache einem Referenzmodell eindeutig zugeordnet werden.

Der Objektbaum zur Abbildung der relationalen Beziehungen auf objektorientierter Ebene hat folgenden Aufbau:

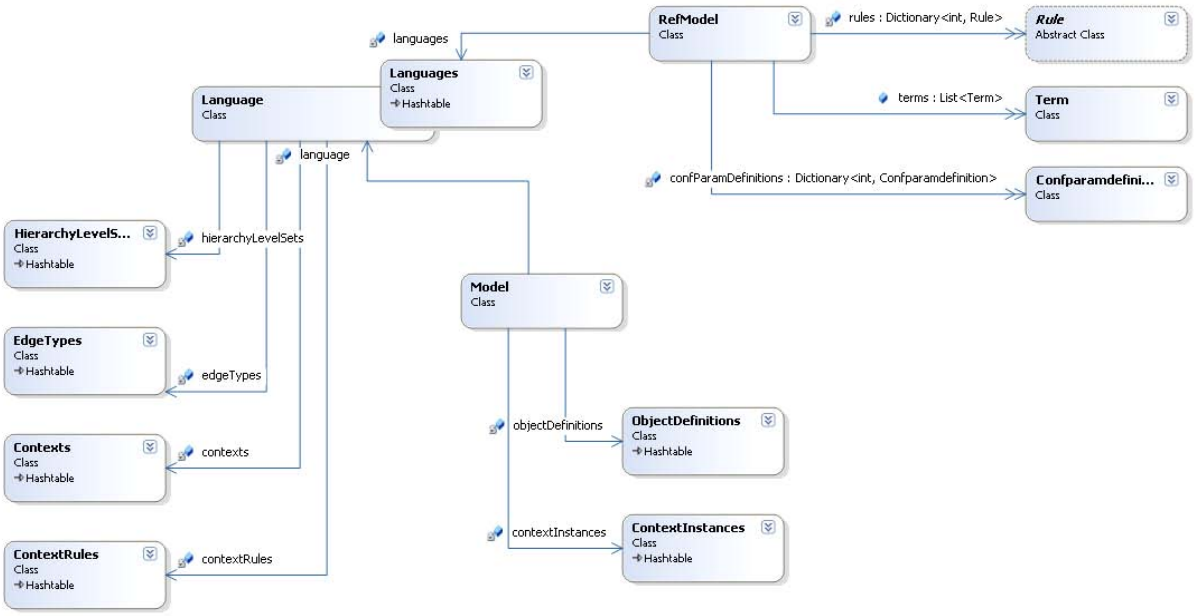



**Abb. 14:** Klassendiagramm des *RefModels* und seiner Komponenten

Da ein Referenzmodell-Objekt als Wurzel eines verzweigten Objektbaums dient, musste bei der Implementierung der Klasse *RefModel* ein besonderes Augenmerk auf den Aufbau dieses Objektbaums gelegt werden. Hierbei ist u. a. die richtige Wahl geeigneter Collection-Typen hervorzuheben (vgl. Abschnitt Objektbaum im Anhang). Aus dem in der nachfolgenden Abbildung dargestellten Klassendiagramm können die jeweils gewählten Collection-Typen entnommen werden.

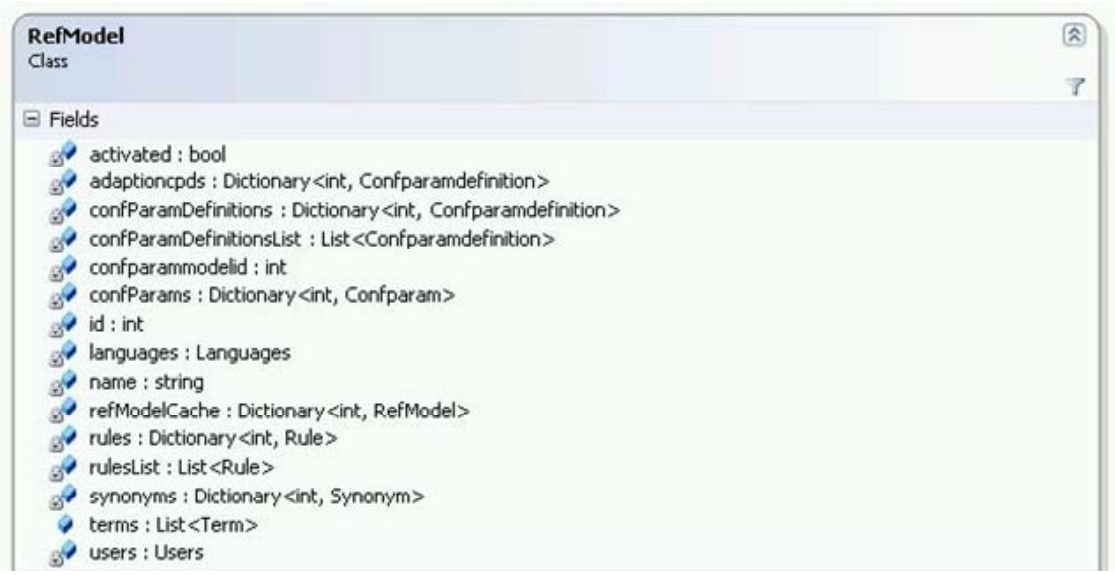



**Abb. 15:** Klassendiagramm der Klasse *RefModel*

Durch die Möglichkeit der Aktivierung und Deaktivierung von Referenzmodellen konnten zusätzlich folgende Feinheiten bei der Implementierung vorgenommen werden: Erst bei der

Aktivierung eines Referenzmodells werden die zugeordneten Komponenten aus der Datenbank bzw. dem Cache geladen. Durch dieses verzögerte Nachladen wird ein verlängerter Ladeprozess direkt nach der Anmeldung vermieden, da nicht alle Referenzmodelle inklusive der abhängigen Komponenten direkt beim Start geladen werden müssen. Ein deaktiviertes Referenzmodell agiert somit nach dem Start des H2-Toolsets zunächst nur als Stellvertreter, abhängige Komponenten werden erst bei Bedarf, d. h. bei erstmaliger Aktivierung aus der Datenbank geladen. Zusätzlich wird nach erstmaligem Laden aus der Datenbank ein Cache mit Referenzmodell-Objekten gefüllt, um unnötige Datenbankzugriffe zu vermeiden. Dies trägt zu einem Performancegewinn bei mehrmaligem Aktivieren und Deaktivieren bei.

***Navigation im Datenbankexplorer***

Abb. 16 zeigt die neue Struktur des Datenbank-Explorers (rechts) im Vergleich zur bisherigen Struktur:

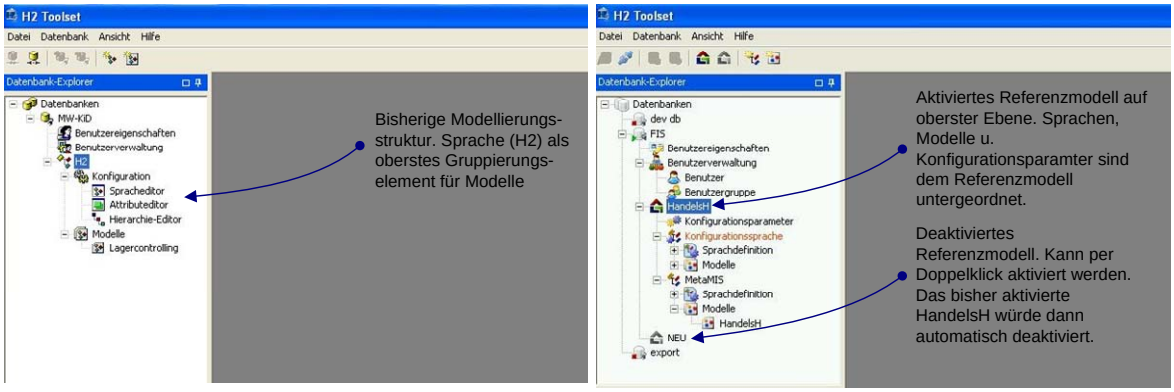



**Abb. 16:** Struktur des Datenbank- Explorer

Innerhalb eines Referenzmodells ist der Benutzer immer eindeutig einer Gruppe zugeordnet, welche seine Handlungsmöglichkeiten in Form ausführbarer Befehle determiniert (siehe auch Kap 3.2: Benutzerverwaltung). Damit der Benutzer immer einen Überblick behält, in welchem Referenzmodell er gerade aktuell arbeitet, müssen diese vor der Bearbeitung aktiviert werden. Wird versucht ein zweites Referenzmodell zu aktivieren, wird das bisher aktivierte Referenzmodell automatisch deaktiviert. Die Aktivierung erfolgt über einen Doppelklick auf den Referenzmodell-Knoten oder über das Kontextmenü. Letzteres wird per Rechtsklick auf einen Referenzmodell-Knoten geöffnet und beinhaltet bei einem deaktivierten Referenzmodell nur den Eintrag ‚Referenzmodell aktivieren'. Wird das Kontextmenü eines aktivierten Referenzmodells geöffnet, so sind folgende Menüpunkte verfügbar:

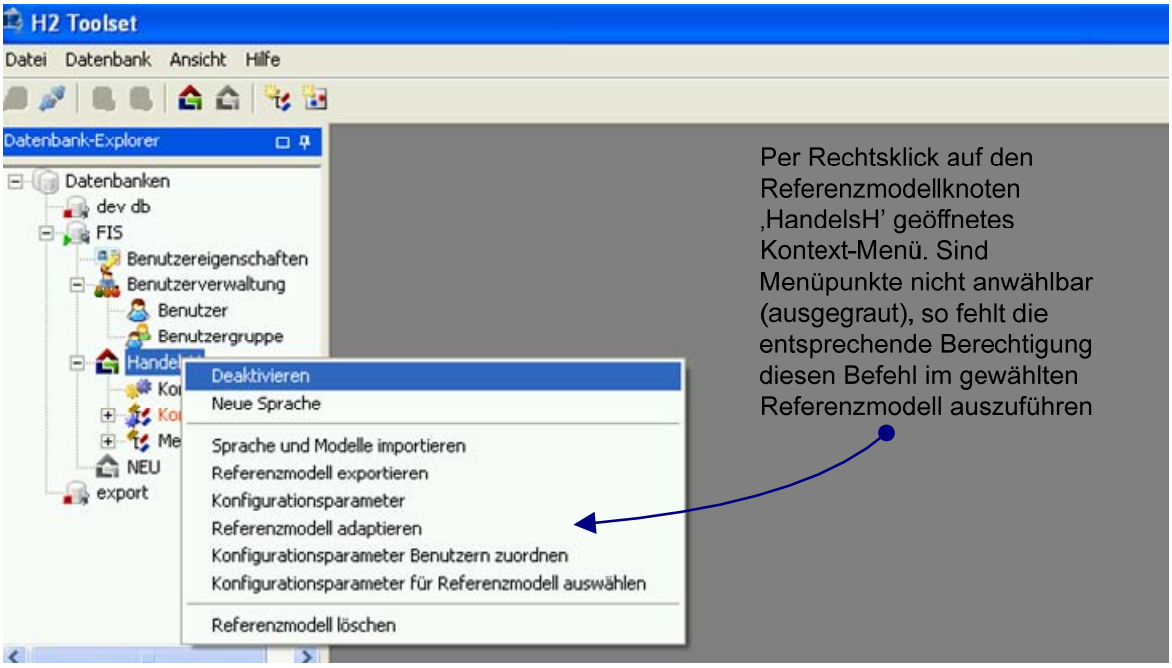



**Abb. 17:** Kontextmenü eines Referenzmodells

Die einzelnen Menüpunkte erfüllen folgende Funktion:

*Deaktivieren*

Das aktuell gewählte Referenzmodell wird deaktiviert. Diese Funktionalität wird automatisch durchgeführt, falls ein weiteres Referenzmodell aktiviert wird. Bei der Deaktivierung werden alle vom Referenzmodell abhängigen Fenster wie beispielsweise der Termeditor geschlossen und alle dem Referenzmodell untergeordneten Knoten aus dem Datenbank-Explorer entfernt. Ebenfalls ändert sich die grafische Darstellung des Referenzmodell-Knotens.

*Neue Sprache*

Fügt einen neuen Sprachknoten unterhalb des Referenzmodell-Knotens hinzu. Zuvor muss in einem Eingabedialog ein Name für die Sprache festgelegt werden. Als Unterknoten der Sprache wird der Knoten zum Verwalten der Sprachdefinition sowie der Gruppierungsknoten zum Anlegen von Modellen automatisch angelegt.

*Sprache und Modelle importieren*

Nach Auswahl dieser Option erscheint ein Dialogfenster, welches die Wahl einer XML-Datei mit Sprachinformationen aus dem Dateisystem ermöglicht. Diese Datei kann optional Daten von Modellen enthalten, welche in der zu importierenden Sprache angelegt sind. Für die neue

Sprache sowie eventuell vorhandene Modelle werden neue Knoten unterhalb des Referenzmodell-Knotens angelegt. Es ist nicht möglich, Modelle ohne Sprachdefinitionen zu importieren.

*Referenzmodell exportieren*

Durch die Export-Funktionalität ist es möglich, Referenzmodelle auch nach außen hin verfügbar zu machen. Hierbei wird die gesamte Struktur eines Referenzmodells in eine XML-Datei übertragen. Diese Datei beinhaltet alle dem Referenzmodell zugeordneten Komponenten wie Sprachen, Modelle, Terme, Konfigurationsparameter etc.

*Konfigurationsparameter*

Öffnet den Editor zum Erstellen neuer Konfigurationsparameter wie Unternehmensmerkmale und Perspektiven. Dieser Editor kann ebenfalls mit Hilfe eines Doppelklicks auf den Knoten ‚Konfigurationsparameter' geöffnet werden.

*Referenzmodell adaptieren*

Bei der Konfiguration (vgl. Abschnitt 3.8) werden Elemente des Referenzmodells anhand der gesetzten Konfigurationsansatzpunkte und der gewählten Konfigurationsparameter ausgeblendet. Das Ergebnis ist ein neues (Referenz)-Modell, welches eine Teilmenge der Elemente des Ausgangsmodells enthält. Dieses neue, konfigurierte Referenzmodell wird nach Beendigung des Konfigurationsprozesses als zusätzlicher Knoten in der Struktur des Datenbank-Explorers deaktiviert angezeigt. Dies bietet die Möglichkeit, das Ergebnis der Konfiguration direkt zu prüfen und ggf. Anpassungen im Ausgangsmodell vorzunehmen. Es ist ebenfalls möglich, das konfigurierte Referenzmodell einem erneuten Konfigurationsprozess zu unterziehen. Entspricht das konfigurierte Referenzmodell nicht den Erwartungen, so kann es problemlos über die Löschen-Funktionalität aus dem Toolset entfernt werden.

*Konfigurationsparameter für Benutzer auswählen*

Bereits angelegte und vom Referenzmodell beinhaltete Konfigurationsparameter können hier den einzelnen Benutzern zugeordnet werden (vgl. Kap. 3.3.2). Zugeordnete Konfigurationsparameter werden bei der Anmeldung des Benutzers in den vorhandenen Termen ausgewertet und können so zu einer eingeschränkten Sichtweise (in Form ausgeblendeter Komponenten) auf die vorhandenen Sprachen und Modelle führen.

*Konfigurationsparameter für Referenzmodell auswählen*

Bei dieser Vorselektion (vgl. Kap. 3.3.2) werden aus einer vorhanden, globalen Liste von Konfigurationsparametern diejenigen ausgewählt, welche für eine Konfiguration potenziell zur Verfügung stehen sollen.

*Referenzmodell löschen*

Der Referenzmodell-Knoten inklusive aller untergeordneten Knoten wird aus dem Datenbank-Explorer entfernt. Ebenfalls werden alle von dem gewählten Referenzmodell abhängigen Komponenten während des Prozesses aus der Datenbank und dem Cache im Speicher gelöscht.

Um im Toolset neue Referenzmodelle anlegen zu können, muss das Kontextmenü eines Datenbank-Knotens geöffnet werden. Dies erfolgt analog durch Rechtsklick auf den jeweiligen Knoten.

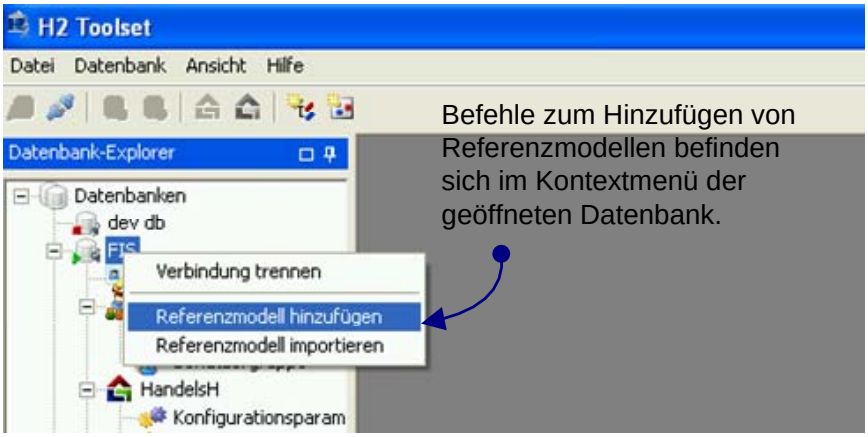



**Abb. 18:** Hinzufügen eines Referenzmodells

*Referenzmodell hinzufügen*

Ein neues Referenzmodell wird direkt unter dem Datenbank-Knoten als deaktivierter Knoten dem Datenbank-Explorer hinzugefügt. Neue Referenzmodelle enthalten weder Sprachen und Modelle, lediglich ein Menüknoten zum Öffnen des Konfigurationsparameter-Editors ist vorhanden. Neue Sprachen und Modelle können in diesem Modell angelegt bzw. vorhandene Sprachen und Modelle können importiert werden.

*Referenzmodell importieren*

Durch Angabe einer gültigen XML-Datei können Referenzmodelle auch durch externe Dateien in das H2-Toolset integriert werden. Ein importiertes Referenzmodell wird im deaktivierten Zustand dem Datenbank-Explorer hinzugefügt.

## 3.2 Benutzer- und Rechteverwaltung

### 3.2.1 Fachkonzept

Ein Modellierungswerkzeug, das sowohl die Erstellung konfigurierbarer Referenzmodelle als auch deren Anwendung unterstützt, benötigt umfangreiche Funktionen für die Verwaltung seiner Benutzer. Je nach Rolle, die der Benutzer in einem individuellen Referenzmodell einnimmt, sind ihm angemessene Funktionsrechte zuzuordnen. So sind einem Nutzer, der Referenzmodelle erstellt und Regeln für dessen weitere Konfiguration definiert, weitaus spezifischere Funktionsrechte einzuräumen als einem Nutzer, der ein fertig konfiguriertes Referenzmodell praktisch anwendet. Die detaillierte Ausgestaltung der Funktionsrechtsausprägungen lehnt sich diesbezüglich sehr stark an die Aufgabenbereiche des Referenzmodellierungs-Workflows (vgl. Kap. 2.1) an.

Um den Wartungsaufwand und die Komplexität des Zuordnungsproblems von Funktionsrechten an Benutzer zu reduzieren, basiert die Realisierung der Benutzerverwaltung im H2-Toolset auf einem flexiblen Benutzergruppenkonzept. Funktionsrechte werden daher ausschließlich an Benutzergruppen vergeben, wobei Benutzer in einem Referenzmodell einer vordefinierten Nutzergruppe angehören können. Auf diese Weise lassen sich die Funktionsrechte nicht-disjunkt an Benutzergruppen vergeben, d. h. die Berechtigungen einzelner Gruppen können sich durchaus überschneiden.

Das ER-Modell in Abb. 19 macht diesen Zusammenhang auf Typebene deutlich: Funktionsrechte werden Benutzergruppen zugeordnet und über eine weitere Relation (UG-UR-Ass.) indirekt referenzmodellspezifisch an Nutzer vergeben.

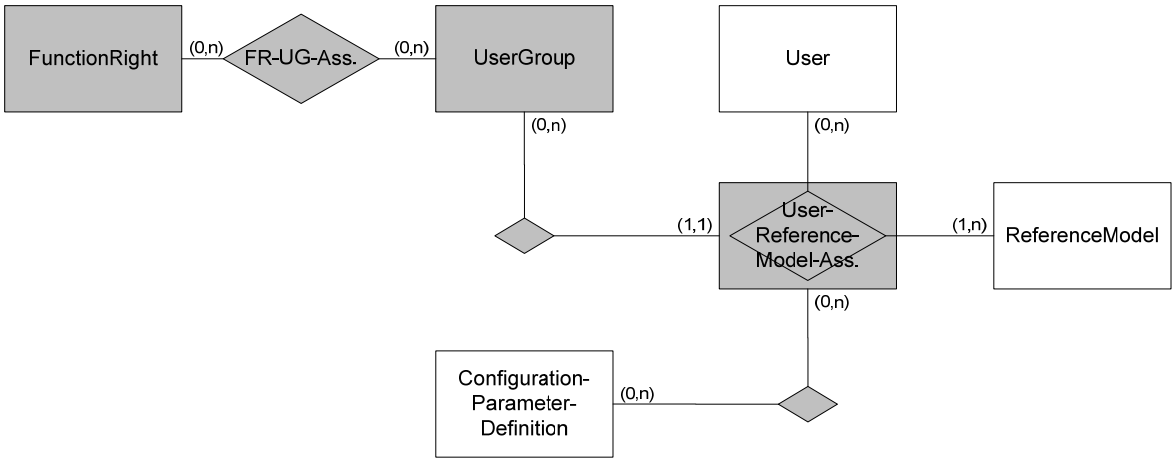



**Abb. 19:** ER-Diagramm der Benutzer- und Rechteverwaltung

Um H2-Nutzern eine individualisierte Sicht auf die im H2-Repository gespeicherten Modelle zu gewähren, sind ihnen referenzmodellspezifisch Konfigurationsparameter zuzuordnen. Je nach Verwendungskontext lassen sich H2-Modelle auf diese Weise entsprechend aufbereiten (z. B. vom Umfang reduzieren). Aus diesem Grund steht *User-ReferenceModel-Ass.* mit *ConfigurationParameterDefinition* in Beziehung, der von der Anzeigekomponente des H2-Toolsets für sämtliche Konfigurationsmechanismen entsprechend auszuwerten ist (eine detaillierte Erläuterung der Konfigurationsparameter erfolgt in Kap. 3.3.2).

*Funktionsrechte*

Die Funktionsrechte kapseln den Zugriff auf beliebige Objekte im H2-Toolset. Zugreifbare Objekte, die einer Benutzergruppe nur bei vorhandenem Funktionsrecht angezeigt werden, betreffen in erster Linie Funktionen aus dem Referenzmodellierungs-Workflow. So sind etwa sämtliche, die Konfiguration von Referenzmodellen betreffende Zugriffsobjekte im Funktionsrecht *FunctionReferenceModelling* gekapselt. Steuerungselemente der Benutzeroberfläche, die exklusiv die Referenzmodellierung betreffen, werden folglich nur bei Vergabe des Rechtes *FunctionReferenceModelling* aktiviert. Insgesamt sind im H2-Toolset die vier in Tab. 10 aufgelisteten Funktionsrechte zu unterscheiden.

| ID | Function |
|---|---|
| 1 | FunctionReferenceModelling |
| 2 | FunctionAdapting |
| 3 | FunctionLanguageConfiguration |
| 4 | FunctionModelling |

**Tab. 10:** Funktionsrechte im H2-Toolset

Die eigentliche Verwaltung der Benutzer und Benutzergruppen inklusive der Vergabe von Funktionsrechten stellt selbst ein Recht dar, dessen Vergabe im H2-Toolset jedoch unabhän-

gig von Referenzmodellen erfolgt. Dieses Recht ist bewusst von den anderen Rechten getrennt, da seine Vergabe auf einem höheren Abstraktionsniveau angesiedelt ist. Es wird daher nicht explizit von der Benutzerverwaltung vergeben, ansonsten bestünde fälschlicherweise die Möglichkeit, dass Administratoren sich selbst das elementarste Recht entziehen könnten. Des Weiteren ist es auf diese Weise von der Vergabe von Funktionsrechten im Zusammenhang mit Referenzmodellen entkoppelt. Im H2-Toolset äußert sich dies darin, dass nur Benutzer, die der Benutzergruppe auf der obersten Ebene der Benutzergruppenhierarchie angehören, befugt sind, die eigentlichen Basisfunktionen der Benutzerverwaltung auszuüben. Hierzu zählen das Anlegen, Löschen und Bearbeiten von Benutzergruppen und Benutzern. Im Detail schließt dies die Zuordnung von Funktionsrechten zu Benutzergruppen sowie die Zuweisung von Benutzergruppen an Benutzer in Referenzmodellen ein.

- FunctionReferenceModelling: Erst wenn ein Benutzer einer Gruppe zugeordnet ist, welche dieses Funktionsrecht besitzt, wird die konfigurative Komponente der Erweiterung des H2-Toolsets für ihn verfügbar. Neben der Möglichkeit Referenzmodelle anzulegen, umzubenennen sowie zu löschen werden folgende, die Konfiguration betreffende Funktionen, zugänglich:

    - Referenzmodelle importieren bzw. exportieren
    - Definition der Konfigurationsparametersprache
    - Bedienung des Modell-Editors für Konfigurationsparameter und Ontologien
    - Zuordnung von Konfigurationsparametern zu Referenzmodellen
    - Verwendung des Termeditors (Anlegen, Bearbeiten und Löschen von Konfigurationstermen)
    - Verwendung des Configuration Customizers (Zuordnung von Konfigurationstermen zu verschiedenen Objekten)

- FunctionAdapting: Sofern ein Benutzer einer Benutzergruppe zugeordnet ist welche dieses Funktionsrecht besitzt, ist er befugt, Referenzmodelle anhand zuvor definierter Konfigurationsregeln an unterschiedliche Anwendungskontexte anzupassen. Hierzu zählen eine Dialogkomponente, die entsprechende Konfigurationsparameter für eine zu generierende Modellvariante auswählt sowie ein Fenster, mit dem Benutzern Konfigurationsparameter zugeordnet werden können.

- FunctionLanguageConfiguration: Bei Vergabe dieses Rechtes besitzen Nutzergruppen im jeweiligen Referenzmodell exklusiv die Möglichkeit der Metamodellierung, d. h. Sprachen anzulegen und zu löschen sowie Sprachdefinitionen bearbeiten.

- FunctionModelling: Um neue Modelle anzulegen, zu löschen sowie innerhalb eines Modellkontextes Elemente anzulegen, zu löschen sowie zu bearbeiten, muss das Funktionsrecht „FunctionModelling“ vergeben sein. Seine Deaktivierung ist insbesondere dann interessant, falls eine Nutzergruppe innerhalb eines Referenzmodells lediglich Lesezugriff besitzen soll.

Entscheidend bei der Administration der Benutzergruppen ist, dass sie hierarchisch angelegt werden und die Vergabe der Zugriffsrechte vererbbar ist. Abgesehen von der Benutzergruppe der Administratoren kann einer Nutzergruppe daher entweder der Zugriff auf eines der Funktionsrechte verwehrt oder aber von der übergeordneten Benutzergruppe vererbt werden. Auf diese Weise ist sichergestellt, dass untergeordnete Benutzergruppen maximal über die gleichen bzw. eingeschränkte Zugriffsrechte verfügen wie ihre Elterngruppen (restriktive Vererbung).

### 3.2.2 DV-Konzept und Implementierung

*Tabelle UserGroup*

Die Erweiterung des H2-Repository anhand des ER-Diagramms aus Abb. 19 macht aufgrund der Umstrukturierung des Benutzergruppenkonzepts zusätzliche Tabellen erforderlich. Neben der Tabelle *Refmodel* (vgl. hierzu ausführlich Abschnitt 3.1), ist die Tabelle *UserGroup* von zentraler Bedeutung. Während bislang Benutzer und Benutzergruppen in einer gemeinsamen Tabelle (*UserList*) gespeichert wurden und mit Hilfe eines Flags festgehalten wurde, ob es sich bei einem Eintrag um eine Gruppe bzw. einen einzelnen Benutzer handelt, erfordert das neue Konzept separate Tabellen.

| **Feld** | **Typ** | **Beispiel** | **Erläuterung** |
|---|---|---|---|
| id | int(11) | 1 | ID des Eintrages |
| name | varchar(50) | Admins | Eindeutiger Name der Benutzergruppe |
| disableDelete | tinyint(1) | 1 | Flag, das das Löschen und Bearbeiten des Datensatzes verbietet<br>1 = Schreibschutz aktiv<br>0 = Schreibschutz inaktiv |
| parentId | int (11) | 0 | ID des Vaterknotens in der Benutzergruppenhierarchie (handelt es sich um einen Wurzelknoten, so ist die parentid 0) |

**Tab. 11:** Tabelle UserGroup

Neben den sich selbst erklärenden Attributen *id* und *name* handelt es sich bei *disableDelete* um ein Flag, das das Löschen und Bearbeiten einer Benutzergruppe verbietet. Es kann im

Toolset über eine Checkbox gesetzt werden. Damit Administratoren sich nicht selbst die Rechte entziehen können, ist *disableDelete* für die Gruppe „Admins“ per default auf 1 gesetzt sowie die Checkbox deaktiviert. Um die hierarchische Baumstruktur der Benutzergruppen abzubilden, speichert das Attribut *parentId* den Vaterknoten einer Benutzergruppe. Im Beispiel enthält er den Wert Null, da es sich bei dem Knoten der Benutzergruppe „Admins“ um den Wurzelknoten handelt.

*Tabelle UserList*

Neben der zur Speicherung der Nutzergruppen ist diese Tabelle mit den eigentlichen Nutzerdaten elementarer Bestandteil der Benutzerverwaltung (vgl. Tab. 12).

| **Feld** | **Typ** | **Beispiel** | **Erläuterung** |
|---|---|---|---|
| id | int(11) | 1 | ID des Eintrages |
| name | varchar(50) | Admins | Eindeutiger Name des Benutzers |
| fullName | varchar(50) | Administrator | Vollständiger Benutzername |
| email | varchar(50) | h2admin@wi.uni-muenster.de | E-Mail-Adresse des Benutzers |
| password | varchar(24) | ISMvKXpXpadDiUoOSoAfww== | Verschlüsselt abgespeichertes Zugangspasswort |
| disableDelete | tinyint(1) | 1 | Flag, das das Löschen und Bearbeiten des Datensatzes verbietet<br>1 = Schreibschutz aktiv<br>0 = Schreibschutz inaktiv |
| loggedOn | int (11) | 1 | Flag, das festhält ob ein Nutzer gerade angemeldet ist<br>1 = Benutzer ist angemeldet<br>0 = Benutzer ist abgemeldet |
| Image | BLOB | <BLOB> | Speicherung eines Fotos des Nutzers |

**Tab. 12:** Tabelle UserList

Die Attribute sind im Wesentlichen selbsterklärend. Entscheidend ist, dass das bislang vorhandene Feld *type* zur Unterscheidung von Benutzern und Benutzergruppen überflüssig geworden ist.

*Tabelle UserFunction*

Die Tabelle *UserFunction* entspricht dem Relationshiptypen „FR-UG-Ass.“ aus dem ER-Modell (vgl. Tab. 13).

| Feld | Typ | Beispiel | Erläuterung |
|---|---|---|---|
| functionId | int(11) | 5 | ID des Funktionsrechtes |
| groupId | int(11) | 1 | ID der zugehörigen Benutzergruppe |
| access | int(11) | 1 | Das dem zusammengesetzten Schlüssel zugeordnete Zugriffs-recht<br>0 = Zugriffsverweigerung (Deny)<br>1 = Zugriff (Grant)<br>2 = Vererbung (Inherit) |

**Tab. 13:** Tabelle Userfunction

Zu den zusammengesetzten Schlüsseln aus Benutzergruppe (*groupId*) und Funktionsrecht (*functionId*) speichert das Attribut *access* den entsprechend gesetzten Zugriffswert. Hierbei steht der Zahlenwert *eins* für Zugriff (Grant), *zwei* für die Vererbung des Zugriffsrechts und *null* für Zugriffsverweigerung.

*Tabelle UserGroupRefmodelRelation*

Da Benutzer-Referenzmodell-Assoziationen (ET: *UserRefmodel-Ass.*) genau eine Beziehung mit Entities vom Typ *UserGroup* eingehen (die Max-Kardinalität ist 1), muss die *Usergroup-User-ReferenceModel-Ass.* nicht als eigene Tabelle umgesetzt werden. Stattdessen geht der Primärschlüssel der *UserGroup* (*groupId*) als Fremdschlüssel in die Tabelle *UserGroupRef-ModelRelation* ein. Da hingegen die referenzmodellspezifische Zuordnung von Konfigurationsparametern zu Benutzern unabhängig von der jeweiligen Benutzerrolle ist, gehen in die Relation *User-Confparam-Ass.* nur die Nutzer- und Referenzmodell-Id ein.

| Attribut | Typ | Beispiel | Erläuterung |
|---|---|---|---|
| refmodelId | int(11) | 1 | ID des Referenzmodells |
| userId | int(11) | 1 | Benutzer-ID |
| groupId | int(11) | 1 | ID der Benutzergruppe |

**Tab. 14:** Tabelle UserGroupRefmodelRelation

*Beteiligte Pakete und Klassen*

Die Klassen zur Implementierung der Veränderungen der Benutzerverwaltung sind in den Paketen *UserManagement* sowie den Subpaketen *DatabaseMapping::UserManagement* und *DatabaseMapping::Configuration* zu finden. Während das erstgenannte Paket die Geschäftslogik und die grafische Benutzeroberfläche betrifft, stellen die beiden Subpakete die Abbildung der objektorientierten auf relationale Daten zur Verfügung.

Die zusätzliche Datenbanktabelle für Benutzergruppen äußert sich in der Implementierung der hinzu gekommenen Klasse *UserGroup*. Neben der Kapselung der Attribute der Benutzer-

gruppen beinhaltet ihre Quellcode-Datei die Konstanten für die Funktionsrechte, die aus Performancegründen in keiner eigenen Tabelle gespeichert sind (*FunctionRechtSet*). Analog dazu stellt die Klasse *UserGroupReferenceModelRelation* Methoden zum Auslesen und Setzen der Funktionsrechte bereit, die von der GUI-Klasse *GroupForm* entsprechend aufgerufen werden.

Ausschlaggebend für die Wirkung der Funktionsrechtzuordnungen ist letztlich die logische Überprüfung der gesetzten Funktionsrechte. Hierzu werden im Hauptfenster des H2-Toolsets *(*der Klasse *MWKiD*) sowie im Datenbank-Explorer (der Klasse *DatabaseExplorer*) von den Funktionsrechten betroffene GUI-Befehle entsprechend aktiviert bzw. deaktiviert. GUI-Befehle sind dabei vom Typ *Command* (Namespace *MW_KiD.General*) und fassen bestimmte Funktionen zusammen, indem sie GUI-Elemente kapseln. Auf diese Weise lässt sich mit Hilfe eines einzigen *Commands* die De- bzw. Aktivierung z. B. von Buttons, Knoten im Datenbank-Explorer oder Kontextmenüelementen, die einem gemeinsamen Kontext entstammen, steuern.

Abb. 20 zeigt exemplarisch einen Ausschnitt aus dem Programmcode des Datenbank-Explorers, der GUI-Befehle für das Funktionsrecht *FunctionRefmodelling* aktiviert. Dazu wird mittels einer If-Abfrage die Methode *CheckFunctionRight* der Klasse *ObjectFilter* (Paket *Configuration*) aufgerufen, die den aktuellen Benutzer aus dem Session-Objekt ausliest und für die zugehörige Benutzergruppe `true` bzw. `false` zurückgibt, sofern das jeweilige Funktionsrecht vergeben bzw. nicht vergeben ist. Als Resultat werden daraufhin mit den einzelnen *Commands* verknüpfte Buttons und Menü-Items gegebenenfalls aktiviert. Ein Überblick über die beteiligten Klassen, Methoden und Attribute der Benutzerverwaltung kann Abb. 21 entnommen werden.

```
if (ObjectFilter.CheckFunctionRight(FunctionRightSet.
          FunctionRefmodelling))
{         this.CmdRefmodelAdd.Enabled = true;
          this.CmdRefModelDelete.Enabled = true;
          this.CmdConfigParameterOpen.Enabled = true;
          this.CmdModeladaption.Enabled = true;
          this.CmdExportReferenceModell.Enabled = true;
          this.CmdConfparamAssignment.Enabled = true;
          this.CmdPreselection.Enabled = true;
          this.CmdLanguageImport.Enabled = true;
}
```

**Abb. 20:** Code-Fragment zur Aktivierung von GUI-Befehlen gemäß zugehöriger Funktionsrechte


**Abb. 21:** Klassendiagramm der Benutzerverwaltung

### *Nutzung der Benutzerverwaltung*

Bei Neuanlage einer Datenbank wird per default das Nutzerobjekt „Administrator“ sowie die Nutzergruppe „Admins“ angelegt, so dass der sich initial anmeldende Nutzer über sämtliche Rechte im Toolset verfügt. Nach der Anmeldung an der Datenbank erscheinen im Menübaum des Datenbank-Explorers auf oberster Ebene die Knoten der Benutzerverwaltung. Während über den Knoten „Benutzereigenschaften“ die Attribute des aktuell angemeldeten Benutzers eingesehen werden können, das Passwort neu gesetzt sowie ein Bild des Benutzers hochgela-

den werden kann, erschließen sich über den aufklappbaren Knoten „Benutzerverwaltung“ die eigentlichen Funktionen der Benutzerverwaltung. Diese sind nur Nutzern der Benutzergruppe der höchsten Ebene der Benutzergruppenhierarchie zugänglich (vgl. Abb. 22).

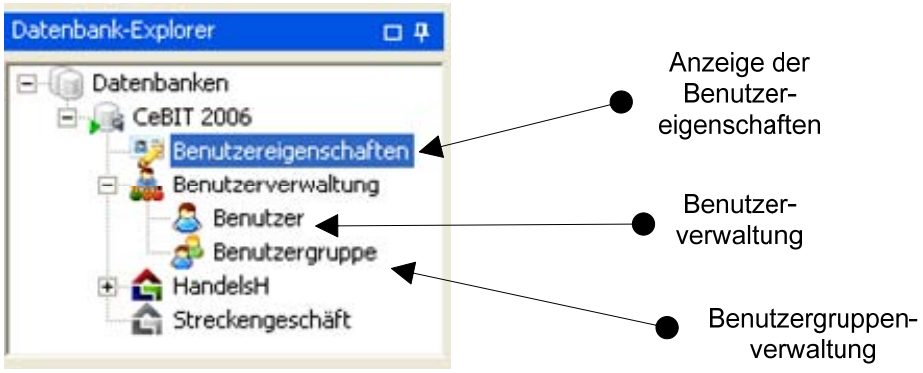



**Abb. 22:** Benutzerverwaltung im Datenbank-Explorer

*Benutzergruppenverwaltung*

Die Neuanlage zusätzlicher Benutzergruppen folgt einer restriktiven Vererbungshierarchie. Im Fenster für die Benutzergruppenverwaltung (vgl. Abb. 23) werden vorhandene Benutzergruppen dementsprechend in einem hierarchischen Baummenü angezeigt, dessen Wurzelknoten auf höchster Ebene die schreibgeschützte Benutzergruppe „Admins“ ist. Von „Admins“ abgeleitete Benutzergruppen erben grundsätzlich die Konfiguration der Funktionsrechte der ihr übergeordneten Benutzergruppe. Über die Registerkarte „Funktionsrechte“ lassen sich die einzelnen Funktionsrechte detailliert konfigurieren, wobei über das jeweilige Auswahlfeld „Benutzerrecht“ entweder der Zugriff verwehrt („Kein Zugriff“) oder aber vom übergeordneten Knoten im Benutzergruppenbaum geerbt werden kann („Erben“).

Eine Benutzergruppenkonstellation, die dem zu Grunde liegenden Referenzmodellierungs-Workflow genügt, ist in Abb. 23 dargestellt. Während dem *Referenzmodellierer* Zugriff auf sämtliche Funktionsrechte gewährt wird, besitzen *Adaptierer* keinen Zugriff auf das Funktionsrecht „Referenzmodellierung“. *Modelliererm* sind nur zur „Modellierung“ berechtigt, wohingegen *Betrachter* durchweg zugriffsbeschränkt sind.

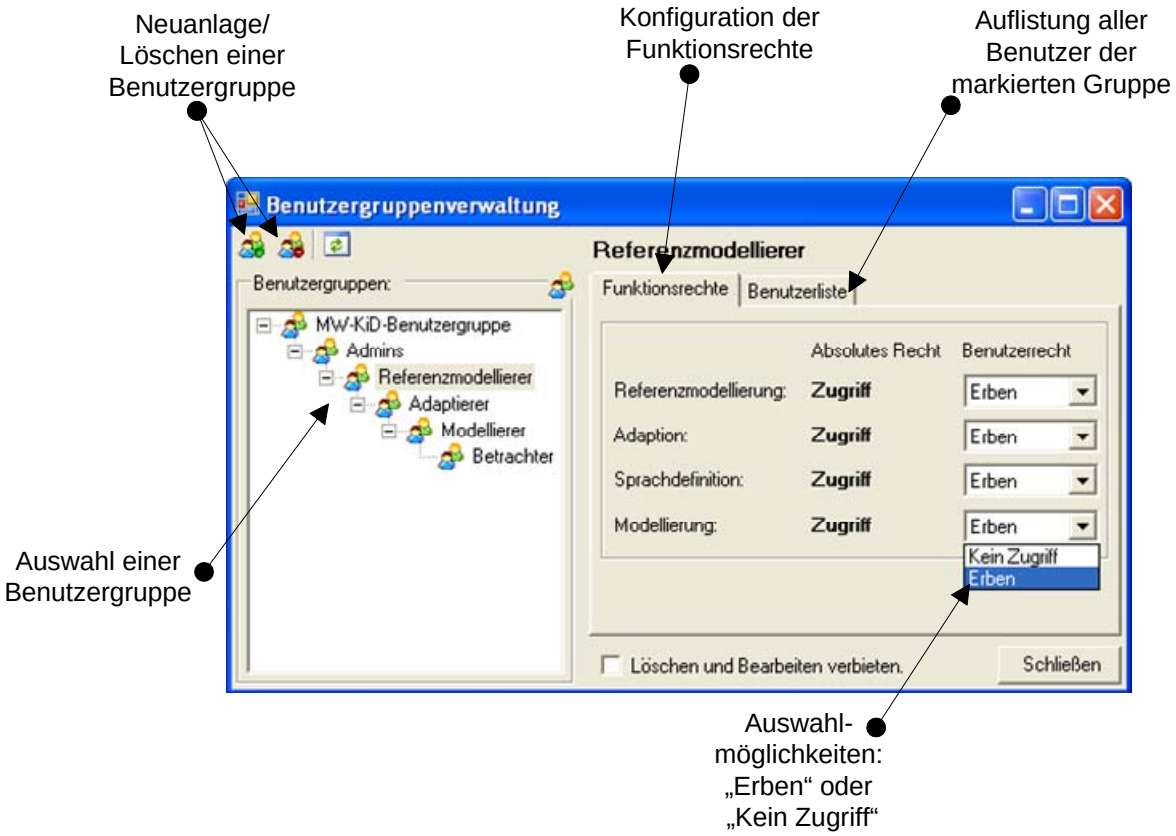



**Abb. 23:** Zuordnung von Funktionsrechten zu Benutzergruppen

*Benutzerverwaltung*

Neben dem Anlegen und Löschen von Benutzern sowie dem Ändern der Benutzerattribute betrifft das Fenster der Benutzerverwaltung im Wesentlichen die referenzmodellspezifische Zuweisung der Benutzergruppen an Nutzer. Um Benutzern Zugriff auf Referenzmodelle zu geben, ist der jeweilige Benutzer auszuwählen sowie in der Registerkarte „Benutzergruppen“ das entsprechende Referenzmodell. Mit Hilfe eines Auswahlfeldes und dem Bestätigungsbutton „Gruppe zuordnen“ lässt sich dann die gewünschte Benutzergruppe zuordnen (vgl. Abb. 24).

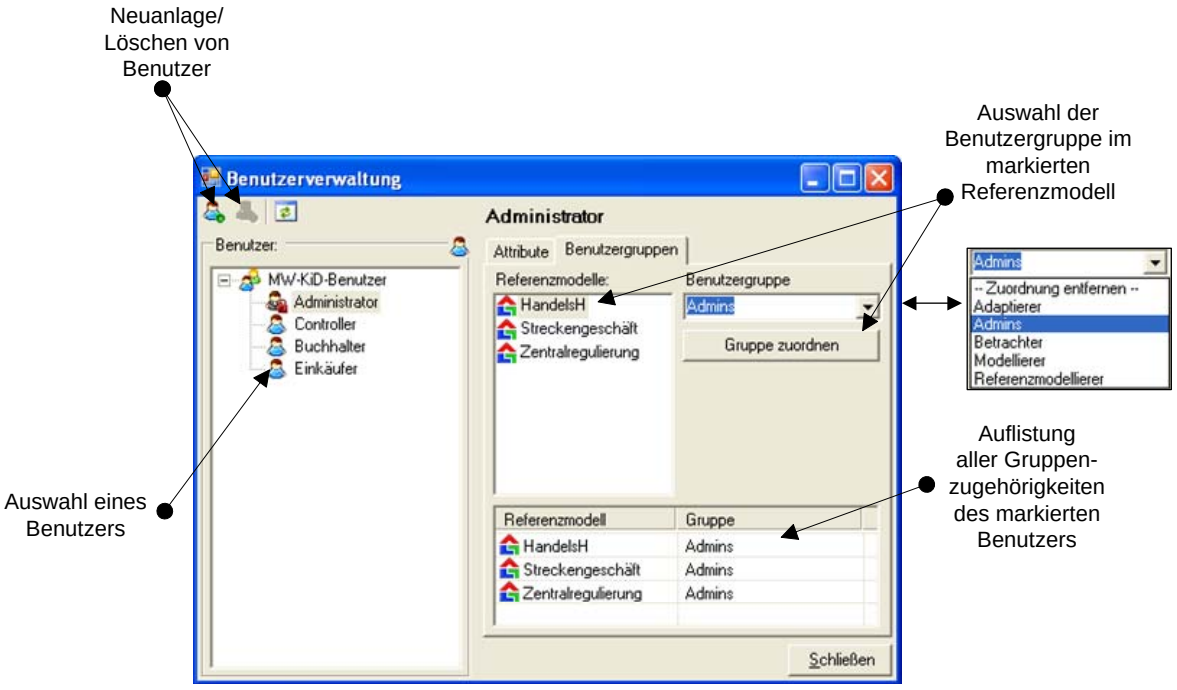



**Abb. 24:** Zuweisung von Benutzergruppen an Benutzer in Referenzmodellen

Um neben der Benutzergruppenzuweisung die Benutzerattribute zu pflegen, steht die Registerkarte „Attribute“ zur Verfügung. Abgesehen von der Änderung sämtlicher Benutzerattribute lässt sich hier ebenso das Passwort zurücksetzen sowie das dem Benutzer hinterlegte Bild ändern (vgl. Abb. 25).

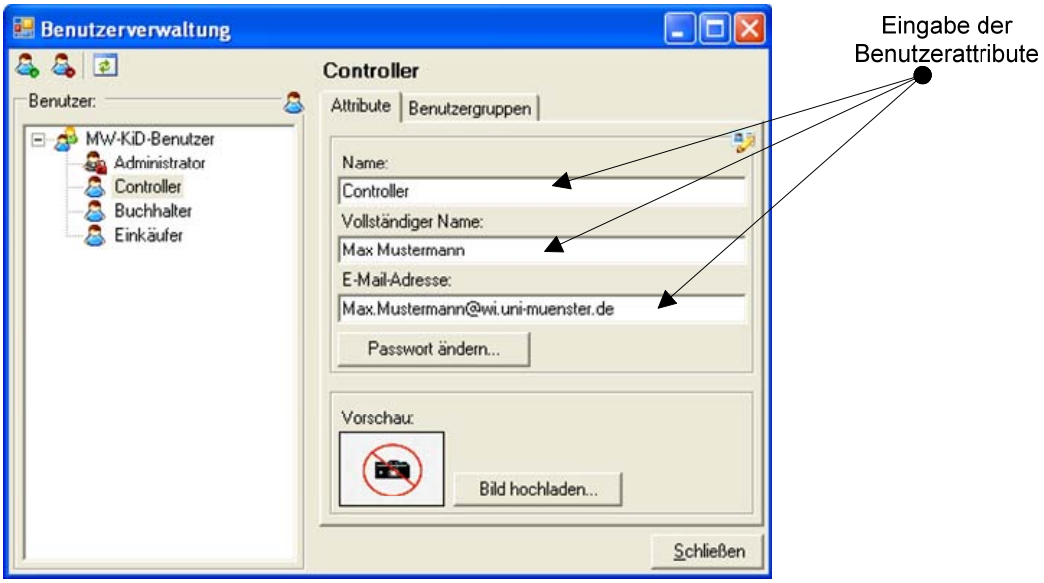



**Abb. 25:** Ändern der Benutzerattribute

## 3.3 Erstellung und Verwaltung von Konfigurationsparametern

### 3.3.1 Fachkonzept

Konfigurationsparameter bilden das Kernstück aller Konfigurationsmechanismen des H2-Toolsets. Sie lassen sich in die drei Gruppen Unternehmensmerkmale, Perspektiven und den Sonderfall kombinierte Konfigurationsparameter unterteilen. Jede Gruppe beinhaltet mehrere Dimensionen, die wiederum mehrere Dimensionsausprägungen enthalten. In Abb. 26 ist die Struktur der Konfigurationsparameter grafisch dargestellt und mit Beispielen für die einzelnen Elemente versehen. Sollte eine zukünftige Erweiterung um zusätzliche Gruppen notwendig werden, muss die hier aufgezeigte Struktur eingehalten werden.

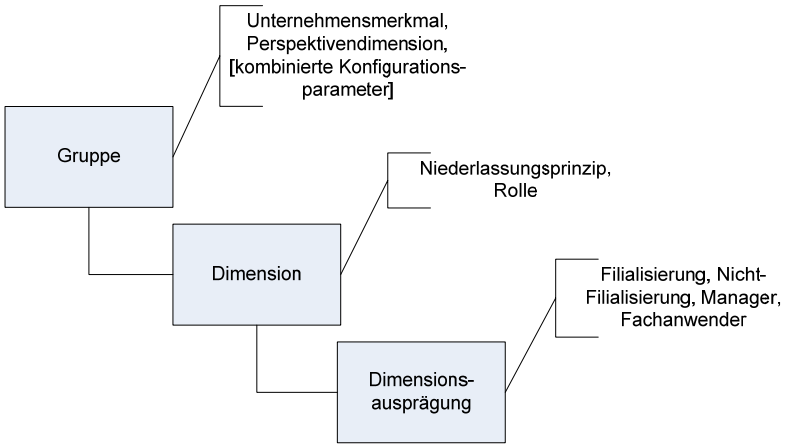



**Abb. 26:** Struktur der Konfigurationsparameter

Die kombinierten Konfigurationsparameter bilden einen Sonderfall, da sie sich nicht in diese Struktur von Dimensionen und Dimensionsausprägungen einfügen lassen. Sie kapseln zwar ähnlich zu Dimensionen Dimensionsausprägungen, verhalten sich jedoch nicht wie diese, sondern sind eher selbst als Dimensionsausprägungen anzusehen. Das Konstrukt der kombinierten Konfigurationsparameter ist notwendig, um Schnittmengen abbilden zu können: (A,C) $\cap$ (B,C) = C.

Wie in Abb. 26 ersichtlich ergibt sich für die Konfigurationsparameter eine Baumstruktur, die schon aus den Modellierungsmethoden des ursprünglichen H2-Toolsets bekannt ist. Aus Gründen der Softwareergonomie werden daher diese Modellierungstechniken für die Eingabe und Pflege der Konfigurationsparameter wiederverwendet. Es existiert eine Sprache bzw. ein Modelltyp für die Verwaltung der Konfigurationsparameter. Diese ist, wie jede Sprache des H2-Toolsets, editierbar. In jedem Fall sollte es für den Benutzer der Software unmöglich sein, Änderungen an der Sprache vorzunehmen. Bei der Anlage eines neuen Referenzmodells werden die Sprache der Konfigurationsparameter und ein Standardmodell von Konfigurationspa-

rametern automatisch angelegt. Es kann jedoch auch weiterhin vollständig auf Konfigurationsaspekte verzichtet werden.

Da das Standardmodell von Konfigurationsparametern nicht zwingend für die Domäne des zu erstellenden Referenzmodells ausreichend ist, kann dieses Modell erweitert und verändert werden. Weiterhin kann der Referenzmodellersteller vor und während der Modellierung den Fokus des Referenzmodells und seiner potenziellen Nutzer einschränken. Nach Abschluss der Eingabe von Konfigurationsparametern charakterisiert der Referenzmodellersteller die Modelldomäne durch Angabe der für sie relevanten Konfigurationsparameter. Er wählt diejenigen aus, die in seinem Modell Verwendung finden und für Konfigurationen verwendet werden können. Dieser Vorgang ist als „Vorauswahl“ im Referenzmodellierungs-Workflow angegeben. Die „vorausgewählten“ Konfigurationsparameter können anschließend im Termeditor für die Formulierung von Termen verwendet werden. Weiterhin können Konfigurationsparameter Benutzern zugeordnet werden, um ihnen im Zusammenspiel mit Termen eine subjektivierte Sicht auf das Gesamtmodell zu ermöglichen. Ähnlich kann mit der Auswahl von Konfigurationsparametern für die Konfiguration aus dem bestehenden Referenzmodell ein neues angepasstes Referenzmodell automatisch abgeleitet werden.

Zwischen den einzelnen Konfigurationsparametern können Beziehungen bestehen. Die Auswahl des einen Konfigurationsparameters kann bedeuten, dass ein weiterer zwingend auch ausgewählt werden muss oder nicht ausgewählt werden darf, weil sich die beiden Konfigurationsparameter bspw. Unternehmensmerkmalsausprägungen ausschließen. Das Programm unterstützt den Benutzer bei der Beachtung solcher Ontologien. Der Ersteller der Konfigurationsparameter kann gleichzeitig etwaige Beziehungen der Form „A bedingt B“ oder „A schließt C aus“ anlegen und pflegen. Einmal angelegt weist das Programm den Benutzer auf Unstimmigkeiten bei der Verwendung von Konfigurationsparametern hin, die aus Verstößen gegen diese Regeln bzw. Beziehungen resultieren.

In Tab. 15 sind alle für den Benutzer relevanten Programmbausteine aufgelistet und angegeben, welche Regeln bzw. Ontologie dort vom Programm beachtet wird und warum. Die Programmbausteine sind aus Abb. 26 entnommen.

| Programmbaustein | Ontologie | | Erläuterung |
|---|---|---|---|
| | „Bedingung“ | „Ausschluss“ | |
| Vorauswahl | Ja | Nein | Ein Referenzmodell kann auch Teilmodelle enthalten, die sich gegenseitig ausschließen. |
| Termeditor | Nein | Ja | Ein Term kann Teilelemente einer Bedingung enthalten, jedoch keine Elemente, die UND-verknüpft sind und sich gegenseitig ausschließen. |
| Benutzerverwaltung | Ja | Ja | Ein Benutzer bekommt ein vollständig konsistentes Modell angezeigt, also keine sich ausschließenden Modellteile gleichzeitig. |
| Konfiguration | Ja | Nein | Aus der Konfiguration entsteht wieder ein Referenzmodell, welches ex definitionem sich gegenseitig ausschließende Teilmodelle beinhalten kann |

**Tab. 15:** Berücksichtigung von Ontologien in Programmbausteinen

### 3.3.2 DV-Konzept und Implementierung

Programmnutzer können mit dem erweiterten Modell-Editor für Konfigurationsparameter das bestehende Modell editieren. Diese Änderungen schlagen sich in der Datenbank sowohl in den Tabellen, die für die Speicherung von Modellinformationen zuständig sind, als auch in den Tabellen, die von der Programmlogik verwendet werden, nieder. In Abb. 27 ist dargestellt, wie Tabellen der Datenbank mit Programmbausteinen, die Konfigurationsparameter verwenden, interagieren.

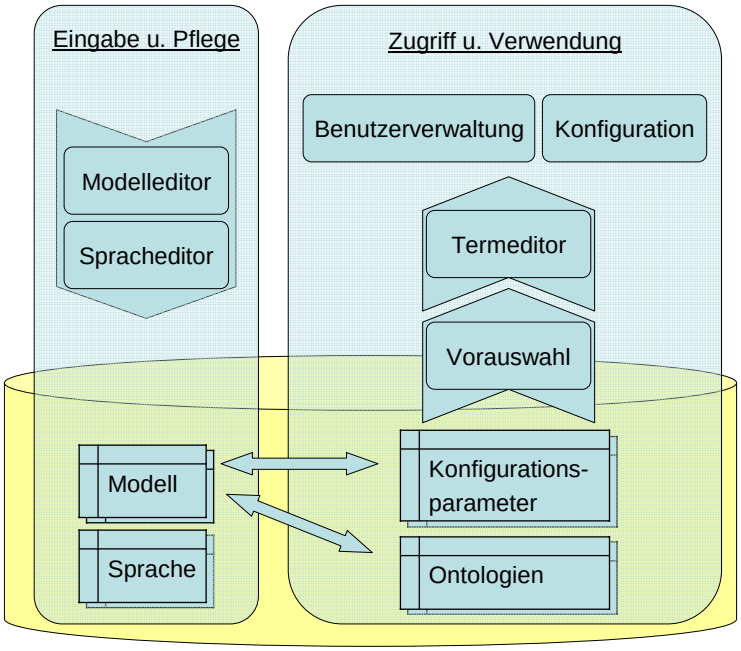



**Abb. 27:** Workflow zur Verwendung von Konfigurationsparametern

Die grafische Anordnung dieser Bausteine ist an den Referenzmodellierungs-Workflow aus Kapitel 2.1 angelehnt. Zunächst werden Konfigurationsparameter mit dem Modell-Editor eingegeben und bearbeitet. Dabei werden sie als Modell in den entsprechenden Tabellen der Datenbank abgespeichert. Gleichzeitig erfolgt die Pflege der in der Grafik rechts angeordneten Tabellen für Konfigurationsparameter und Ontologien im Modell-Editor.

Einen Überblick über die benötigten Datenstrukturen bietet das ER-Modell in Abb. 28. Die grau hinterlegten Entities kapseln die Konfigurationsparameter und Ontologien. Die Entitytypen *ObjectOccurence* und *ObjectDefinition* sind dieselben, die auch für Modelle benutzt werden. Hieran wird ersichtlich, dass Konfigurationsparameter als Modell und eigene Datenstruktur abgelegt werden. Damit bildet der Entitytyp *ConfigurationParameterDefinition* das Pendant zu *ObjectDefinition* sowie *ConfigurationParameter* zu *ObjectOccurence.* Jedes sichtbare Element eines Modells ist eine eigene *ObjectOccurence*. Damit kann mit den *Occurences* bzw. den Elementen von *ConfigurationParameter* die hierarchische Struktur abgelegt werden. Die Beziehung von *ConfigurationParameter* zu *ConfigurationParameterDefinition* ist damit begründet, dass jeder *ConfigurationParameter* durch eine *ConfigurationParameterDefinition* definiert wird. Dies ist wiederum deckungsgleich mit den Beziehungen zwischen *ObjectOccurence* und *ObjectDefinition*, wie sie aus den Modellen bekannt sind. Eine *Rule* bzw. Ontologie definiert eine Beziehung zwischen zwei *ConfigurationParameterDefinitions*. Ihre Beziehung zu *ConfigurationParameterType* legt fest, ob es sich um eine „Bedingungs“- oder „Ausschluss“-Regel handelt. Nachdem die Konfigurationsparameter in das System eingetra-

gen sind, können sie verwendet werden. Die *ConfigurationParameterDefinitions* können mit dem Attribut „isSelected“ einem Referenzmodell zugeordnet werden. Diese „vorselektierten“ Konfigurationsparameter können in Termen verwendet und einzelnen Benutzern des Referenzmodells zugeordnet werden. Ist ein Konfigurationsparameter nicht „vorselektiert“ worden, kann er im weiteren Programmablauf nicht verwendet werden.

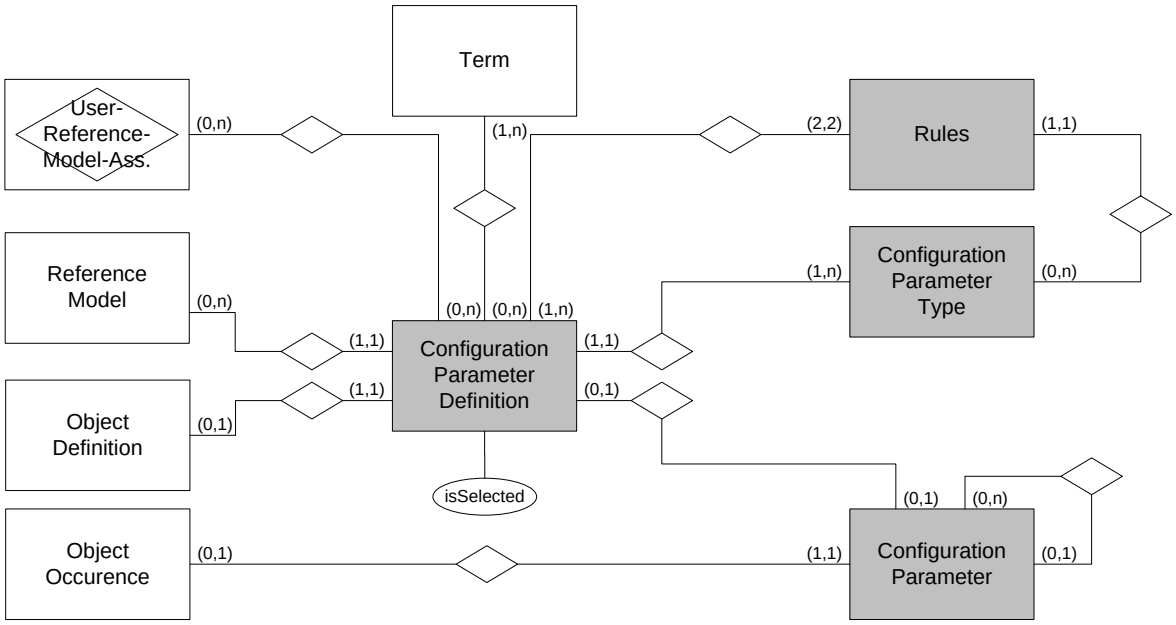



**Abb. 28:** ER-Diagramm der Konfigurationsparameter

Die Einträge in der Tabelle *ConfParam* dienen dazu, die hierarchische Struktur der Konfigurationsparameter zu speichern. Der einzelne Eintrag wird in der Tabelle über die *objectOccurenceId* identifiziert. Das Feld *parentId* nimmt die o*bjectOccurenceId* eines etwaigen Vaterelementes auf. Ist keines vorhanden hat dieses Feld den Wert „-1“ zugeordnet. Die *confParamDefinitionId* ist ein Fremdschlüssel, der einen Datensatz in der Tabelle *Confparamdefinition* (ET: *ConfigurationParameterDefinition)* referenziert. Dies ist notwendig, da nur die Definition Felder wie Name oder Beschreibung enthält. Das Feld *refmodelId* gibt an, welchem Referenzmodell dieser Datensatz zugeordnet ist.

| **Feld** | **Typ** | **Beispiel** | **Erläuterungen** |
|---|---|---|---|
| confParamDefinitionId | int(11) | 1 | Verweis auf Element der Tabelle Confparamdefinition |
| parentId | int(11) | 1 | Dient der Hierarchiespeicherung |
| refmodelId | int(11) | 1 | |
| objectOccurenceId | int(11) | 1 | Ist identisch mit id aus der Tabelle ObjectOccurence |

**Tab. 16:** Tabelle ConfParam

In der Tabelle Confparamdefinition werden die einzelnen Datensätze über das Feld *confParamDefinitionId* identifiziert. Im Programm wird dem Benutzer der Wert des Feldes *name* als Bezeichnung für einen Konfigurationsparameter angezeigt. Das Feld *description* wird in der momentanen Programmversion noch nicht benutzt, also auch nicht mit Werten gefüllt. Das Feld *definingConfParamId* referenziert einen Datensatz in der Tabelle *Confparam*, der gleichzeitig mit der *confParamDefinition* erzeugt wurde und damit ein „defining" Confparam wurde. Um zu speichern, welche Konfigurationsparameter in einem Referenzmodell zur Konfiguration verwendet werden können, wird das Feld *isSelected* benutzt. Im Programm wird der Wert in den Typ Boolean konvertiert. Ist der Wert also `true`, so ist der Konfigurationsparameter dem Referenzmodell zugeordnet und kann im weiteren Programmverlauf verwendet werden. Um in der Progammlogik herauszufinden, um welchen Typ von Konfigurationsparameter es sich handelt, oder ob es vielleicht auch eine Ontologie ist, werden die Datensätze in der Tabelle *Confparamtype* benutzt. Diese wird in *Confparamdefinition* über das Feld *confParamTypeId* referenziert. Weiterhin wird mittels *refModelId* abgespeichert, welchem Referenzmodell der Konfigurationsparameter zugehörig ist.

| **Feld** | **Typ** | **Beispiel** | **Erläuterungen** |
|---|---|---|---|
| definingConfparamId | int(11) | 1 | Verweis auf die Tabelle Confparam |
| refmodelId | int(11) | 1 | Zugehöriges Referenzmodell |
| confparamDefinitionId | int(11) | 1 | Ist identisch mit einer Objectdefinitionid |
| name | varchar(100) | 1 | Bezeichnung |
| description | varchar(100) | 1 | wird noch nicht benutzt |
| isSelected | tinyint(1) | 1 | Gibt an, ob der Parameter für das Referenzmodell ausgewählt wurde. |
| confparamTypeId | int(11) | 1 | Verweis auf Tabelle Confparamtype |

**Tab. 17:** Tabelle Confparamdefinition

Im Gegensatz zu den übrigen in diesem Abschnitt beschriebenen Tabellen ist die Tabelle *Confparamtype* mehr oder minder unveränderlich (vgl. Tab.18). Die in Tab. 19 aufgelisteten Einträge sollten nicht verändert werden. Die Einträge von *Confparamtype* bilden die Verbindung mit der Sprache der Konfigurationsparameter. Eine Veränderung an dieser Stelle muss damit auch zwingend mit einer Änderung an der Sprachdefinition einhergehen. Weiterhin stützt sich die Programmlogik auf die Werte dieser Tabelle. Insbesondere ist darauf zu achten, dass die Werte des Feldes *name* identisch mit den Bezeichnungen der Objekttypen sind, die im Spracheditor für die Sprache der Konfigurationsparameter stehen.

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| image | blob | <BLOB> | Bild |
| mnemonic | varchar(10) | UM | Abkürzung |
| type | int(11) | 1 | Eine von 4 Typ-Ids |
| isconfparam | tinyint(1) | 1 | Bei Ontologie ist es 0 |
| name | varchar(100) | Unternehmensmerkmal | Bezeichnung |
| id | int(11) | 1 | Eine von 8 Ids |

**Tab. 18:** Tabelle Confparamtype

| image | mnemonic | type | id | isconfparam | name |
|---|---|---|---|---|---|
| | KK | 3 | 1 | 0 | Kombinierte Konfigurationsparameter |
| | UM | 1 | 2 | 0 | Unternehmensmerkmal |
| | UMA | 2 | 3 | 1 | Unternehmensmerkmalsausprägung |
| | PD | 1 | 4 | 0 | Perspektivendimension |
| | P | 2 | 5 | 1 | Perspektive |
| | A | 0 | 6 | 0 | Ausschluss |
| | B | 0 | 7 | 0 | Bedingung (asymmetrisch) |

**Tab. 19:** Daten der Tabelle Confparamtype

Die Tabelle ist jedoch in der Hinsicht erweiterbar, als dass zusätzlich zu den Paaren Perspektivendimension/Perspektive und Unternehmensmerkmal/Unternehmensmerkmals-ausprägung weitere hinzugefügt werden können. Im Fall einer solchen Erweiterung ist es jedoch notwendig im Spracheditor der Konfigurationssprachen für die Erweiterungen zusätzliche Kontexte zu modellieren, die strukturgleich mit den bereits bestehenden Paaren sind. Ein Beispiel findet sich im Abschnitt „Eingabe und Pflege“ in diesem Kapitel.

Ontologien werden in der Tabelle *Rules* abgespeichert. Primärschlüssel dieser Tabelle ist der Wert des Feldes *objectDefinitionId*. Um unterscheiden zu können, um welche Art von Ontologie es sich handelt, wird die Tabelle *Confparamtype* durch das Feld *confparamTypeId* referenziert. In Abb. 29 ist dargestellt, wie die Struktur von Ontologien aufgebaut ist.

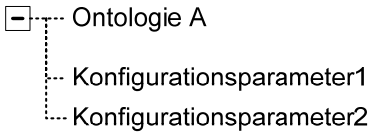



**Abb. 29:** Struktur der Ontologien

Die Felder, deren Bezeichnung identisch mit der aus den bereits genannten Tabellen ist, erfüllen die gleiche Funktion wie diese. Auf eine weitere Erläuterung derselben wird daher verzichtet. Zu beachten ist jedoch, dass es sich bei den Werten der Felder *confparamDefinitionId1* und *confparamDefinitionId2* um Id’s der Tabelle *Confparamdefinition* handelt. Die damit identifizierbaren Konfigurationsparameter werden in dieser Tabelle in Beziehung zueinander gesetzt. Da es sich immer um eine asymmetrische Beziehung handelt, ist die Beziehungsrichtung immer von *confparamDefinitionId1* zu *confparamDefinitionId2*. Werden kom-

binierte Konfigurationsparameter modelliert, wird automatisch für jeden Bestandteil des kombinierten Konfigurationsparameters eine „Bedingt“-Beziehung erzeugt. Die damit automatisch entstandene Regel wird über den Wert `true` im Feld generated gekennzeichnet und ist für den Benutzer durch die Bezeichnung „Generiert von [Bezeichnung des kombinierten Konfigurationsparameters]“ erkennbar.

| **Feld** | **Typ** | **Beispiel** | **Erläuterungen** |
|---|---|---|---|
| objectDefinitionId | int(11) | 1 | Id, die das Element in den Modelltabellen hat |
| name | varchar(100) | E-commerce bedingt Versandhandel | Bezeichnung |
| description | varchar(100) | | Wird noch nicht verwendet |
| confparamTypeId | int(11) | 5 | Verweis auf Tabelle Confparamtype |
| confparamDefinitionId1 | int(11) | 1 | Verweis auf Element in Confparamdefinition |
| confparamDefinitionId2 | int(11) | 2 | Verweis auf Element in Confparamdefinition |
| refmodelId | varchar(100) | 1 | Id de Referenzmodells |
| generated | tinyint(1) | 0,1 bzw. true, false | „Generated“ sind Regeln, die durch das Anlegen von kombinierten Konfigurationsparamtern entstehen. |

**Tab. 20:** Tabelle Rules

Zur Eingabe und Pflege von Konfigurationsparametern wurde die Klasse *EditorConfParam* geschrieben. Diese erbt von *Editor*, welche für den Modell-Editor zuständig ist. *EditorConfParam* stellt einen Editor zur Verfügung, mit dem zum einen ein normales Modell in einer Sprache erstellt werden kann und zum anderen die eigentlichen Konfigurationsparameter und Ontologien gepflegt werden.

*Konfigurationsparameter*

Wird ein Konfigurationsparameter angelegt, ruft *EditorConfParam* den Konstruktor der jeweiligen Klasse auf. Im Konstruktor wird das Objekt erzeugt und in ein *Dictionary* im aktuellen bzw. übergebenen Refmodel-Objekt eingetragen. Auf dieses *Dictionary* wird aus den weiteren Programmteilen bei Bedarf zugegriffen. Wie Konfigurationsparameter angelegt werden dürfen ist außerdem durch die Sprache festgelegt. Das Modell der Sprache ist in Abb. 30 in der H2-Toolset Notation dargestellt.

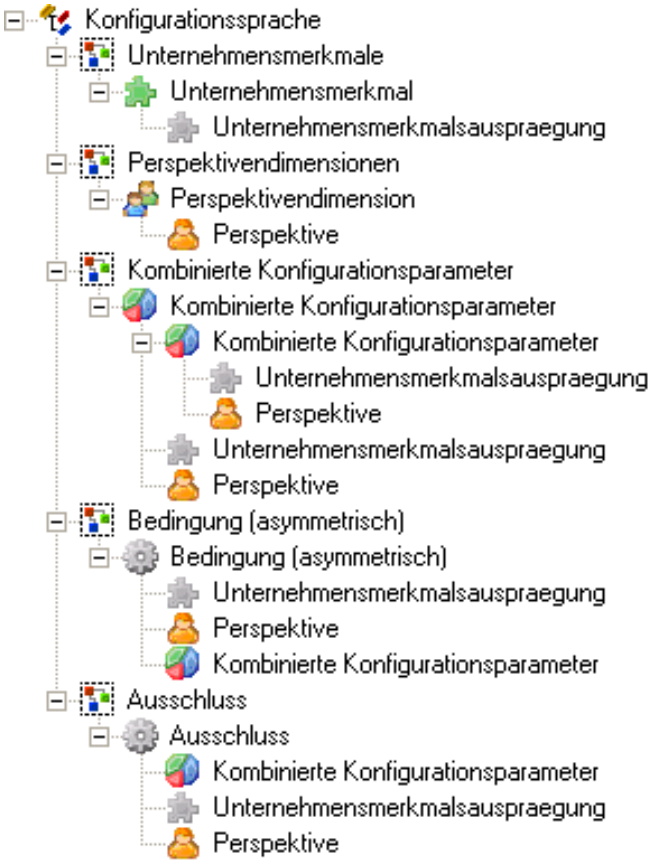



**Abb. 30:** Metamodell der Konfigurationsparameter und Ontologien

Tab. 21 gibt einen Überblick über alle Klassen, die mit den Konfigurationsparametern bzw. Ontologien in Verbindung stehen.

| **Klasse** | **Erläuterungen** |
|---|---|
| EditorConfParam | Editor zur Eingabe und Pflege |
| Confparam | enthält den Konstruktor, Zugriffsmethoden und Datenbankzugriff |
| Confparamdefinition | enthält den Konstruktor, Zugriffsmethoden und Datenbankzugriff |
| Confparamtype | enthält den Konstruktor, Zugriffsmethoden und Datenbankzugriff |
| Rules | Oberklasse zu Implication und Exclusion |
| Implication | Erbt von Rule und implementiert die beiden Ontologien „Auschluss“ und „Bedingung“ |
| Exclusion | |
| RefModel | Enthält Dictionaries und typisierte Listen mit der Konfigurationsparameter und Ontologien. |
| UserConfparamAssignment | Ist für die Zuweisung von Konfigurationsparametern zu Benutzern eines Referenzmodells zuständig |
| Preselection | Ist für die Zusweisung von Konfigurationsparametern zu einem Referenz-modell zuständig |

**Tab. 21:** Klassen für die Konfigurationsparameter

***Eingabe und Pflege***

*Konfigurationsparametersprache*

**Die Veränderung bzw. Erweiterung der Sprache der Konfigurationsparameter und Ontologien stellt einen schwerwiegenden Eingriff in das Programm dar. Sollte dies nicht exakt nach den in diesem Abschnitt erläuterten Anweisungen erfolgen, muss der Benutzer damit rechnen, dass konfigurierbare Modelle inkonsistent werden.**

Folgende Anpassungen können vorgenommen werden:

Durch Anklicken eines Objekttypen öffnet sich im rechten Teil des Spracheditors eine „Eigenschaften“-Box. Dort kann im Gliederungspunkt „Symbol“ eine andere Bilddatei für den Objekttyp ausgewählt werden. Im Spracheditor und Modell-Editor wird dieses neue Symbol direkt übernommen. Um das neue Symbol an allen Stellen im Programm verwenden zu können ist es nötig, einen Konfigurationsparameter anzulegen oder zu editieren.

Die Bezeichnung eines Objekttypen wird ebenfalls in der „Eigenschaften“-Box des Spracheditors im Gliederungspunkt „Name“ angezeigt. Dort kann sie auch direkt editiert werden. Ist dies geschehen, sollte das Programm bzw. die Datenbankverbindung geschlossen werden und die Tabelle *Confparamtype* in der Datenbank manuell editiert werden. Jeder Objekttyp des Spracheditors hat dort einen entsprechenden Eintrag, der über seine Bezeichnung identifiziert werden kann. Dieser Eintrag in der Spalte „name“ muss den exakt identischen String enthalten, wie er auch im Spracheditor für den Objekttypen eingegeben wurde.

Die Sprache der Konfigurationsparameter kann um weitere Kontexte erweitert werden. Diese müssen allerdings der Struktur entsprechen, wie sie für Unternehmensmerkmale oder Perspektivendimensionen gilt.

Es muss also ein neuer Kontext angelegt werden. In diesem Kontext müssen Objekte eines neuen Dimensionsobjekttypen angelegt werden können. Dessen Instanziierung ist auf „CreateDefinition“ zu setzen. Unter den Dimensionsobjektypen ist in der Sprache ein Ausprägungsobjekttyp zu modellieren, dessen Instanziierung ebenfalls auf „CreateDefinition“ zu setzen ist. Sollen kombinierte Konfigurationsparameter mit dem neuen Objekttyp gebildet werden können, müssen ebenfalls Änderungen im Kontext „Kombinierte Konfigurationsparameter“ erfolgen. Ein kombinierter Konfigurationsparameter kann den neuen Ausprägungsobjekttyp mit der Instanziierung „CreateOccurence“ enthalten. Gleiches gilt für einen kombinierten Konfigurationsparameter, der bereits einem kombinierten Konfigurationsparameter untergeordnet ist. Die Kontexte der Regeln müssen ebenfalls erweitert werden. Es sollte möglich sein, unter ein „Ausschluss“- und „Bedingungs“-Objekt ein Objekt des neuen Ausprä-

gungsobjekttypen zu modellieren. Nachdem die Erweiterungen abgeschlossen sind, sollte sich der Spracheditor wie in Abb. 31 darstellen. Die Erweiterungen bestanden darin, einen neuen Kontext „Musterkontext“ mit den Objekttypen „Musterdimension“ und „Musterausprägung“ zu modellieren. Objekte der „Musterausprägung“ sollen später im Programm als zusätzliche Konfigurationsparameter dienen.

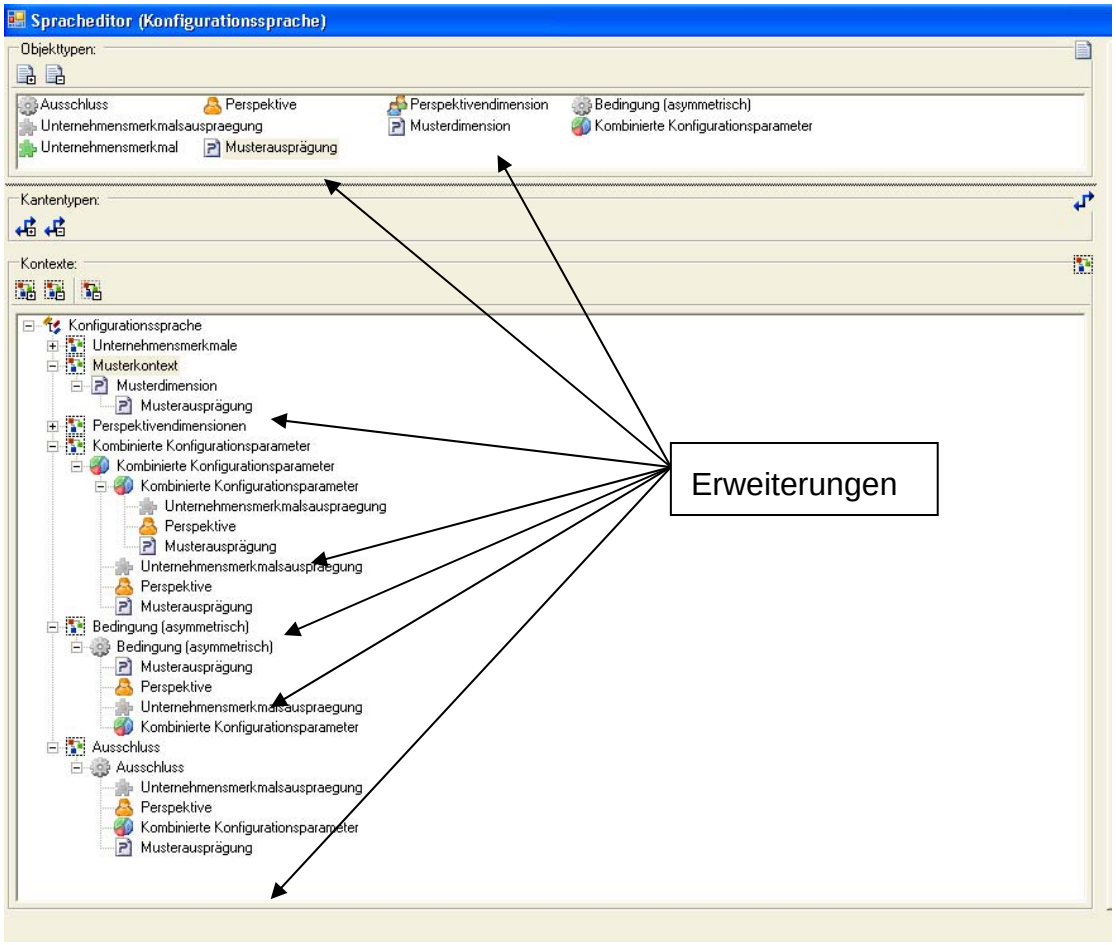



**Abb. 31:** Beispiel der Erweiterung der Konfigurationssprache

*Konfigurationsparametereditor*

Der Editor zur Eingabe und Pflege von Konfigurationsparametern wird aus dem Datenbank-Explorer aufgerufen. Dort befindet sich unterhalb eines jeden Referenzmodells ein Knoten für diesen Editor. Die Funktionsweise ist eng an den Modell-Editor des H2-Toolsets angelehnt. Normalerweise bleiben die Erweiterungen des Editors für den Benutzer transparent und werden daher an dieser Stelle nicht weiter erläutert. Jedoch sind für die Eingabe von kombinierten Konfigurationsparametern und Ontologien einige Besonderheiten zu beachten. Bei der Modellierung der ersteren wird für jeden Bestandteil eines kombinierten Konfigurationsparameters eine „Bedingungs“-Ontologie erzeugt. Sie befindet sich dementsprechend im Kontext

„Bedingung“. Die automatisch generierten Ontologien können nicht manuell editiert oder gelöscht werden. Sie lassen sich jedoch mittels eines Buttons in der Menüleiste des Editors aus- und einblenden. Soll eine Ontologie hingegen manuell erstellt werden, ist darauf zu achten, in welcher Reihenfolge sich die beiden Konfigurationsparameter befinden, zwischen denen eine Beziehung modelliert wird. Die Beziehungsrichtung geht immer vom oberen bzw. ersten zum unteren bzw. zweiten Konfigurationsparameter aus. In Abb. 32 ist dargestellt, welche automatisch generierten Ontologien bei der Modellierung eines kombinierten Konfigurationsparameters entstehen.

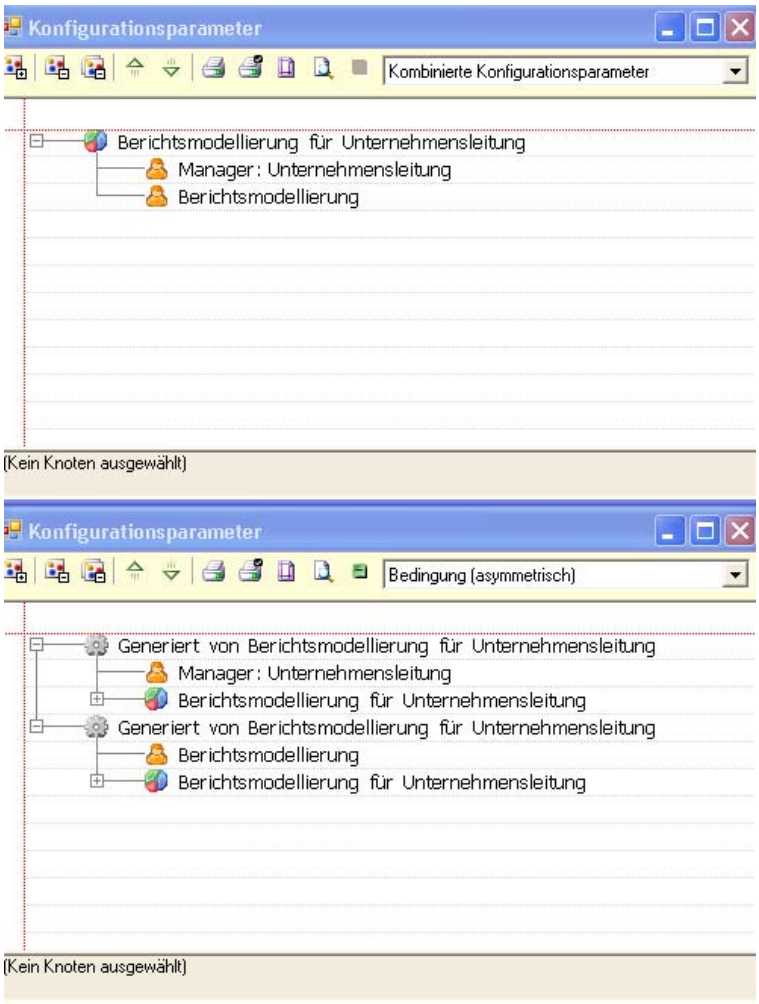



**Abb. 32:** Automatisch generierte Ontologien im Modell-Editor für Konfigurationsparameter

Beim Löschen von Konfigurationsparametern ist der Benutzer einigen Restriktionen unterworfen. Dies trifft insbesondere dann zu, wenn der zu löschende Konfigurationsparameter bereits an unterschiedlichen Stellen im Programm verwendet wird. In einem solchen Fall ist das Löschen nicht möglich und der Benutzer wird vom Programm auf die Ursache hingewiesen.

Die Überprüfung von Ontologien erfolgt in den verschiedenen Teilen des Programms unterschiedlich. Eine Übersicht und Erklärung darüber, wann welche Ontologie beachtet werden muss, gibt Tab. 15.

*Vorauswahl von Konfigurationsparametern*

Ein Referenzmodell wird für eine Domäne erstellt. Die Charakteristika dieser Domäne werden durch die Konfigurationsparameter abgebildet. Da die Domäne, für die ein Referenzmodell erstellt wird nicht alle im Programm zur Verfügung stehenden Konfigurationsparameter umfasst, müssen die relevanten Konfigurationsparameter vom Referenzmodellersteller selektiert werden. Dies geschieht über den Dialog der „Konfigurationsparameter-Vorselektion“, wie er in Abb. 33 dargestellt ist. Im Fenster „Konfigurationsparameter“ sind alle zur Verfügung stehenden Konfigurationsparameter aufgelistet. Der Referenzmodellersteller kann durch die Selektion der gewünschten Konfigurationsparameter diese dem Referenzmodell zuordnen. Dabei muss er etwaige hinterlegte „Bedingungs“-Ontologien beachten. Diese werden im Fenster Ontologien für den gerade selektierten Konfigurationsparameter angezeigt. Nur die vorselektierten Konfigurationsparameter können im späteren Programmablauf bspw. für Terme verwendet werden. Wählt der Benutzer einen kombinierten Konfigurationsparameter aus, so wird dessen Zusammensetzung im Feld „Kombination aus“ angezeigt.

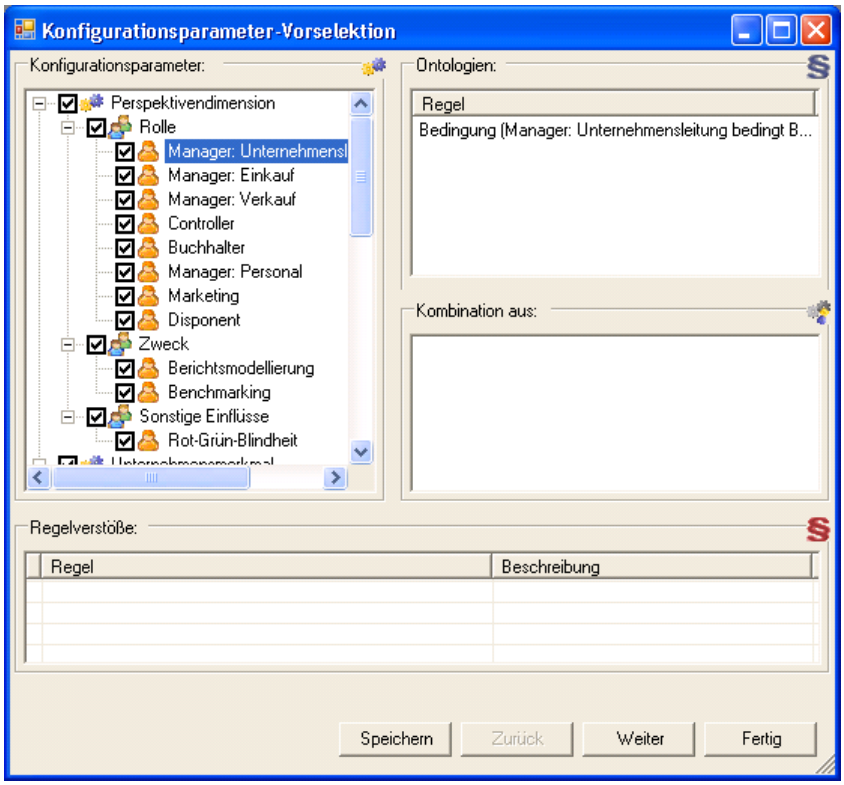



**Abb. 33:** Vorselektion von Konfigurationsparameter für ein Referenzmodell

*Zuordnung zu Benutzern*

Um H2-Nutzern eine individualisierte Sicht auf die im H2-Repository gespeicherten Modelle zu gewähren, sind ihnen referenzmodellspezifisch Konfigurationsparameter zuzuordnen. Je nach Verwendungskontext lassen sich H2-Modelle auf diese Weise in Abhängigkeit der den Nutzern zugeordneten Konfigurationsparameter aufbereiten, z. B. durch eine Reduktion des Umfangs.

Die Zuordnung von Konfigurationsparametern zu Benutzern erfolgt im Dialog „Konfigurationsparameter zuweisen“. Dieser wird über „Datenbank → Konfigurationsparameter Benutzer zuordnen“ oder durch einen rechten Mausklick auf das Referenzmodell geöffnet. Zu beachten ist, dass das Referenzmodell, für welches die Zuweisung erfolgen soll, aktiviert ist.

Für die Zuordnung von Konfigurationsparameter zu Benutzern muss zunächst ein Benutzer ausgewählt werden, für den die Zuordnung erfolgen soll. Dazu sind im rechten Teil des Fensters alle Benutzer aufgelistet, die dem betrachteten Referenzmodell zugeordnet wurden. Wurde ein Benutzer ausgewählt, können nun analog zum Dialog der Vorselektion die Konfigurationsparameter zugewiesen werden.

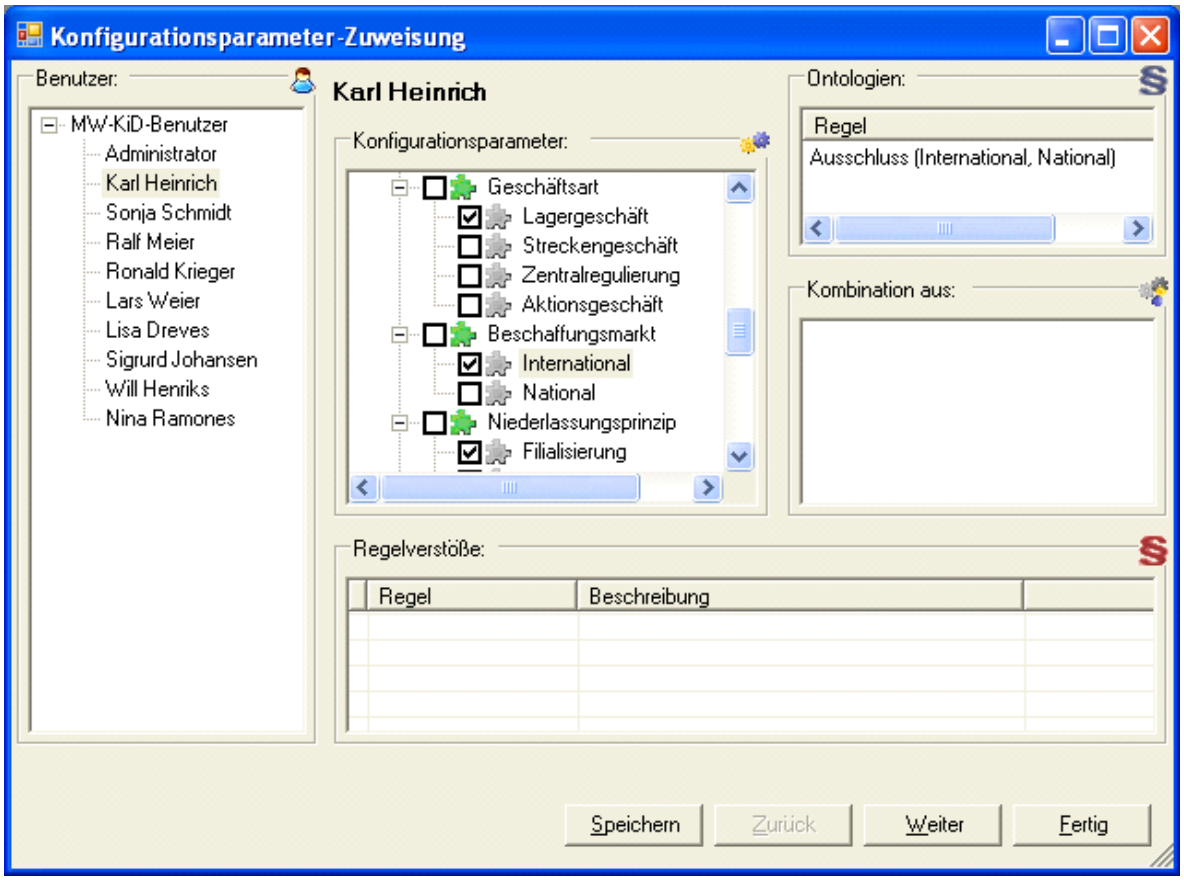



**Abb. 34:** Konfigurationsparameter-Zuweisung zu Benutzern

*Konfiguration*

Wurden in einem Referenzmodell sowohl Konfigurationsparameter angelegt, als auch seine Modelle mit Konfigurationsansatzpunkten hinterlegt, kann eine Konfiguration des Referenzmodells entsprechend der ausgewählten Konfigurationsparameter erfolgen. Durch die Konfiguration wird eine losgelöste Modellvariante des Referenzmodells erstellt, die anschließend vermarktet werden kann.

Die Konfiguration geschieht im Dialog „Konfiguration“. Dieser wird über „Datenbank → Modelladaption“ oder durch einen rechten Mausklick auf das Referenzmodell geöffnet. Wichtig ist, dass das Referenzmodell, für welches die Zuweisung erfolgen soll, aktiviert ist.

Die Auswahl der Konfigurationsparameter zur Konfiguration erfolgt analog zu der Auswahl der Konfigurationsparameter für die Vorselektion.

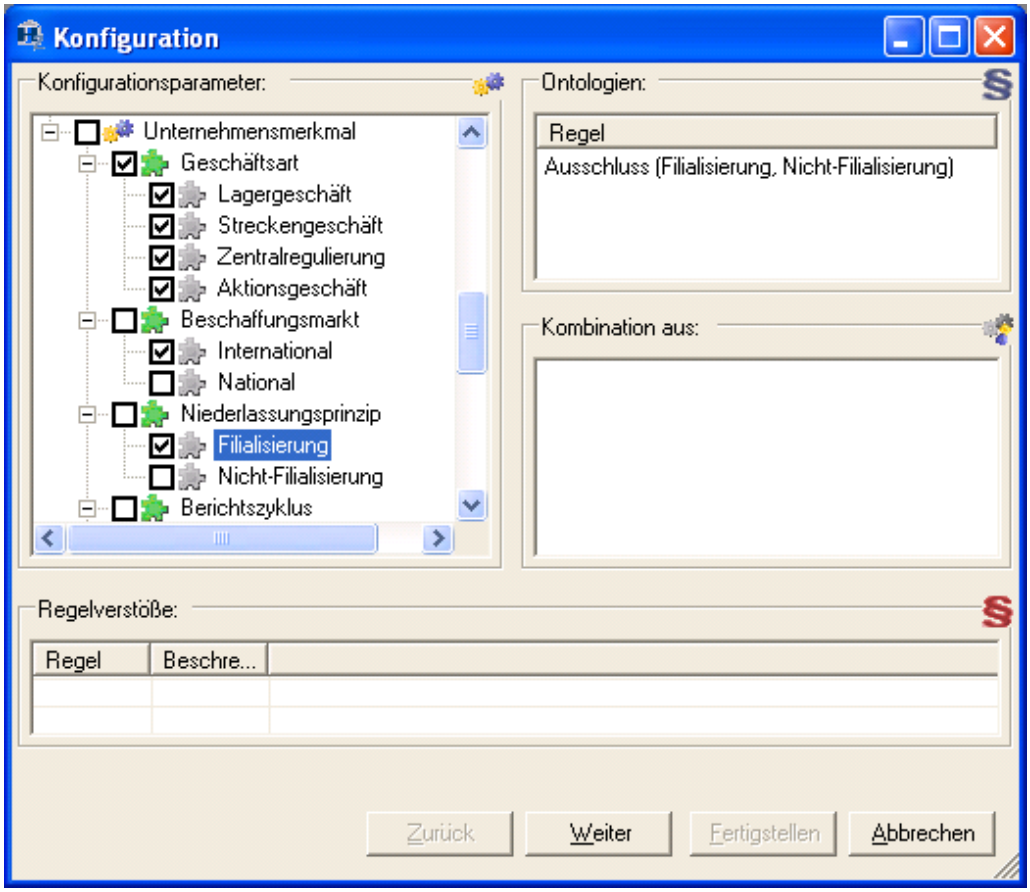



**Abb. 35:** Auswahl der Konfigurationsparameter zur Konfiguration

## 3.4 Erstellung und Verwaltung von Termen

### 3.4.1 Fachkonzept

Um die verschiedenen Konfigurationsmechanismen durch ein benutzerfreundliches und leicht erweiterbares Konzept zu realisieren, kommen boolesche Terme zum Einsatz. Diese Terme werden einmal definiert und können im Nachhinein für sämtliche Konfigurationsmechanismen verwendet werden. Dadurch ist es möglich, komplexe Sachverhalte einheitlich auszudrücken.

Bestandteile eines Terms sind in erster Linie Konfigurationsparameter sowie deren Ausprägungen. Zudem können bereits bestehende Terme in einem Term referenziert werden. Die verwendeten Konfigurationsparameter und referenzierten Terme werden durch boolesche Operatoren miteinander kombiniert. Ein Term kann mehreren Sprachen und Modellen sowie deren Elementen zugeordnet werden, um diese auszublenden bzw. zu konfigurieren. Der (boolesche) Term eines Elements muss zu `true` auswerten, damit dieses angezeigt wird (Elemente ohne zugewiesenen Term werden grundsätzlich angezeigt).

Ob ein Term zu `true` oder `false` auswertet, hängt von drei Faktoren ab: Den Konfigurationsparametern, die dem aktuellen Benutzer zugeordnet sind, den im Term verwendeten Konfigurationsparametern und referenzierten Termen sowie den Operatoren, mit denen die Konfigurationsparameter und Terme miteinander verknüpft sind. Mögliche n-näre Operatoren sind `And`, `Or` und `Xor`, die mehrere Konfigurationsparameter und / oder Terme miteinander kombinieren. Der unäre Operator `Not` bezieht sich lediglich auf einen einzelnen Ausdruck (Konfigurationsparameter, Term oder weiteren Operator). Die Konfigurationsparameter selbst werden in der Form `Typ.Parameter is Ausprägung` angegeben – z. B. `UM.Absatzmarkt is International`. Neben dem üblichen `Is-Ausdruck` gibt es noch den `One-of-Ausdruck`, der mehrere Ausprägungen beinhalten kann, beispielsweise `UM.Geschäftsart one of {Lagergeschäft, Aktionsgeschäft}`. Rein logisch stellt ein `One-of-Ausdruck` dasselbe dar wie eine Kombination mehrerer `Is`-Ausdrücke mit dem `Or`-Operator. Das vorherige Beispiel ist also äquivalent zu `UM.Geschäftsart is Lagergeschäft or UM.Geschäftsart is Aktionsgeschäft`:

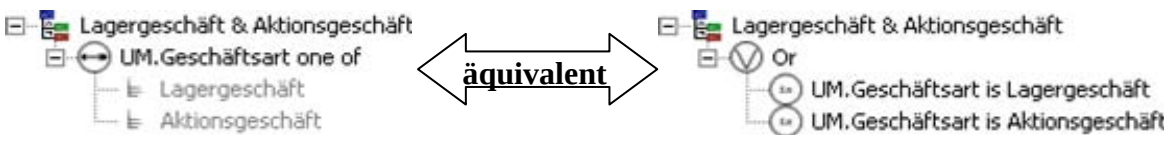



**Abb. 36:** Gegenüberstellung von One-of-Ausdruck und Or-Ausdruck

### 3.4.2 DV-Konzept und Implementierung

Terme sind immer genau einem Referenzmodell zugeordnet, demnach besitzt ein Term – neben seiner eigenen ID – die ID des Referenzmodells. Als spezifische Informationen werden die Termbezeichnung, ggf. eine Beschreibung und der Termausdruck in Textform in der Datenbank hinterlegt (vgl. Tab. 22).

| Attribut | Typ | Beispiel | Erläuterung |
|---|---|---|---|
| id | int(11) | 1 | ID des Terms |
| refmodelId | int(11) | 1 | ID des Referenzmodells |
| name | text | AWS-Gestalter | Bezeichnung des Terms |
| termText | text | #t2#.#p458# is #p459# | Termausdruck in textueller Form |
| description | text | Blendet alle nicht rele... | Ausführliche Beschreibung |

**Tab. 22:** Tabelle Term

Da die Terme textuell in der Datenbank gespeichert werden, wird die Anzahl der Tabelleneinträge gering gehalten. Es ist nicht für jeden Knoten der Baumstruktur eines Terms ein eigener Eintrag nötig – pro Term reicht ein einziger Eintrag.

Jedoch sind die Terme in der Datenbank nicht so leserlich aufzufinden, wie sie im H2-Toolset dargestellt werden, denn es werden nicht die Bezeichnungen der Konfigurationsparameter und referenzierten Terme etc. hinterlegt, sondern jeweils die Datenbank-ID des entsprechenden Tabelleneintrages. Somit haben etwaige Änderungen der Bezeichnungen keine negativen Auswirkungen auf die Konsistenz der Terme. Bei Lese- und Schreibzugriffen müssen die Termtexte jedoch erst in ein entsprechendes Format konvertiert werden. Je nachdem, ob in die Datenbank geschrieben oder aus der Datenbank gelesen wird, müssen die Konfigurationsparameter etc. durch ihre jeweilige Datenbank-ID ersetzt werden oder umgekehrt. Die IDs sind von Rauten („#“) umgeben und werden von einem Präfix geführt. Das Präfix „t“ bedeutet die ID eines Konfigurationsparametertypen, „p“ steht für Konfigurationsparameter sowie Konfigurationsparameterausprägung. Andere referenzierte Terme erhalten als Präfix ein „r“. Weitere Arten können in Termen nicht auftreten. Sie werden demnach auch nicht unterschieden. Jedoch stellen spätere Erweiterungen keine Probleme dar.

Zur Verarbeitung eines Terms – ausgenommen Datenbankinteraktionen – existiert eine zentrale Klasse: „TermProcessing“. Hauptaufgaben der Klasse sind die Transformation von Textform in Baumform – und umgekehrt – sowie die Termauswertung zu `true` oder `false`. Zusätzlich existiert noch eine ganze Palette an Zusatzfunktionen wie beispielsweise Gültig-

keits- und Konsistenzchecks sowie die Konvertierung zwischen dem Rohformat und einem leserlichen Format[15].

*Text2Tree*

Die Methode *Text2Tree* transformiert die Textform eines Terms in eine Baumstruktur. Dafür wird der Text zunächst in so genannte Tokens zerlegt. Die so entstandene lineare Liste verwandelt sich im weiteren Verlauf sukzessiv in eine baumförmige Struktur, den Syntaxbaum (vgl. Abb. 37).

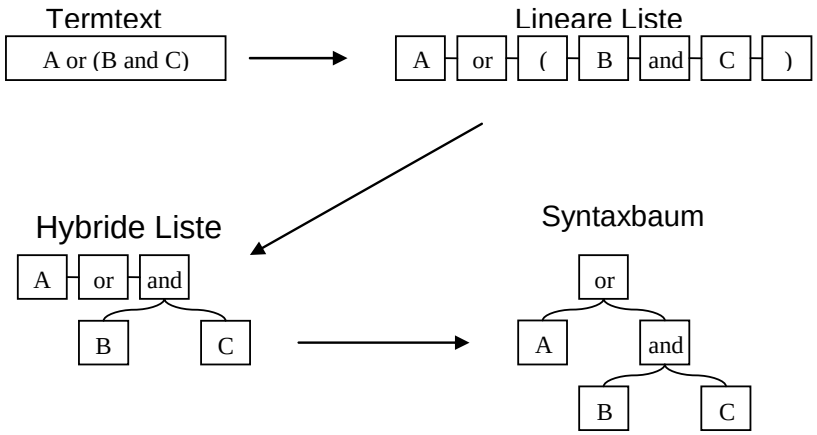



**Abb. 37:** Erzeugung des Syntaxbaumes eines Termtextes

Die Tokens werden hinsichtlich der Klammersetzung des Ausdruckes von innen nach außen abgearbeitet, bis alle Klammern aufgelöst wurden. Während dieses Prozesses wird bereits ein Großteil der linearen Liste durch die entsprechende Baumstruktur ersetzt. Innerhalb eines Klammerausdruckes sowie auf oberster Ebene der hybriden Liste werden die klammerlosen Tokens dann einer weiteren Verarbeitung unterzogen: Es wird geprüft, ob es sich um einen `And`-, `Or`- oder `Xor`-Ausdruck handelt.[16] Falls keiner der drei Fälle vorliegt wird noch geprüft, ob der vorliegende Ausdruck ein `Not`-Ausdruck ist. Genauso wird auch mit den Teilausdrücken eines `And`-, `Or`- bzw. `Xor`-Ausdruckes verfahren. Nach der Verarbeitung etwai-

[15] Unter Rohformat wird die Form verstanden, in der die Terme in der Datenbank hinterlegt werden: Die Bezeichnungen der referenzierten Konfigurationsparameter und Terme etc. werden durch ihre Datenbank-IDs ersetzt. Das Rohformat ist nicht besonders „leserlich“ – ein leserliches Format beinhaltet die korrekten Bezeichnungen.

[16] Eine Mischform wird nicht unterstützt: And-, Or- und Xor-Ausdrücke müssen also durch entsprechende Klammersetzung ineinander verschachtelt sein.

ger Not-Ausdrücke verbleiben lediglich atomare Ausdrücke, die noch nicht betrachtet wurden:[17]

- Wurzelknoten eines Teilbaumes der hybriden Liste (genau ein Token)
  → keine weitere Verarbeitung nötig.
- `Is`-Ausdruck (genau drei Tokens).
- `One-of`-Ausdruck (mindestens fünf Tokens).
- Referenzierter Term (genau ein Token).
- `true` (einzelnes Token).
- `false` (einzelnes Token).

*Tree2Text*

Die Methode *Tree2Text* transformiert die Baumstruktur eines Terms in eine Textform. Dabei wird der Baum – angefangen beim Wurzelknoten – rekursiv traversiert. Für jeden Knoten wird dann eine entsprechende Zeichenkette generiert, die durch weitere Kinderknoten beeinflusst wird: Der String eines `And`-Knotens muss zum Beispiel um die generierten Zeichenketten seiner Kinderknoten ergänzt werden. Ferner wird beispielsweise bei einem `Not`-Knoten der verschachtelte Teilstring des Kinderknotens mit Klammern umgeben, falls letzterer einen `And`-, `Or`- oder `Xor`-Ausdruck repräsentiert. Damit ist eindeutig festgelegt, dass sich der `Not`-Operator auf den gesamten Teilausdruck bezieht – nicht nur auf das erste Element. Ebenso werden die Klammern verschachtelter `And`-, `Or`- und `Xor`-Ausdrücke automatisch korrekt gesetzt.

*EvaluateTerm*

Die Methode *EvaluateTerm* wertet einen Term bzw. die Baumstruktur eines Terms zu `true` oder `false` aus. Dafür wird der Baum rekursiv traversiert. Die „inneren“ Knoten eines komplexen Termbaumes stellen grundsätzlich Operatoren dar (`And`, `Or`, `Xor` und `Not`). Lediglich die Blattknoten können direkt zu `true` oder `false` ausgewertet werden.[18] Die an den Blattknoten generierten Werte werden dann durch die Operatorenstruktur geschickt und somit einer weiteren Auswertung unterzogen. Am Wurzelknoten steht schlussendlich das Resultat der Termauswertung.

---

[17] Atomare Ausdrücke sind nicht zwangsläufig einzelne Tokens. Die Tokens eines atomaren Ausdruckes gehören jedoch untrennbar zusammen (z. B. die Kommata, geschweiften Klammern und Konfigurationsparameter eines One-of-Ausdruckes).

[18] Eine Ausnahme bildet der One-of-Knoten, der kein Blattknoten im eigentlichen Sinne ist, weil ihm die Knoten der Konfigurationsparameterausprägungen untergeordnet sind.

Bei jeder Aktion, die eine Auswertung, Transformation, Konvertierung oder sonstige Bearbeitung eines Terms mit sich bringt, findet automatisch eine Überprüfung auf Konsistenz des Terms und Gültigkeit der verwendeten Konfigurationsparameter statt.

Die Anwendung der einzelnen Konfigurationsmechanismen wird durch boolesche Terme gesteuert, deren Operanden Konfigurationsparameter sind. Diese Terme können unterschiedlichen Elementen des H2-Toolsets zugewiesen werden, deren Zustand sie dann beeinflussen können, wie z. B. Elemente, die ausgeblendet werden, wenn der zugewiesene Term zu `false` auswertet. Das Auswertungsergebnis eines Terms hängt davon ab, seine Operanden in der aktuellen Umgebung aktiv sind. Dies bedeutet beim Modellieren, dass der Konfigurationsparameter dem aktuellen Benutzer zugewiesen wurde, bei der Adaption hingegen, dass er im Adaptionsdialog selektiert wurde.

Um Terme anlegen, editieren und löschen zu können, wurde im H2-Toolset der Termeditor eingeführt. Er bietet eine übersichtliche, grafische Baumdarstellung, in denen Terme in wenigen Mausklicks editiert werden können, sowie die Möglichkeit zur Eingabe von Termen über die Tastatur.

Um den Termeditor starten zu können, muss sich der Benutzer angemeldet haben und das Funktionsrecht *FunctionRefmodelling* im aktivierten Referenzmodell besitzen. Sind diese Bedingungen erfüllt, so kann der Termeditor über die Menüleiste des H2-Toolsets unter „Ansicht→ Termeditor einblenden“, gestartet werden. Abb. 38 zeigt eine mögliche Ansicht des Termeditors.

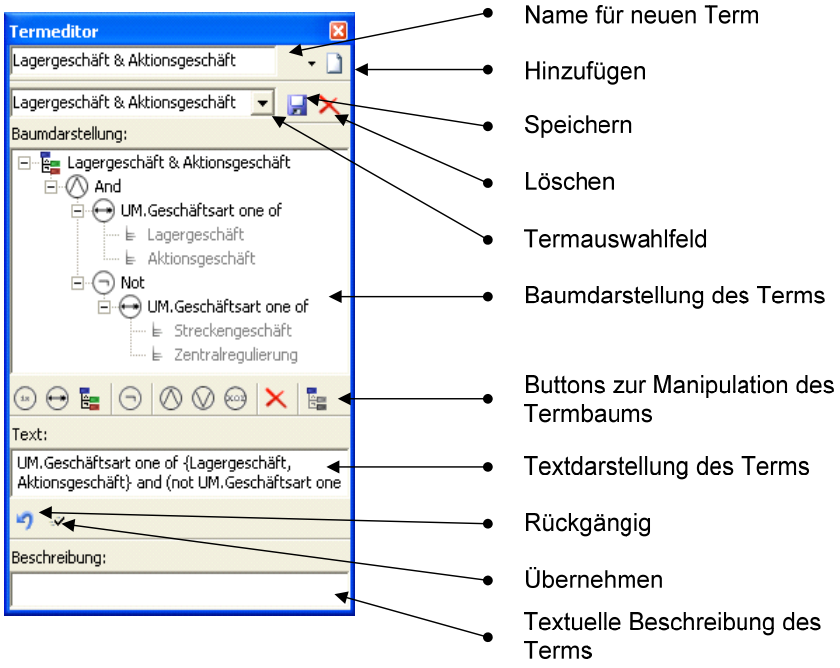



**Abb. 38:** Termeditor

*Anlegen eines neuen Terms*

Um einen neuen Term anzulegen, muss im oberen Textfeld ein (möglichst aussagekräftiger) Name eingegeben werden und daraufhin auf den Button „Hinzufügen“ geklickt werden. Dadurch wird ein neuer Term in der Datenbank abgelegt, der den eingegebenen Namen trägt. Initial wertet der neue Term immer zu `true` aus, bis er mit anderen Werten belegt wird.

Soll stattdessen ein bereits bestehender Term als Vorlage benutzt werden, muss dieser zunächst im Termauswahlfeld ausgesucht werden. Anschließend muss ein Name im entsprechenden Textfeld eingegeben werden und der Term wird über den Button „Übernehmen“ (erreichbar über den Pfeil neben dem „Hinzufügen“ Button) in die Datenbank eingefügt.

*Editieren eines Terms in der Baumansicht*

Wenn ein Term neu erstellt oder ein bereits bestehender Term über das Termauswahlfeld geöffnet wird, zeigt der Termeditor eine Baumansicht des entsprechenden Terms. Diese kann über die darunter befindlichen Buttons editiert werden. Die Buttons haben die in Tab. 23 aufgelisteten Bedeutungen.

| | |
|---|---|
|  | `IS`-Operator |
|  | `OneOf`-Operator |
|  | `Term`-Operator |
|  | `Not`-Operator |
|  | `And`-Operator |
|  | `Or`-Operator |
|  | `Xor`-Operator |
|  | Konstante `true` |
|  | Konstante `false` |

**Tab. 23:** Buttons des Termeditors

Terme werden immer von ihren Blättern zur Wurzel hin ausgewertet. Das bedeutet, dass das Auswertungsergebnis eines Terms immer von seinen Blättern abhängt.

- Operatoren `IS` und `OneOf`: Mit dem Operator `IS` kann in einem Term überprüft werden, ob ein Konfigurationsparameter aktiv ist. Ist dies der Fall, wertet der Operator zu `true` aus. Um einen `IS`-Operator anzulegen, muss im Termeditor der Knoten ausgewählt sein,

der durch den Operator ersetzt werden soll. Daraufhin wird mit einem Klick auf den entsprechenden Button der Operator erzeugt. Im sich nun öffnenden Dialogfenster muss in der Combobox der Typ des Konfigurationsparameters ausgewählt werden. Daraufhin kann in der angezeigten Auswahl der gewünschte Konfigurationsparameter angeklickt und auf „OK“ geklickt werden. Der `IS-`Ausdruck wird daraufhin in der Baum- und in der Textdarstellung angezeigt.
Der Operator `One Of` kapselt mehrere durch `OR` verknüpfte `Is`-Ausdrücke. Er vereinfacht so die Erstellung komplexer Ausdrücke und sorgt für mehr Übersicht in der Darstellung. Die Erstellung erfolgt analog zum `Is`-Operator.

- Operator `Term`: Mit dem Operator `Term` kann ein bereits bestehender Term als Teil eines Terms verwendet werden. Um einen bereits bestehenden Term in den Termbaum einzufügen, muss der Knoten markiert werden, der durch den Term ersetzt werden soll und der entsprechende Button geklickt werden. Der Name des eingefügten Terms wird daraufhin in der Baum- und Textdarstellung des bearbeiteten Terms angezeigt.

- Operatoren `Not`, `And`, `Or`, und `Xor`: Die booleschen Operatoren haben die aus der booleschen Algebra bekannten Funktionen. Sie können auf die nun bereits bekannte Weise in den Term eingefügt werden. Mit diesen Operatoren ist es möglich, beliebig komplexe logische Verknüpfungen zwischen den Auswertungsergebnissen der andern Operatoren herzustellen.

- Konstanten `true` und `false`: Die booleschen Konstanten können zu Testzwecken in einen Term eingefügt werden. Sie werden auf die bekannte Art und Weise in den Term eingefügt.

*Editieren eines Terms in Textansicht*

Um einen Term in der Textansicht zu editieren, ist ein gewisses Maß an Erfahrung mit der Textdarstellung erforderlich. Daher wird empfohlen Terme zunächst nur in der Baumansicht zu editieren und sich mit den entsprechenden Ergebnissen in der Textansicht vertraut zu machen.

Terme können in der Textansicht mit der Tastatur editiert werden. Wurde ein Term editiert, kann er mit dem „Übernehmen“-Button in die Baumansicht übernommen werden. Eventuelle Syntaxfehler werden daraufhin angezeigt. Terme werden nur übernommen wenn sie fehlerfrei sind. Soll die Textansicht auf den Zustand vor dem Editieren zurückgesetzt werden, ist der Button „Rückgängig“ zu klicken.

*Textuelle Beschreibung eines Terms*

Für komplexe Terme kann eine kurze textuelle Beschreibung dem schnelleren Verständnis zuträglich sein. Das entsprechende Feld des Termeditors bietet dafür die Möglichkeit.

*Speichern und Löschen eines Terms*

Ein im Termeditor ausgewählter Term kann jederzeit über den Button „Speichern“ gespeichert oder über den Button „Löschen“ gelöscht werden. Beim Speichern wird die aktuelle Ansicht in die Datenbank übernommen. Wurde beim Verlassen des Editors eine Ansicht noch nicht gespeichert, so folgt ein Benutzerdialog in dessen Folge ausgewählt werden kann, ob die gemachten Änderungen übernommen oder verworfen werden sollen.

## 3.5 Erstellung und Verwaltung der Selektionsmechanismen

### 3.5.1 Fachkonzept

Ein Referenzmodell kann dabei aus mehreren Sprachen und Modellen bestehen. Sprachen werden durch Elementtypen definiert. Neben den Referenzmodellen werden Konfigurationsparameter definiert, die die Charakterisierung eines Referenzmodells und dessen Konstrukte (Elementtypen, Objektdefinitionen, Kontexte etc.) ermöglichen. Beispielsweise ist die Objektdefinition eines Kennzahlensystems für das Controlling der Lagerhaltung in Handelsunternehmen (z. B. mit der Kennzahl Lagerfüllgrad) nur dann relevant, sofern das Modell nutzende Unternehmen auch tatsächlich die Geschäftsart Lagergeschäft betreibt.

Die Erweiterung des H2-Toolsets um Konfigurationsmechanismen ermöglicht die Prüfung der Konstrukte gegen Konfigurationsparameter und somit ihre Existenzberechtigung. Anhand dieser Prüfung erfolgt die Selektion von Konstrukten, die zuvor den Konfigurationsparametern zugeordnet wurden. Diese Zuordnung der Konfigurationsparameter erfolgt über Terme. Je nach Granularitätsgrad des Konstruktes werden verschiedene Selektionsmechanismen unterschieden. Ihre Abhängigkeiten und Mechanismen zur Konsistenzsicherung werden sind:

- *Sprachselektion:* Wird eine Sprache ausgeblendet, so werden alle Modelle und dessen Elemente der Sprache mit ausgeblendet.
- *Modellselektion:* Die Modellselektion verhindert das Anzeigen des selektierten Modells und dessen Elemente. Diese Elemente sind die Objektdefinitionen und die Objektausprägungen des Modells. Die zugehörige Sprache und andere Modelle in der Sprache sind von der Modellselektion nicht betroffen.

- *Objekttypselektion:* Objekttypen sind Elemente einer Sprache. Die Selektion eines Objekttypen wirkt sich auf die Objektdefinitionen und Objektausprägungen, die über diesen Objekttypen definiert wurden, in allen Modellen der Sprache aus. Zudem wirkt sich dies auf die kontextabhängigen Objekttypen bzw. Objekttypen in einem Kontext aus. Zu beachten ist hierbei, dass nachfolgende Elemente (bei Kontextregeln oder Objektdefinitionen/ -ausprägungen) der Baumstruktur mit ausgeblendet werden, um die Konsistenz der Modelle sicherzustellen.
- *Kontextabhängige Objekttypselektion:* Entspricht der Selektion von Objekttypen mit der Einschränkung, dass nur die Objekttypen in einem Kontext ausgeblendet werden. Die Erläuterungen zur Objekttypselektion gelten ansonsten analog.
- *Objektdefinitionsselektion:* Die Selektion einer Objektdefinition wirkt sich auch auf dessen Ausprägungen aus. Alle im Baum untergeordneten Objekte, werden mit ausgeblendet.
- *Objektausprägungsselektion:* Die Selektion einer Objektausprägung führt zu dessen Ausblendung und wirkt sich auf kein anderes Konstrukt aus. Im Baum untergeordnete Objekte werden mit ausgeblendet.

Das ER-Diagramm in Abb. 39 zeigt die Zuordnung der Terme zu den Konstrukten des H2-Toolsets. Einem Referenzmodell können mehrere Terme zugeordnet werden. Jedem Konstrukt des Referenzmodells kann dabei maximal ein Term zugeordnet werden. Diese vermeintliche Restriktion wurde gewählt, da mehrere Terme aus Konsistenzgründen wiederum über einen Operator verknüpft werden müssen und dies auch wieder als Term aufgefasst werden kann.

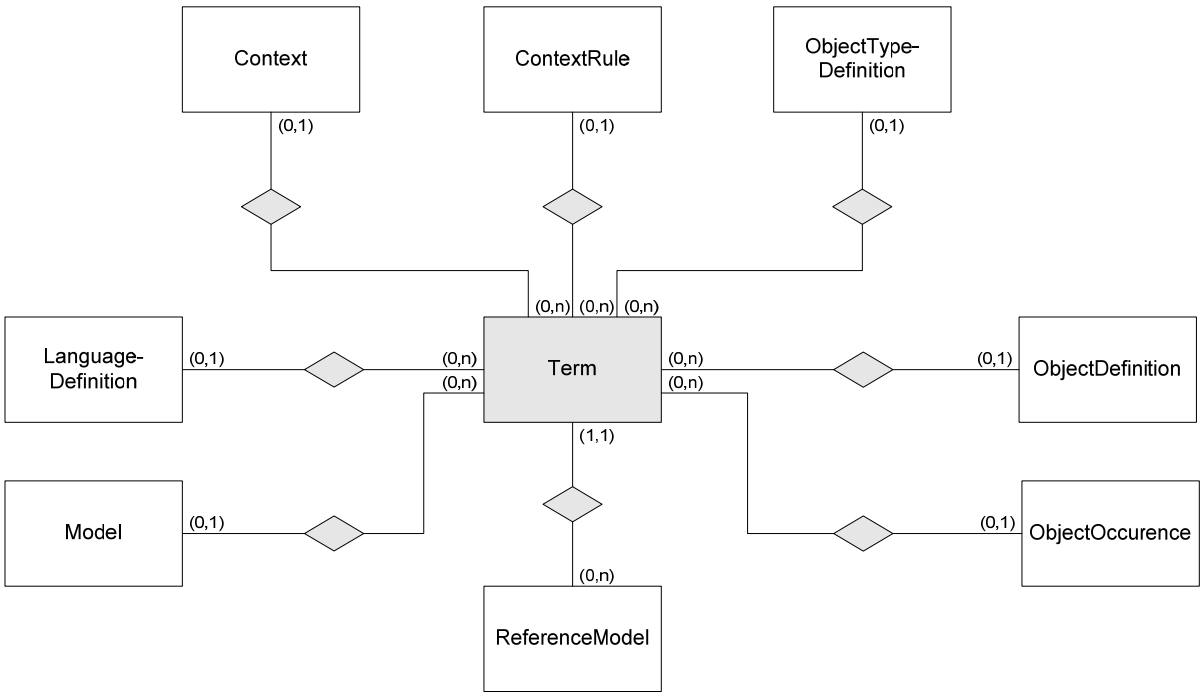



**Abb. 39:** ER-Diagramm der Konfigurationsmechanismen

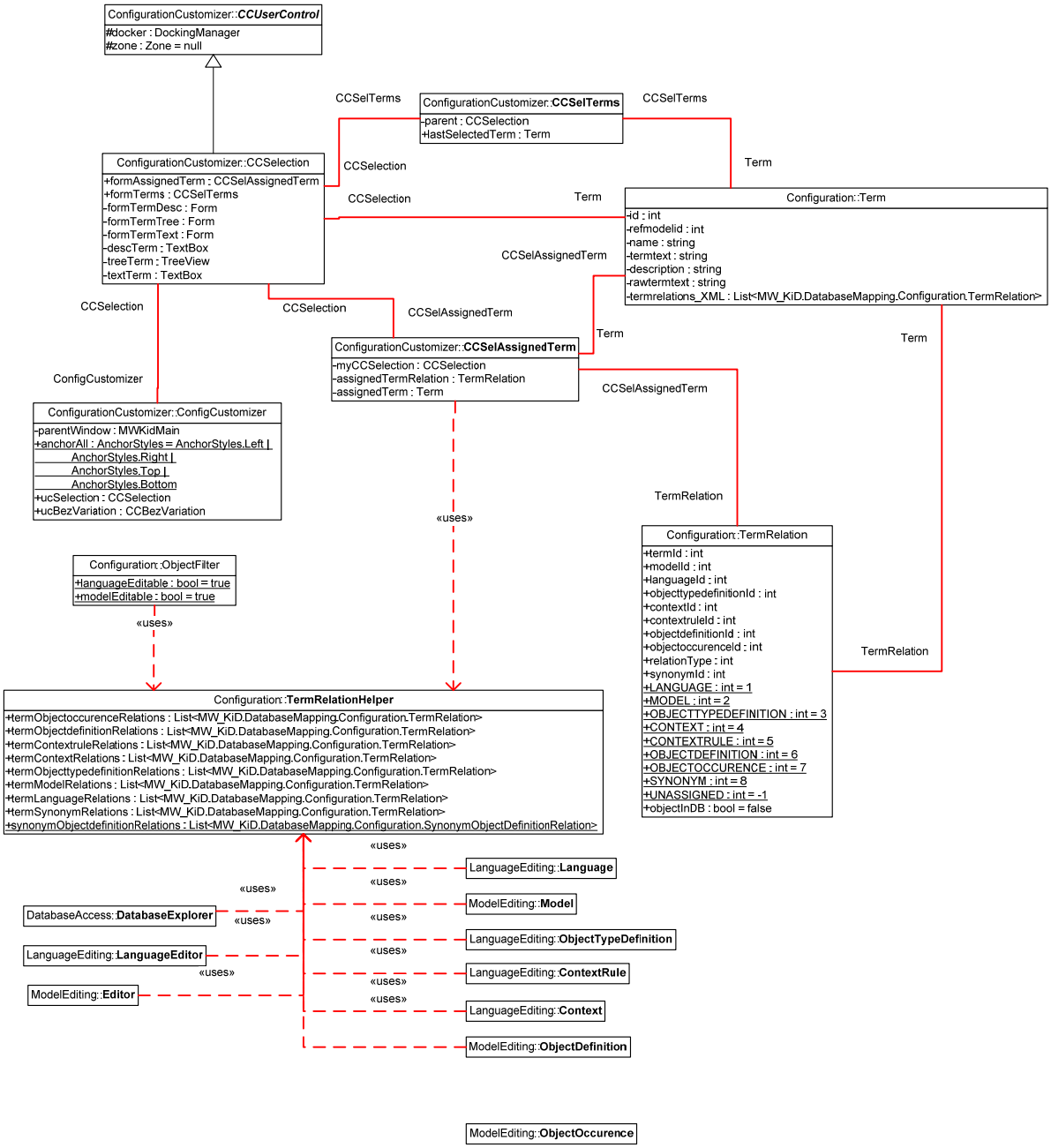



**Abb. 40:** Klassendiagramm der Konfigurationsmechanismen

### 3.5.2 DV-Konzept und Implementierung

Zu selektierende Konstrukte im H2-Toolset sind: *language*, *model*, *context*, *contextRule*, *objectTypeDefinition*, *objectDefinition* und *objectOccurence*. Durch die Zuordnung dieser zu einem Term kann beim Laden der Konstrukte bzw. der Adaption des Referenzmodells geprüft werden, ob der Term zum Ausblenden eines Konstruktes führt. Die Zuweisung erfolgt über die Primärschlüssel der jeweiligen Tabellen. Das erste Attribut stellt die *termId* dar, das zweite Attribut die Id des Konstruktes, welchem der Term zugewiesen wurde (z. B. *languageId*,

*modelId*, etc.). Eine Erweiterung der o. g. Tabellen um das Attribut *termId* wäre zwar möglich, jedoch wird durch das Anlegen der Relationentabellen (vgl. Tab. 24 bis Tab. 30) eine Trennung des Toolsets und dessen Erweiterungen erreicht und so Interdependenzen mit bestehenden Klassen verhindert.

*Sprachselektion*

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termId | int(11) | 1 | Id des Terms |
| languagelId | int(11) | 1 | Id der Sprache |

**Tab. 24:** Tabelle Termlanguagerelation

*Modellselektion*

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termId | int(11) | 1 | Id des Terms |
| modelId | int(11) | 1 | Id des Modells |

**Tab. 25:** Tabelle Termmodelrelation

*Objekttypselektion*

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termid | int(11) | 1 | Id des Terms |
| objecttypedefinitionid | int(11) | 1 | Id des Objekttypen |

**Tab. 26:** Tabelle Termobjecttypedefinition

*Kontextselektion*

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termid | int(11) | 1 | Id des Terms |
| contextid | int(11) | 1 | Id des Kontextes |

**Tab. 27:** Tabelle Termcontextrelation

*Kontextabhängige Objekttypselektion*

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termid | int(11) | 1 | Id des Terms |
| contextruleid | int(11) | 1 | Id der Kontextregel, für die kontextabhängige Objekttypselektion |

**Tab. 28:** Tabelle Termcontextrulerelation

*Objektdefinitionsselektion*

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termid | int(11) | 1 | Id des Terms |
| objectdefinitionid | int(11) | 1 | Id der Objektdefinition |

**Tab. 29:** Tabelle Termobjectdefinitionrelation

*Objektausprägungsselektion*

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termid | int(11) | 1 | Id des Terms |
| objectoccurenceid | int(11) | 1 | Id der Objektausprägung |

**Tab. 30:** Tabelle Termobjectoccurencerelation

Das Tabellenschema für die Tabellen *Refmodel* und *Term* wurden bereits in den Abschnitten 3.1.2 und 3.4.2 erläutert (vgl. Tab. 8 und Tab. 22).

*Beteiligte Pakete und Klassen*

Die Klasse *TermRelation* in dem Paket *DatabaseMapping::Configuration* dient dem Mapping von Datenbankeinträgen und C#-Objekten. Aufgrund der gleichartigen Struktur wurden alle Relationen-Tabellen zwischen Termen und den genannten H2-Elementen durch eine einzelne Klasse abgebildet. Die Unterscheidung der Konstrukte erfolgt über das Attribut *relationType*, welches über entsprechende Konstanten gesetzt werden kann. Im Wesentlichen beinhaltet diese Klasse Methoden zum Lesen, Löschen und Schreiben von Datensätzen aus der Datenbank.

Die Klasse *TermRelationHelper* beinhaltet Methoden, die die Verwaltung der TermRelations unterstützt. Über den Konstruktor werden initial die Daten aus allen Term-Relationentabellen geladen. Die Klasse wird instanziiert, nachdem das H2-Toolset eine Verbindung zur Datenbank hergestellt hat. Um das Auffinden von Termen zu einem Konstrukt zu erleichtern, dienen die Methoden *getTerm*[H2-Konstrukt]*Relation*. Diese werden mit Hilfe eines Prädikates in der generischen Liste gesucht. Zum Anlegen neuer Konfigurationsansatzpunkte dienen die Methoden *manageNewTerm*[H2-Konstrukt]*Relation.* Diese sorgen dafür, dass eine neue Relation in die Datenbank geschrieben wird und die generischen Listen sowie die grafische Benutzeroberfläche des Configuration Customizers aktualisiert werden.

Die Klasse *ObjectFilter* beinhaltet Methoden zum Ausblenden von Konstrukten. Das Ausblenden erfolgt anhand von Termen, Konfigurationsparametern und Benutzerrechten. Die Klasse beinhaltet folgende Methodensätze.

Die Methoden *checkTermOn*[H2-Konstrukt] prüfen ausgehend von einem Konstrukt, ob und wie ein zugewiesener Term ausgewertet wird. Dabei wird erst ermittelt, ob ein Term in Relation zu dem Konstrukt steht. Ist dies nicht der Fall, führt dies grundsätzlich zur Anzeige des Konstruktes. Andernfalls wird über die Methode *EvaluateTerm* der Term anhand der aktuell gesetzten Umgebungsparameter überprüft.

Da die Konstrukte auf Metaebene und Modellebene zusammenhängen, kann mit den Methoden *checkTermOn*[H2-Konstrukt] keine vollständige Auswertung stattfinden. Die Vollständige Auswertung erfolgt in den Methoden *CheckAndPerfomVisibility(*[H2-Konstrukt]*)* und greift dabei auf die Methoden *checkTermOn*[H2-Konstrukt] zurück, um die Konstrukte auf sich selbst und ihre Abhängigkeiten zu prüfen. Abb. 41 verdeutlicht diese Abhängigkeiten.

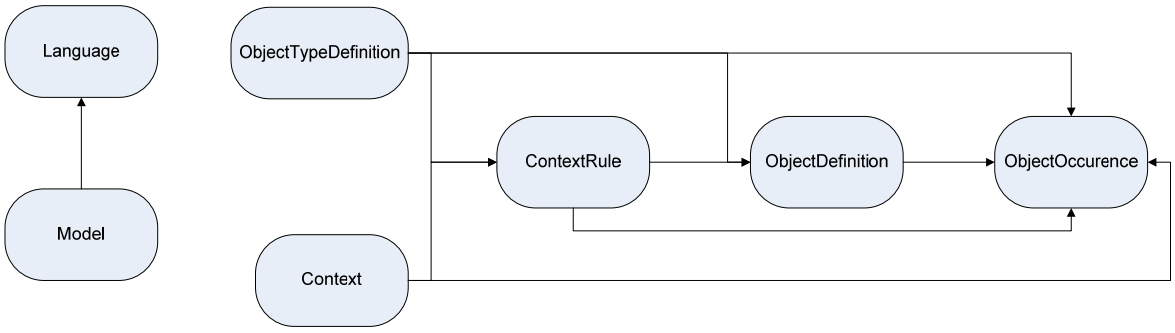



**Abb. 41:** Abhängigkeiten der zu selektierenden Konstrukte

Um die Abhängigkeiten bei der Ausblendung zu verdeutlichen folgt für jedes Konstrukt des H2-Toolsets eine Erläuterung für das Vorgehen. Grundsätzlich wurde der Ansatz verfolgt, die Konstrukte beim Laden aus der Datenbank der Sichtbarkeitsüberprüfung zu unterziehen, da zum einen die grafische Benutzeroberfläche nur auf Objekte des Programms zugreifen und hiermit die Anzahl der zu ändernden Stellen reduziert wurden und zum anderen die Caches des Programms nicht unnötig mit Daten gefüllt werden, um die Performanz und Konsistenz des Modells zu gewährleisten. Die einzige Abweichung von diesem Konzept erfolgt bei den Objektausprägungen: Diese werden nur bei Bedarf für die Anzeige der Modelle aus der Datenbank geladen. Somit ist vorher nicht abzusehen, welche Objekte benötigt werden. Der Cache der Objektausprägungen füllt sich je nach Bedarf für die Anzeige.

*Beteiligte Konstrukte*

Das Laden der Sprachen aus der Datenbank erfolgt über die Klasse *Language*. Diese stellt die Methode *LoadLanguageByRefmodel* bereit und füllt den statischen Language-Cache. Beim Laden der Sprachen wird für jede Sprache die Sichtbarkeit überprüft. Dadurch, dass das Laden der Modelle (nicht Model) und aller weiteren Konstrukte abhängig von dem Language-Cache ist, genügt es zu überprüfen, ob die Sprache angezeigt werden soll oder nicht.

Das Laden und Ausblenden der Modelle erfolgt analog zu dem von *Language*. Das Laden von Modellen erfolgt abhängig von dem Language-Cache. Die Methoden zum Laden der Modelle aus der Datenbank sind *GetByLanguage(Language l)* und *GetByID(int id).* Hier wird für jedes Modell geprüft, ob es angezeigt werden soll. Die Methode *CheckAndPerformVisibility(Model model, ref string error)* prüft zusätzlich, ob die Sprache geladen wurde bzw. ob die Sprache angezeigt wird. Dadurch wird absolut sichergestellt, dass keine Modelle ohne Sprachen geladen werden.

Das Selektieren der Objekttypdefinitionen erfolgt nicht direkt über das Laden der Konstrukte aus der Datenbank, da die Abhängigkeiten zwischen den Konstrukten sehr komplex werden können. Daher wäre die Konsistenz der Modelle nicht zwangsläufig sichergestellt und die Programmstabilität negativ beeinflusst. Nach dem Laden der Language werden alle der Sprache zugeordneten Konstrukte geladen. Dies sind insbesondere die Kontexte, Kontextregeln und die Objekttypdefinitionen. Das Selektieren der Objekttypdefinitionen erfolgt über die Methode *clearForSelection()* in der Klasse *Language*. Diese ruft die Methode *removeOTDs(ObjectTypeDefinitions otds)* auf, die die komplette Liste an Objekttypdefinitionen bereinigt. Das Selektieren erfolgt in der Methode schließlich wieder über die *CheckAndPerformVisibility*-Methode.

Das Selektieren von Kontexten erfolgt analog zu den Objekttypdefinitionen. Über die Methode *clearForSelection()* aus der Klasse *Language* wird die Methode *removeCs(Contexts cs)* aufgerufen, die wiederum die Methode *CheckAndPerfomrVisibility(Context c)* zum Prüfen der Selektion verwendet.

Das Selektieren der Kontextregeln erfolgt analog zu denen der Kontexte und Objekttypdefinitionen. Über die Methode *clearForSelektion()* in der Klasse *Language* wird die Methode *removeCRs(ContextRules crs)* aufgerufen, die wiederum über *CheckAndPerformVisibility(ContextRule contextRule, ref string error)* die Sichtbarkeit überprüft. Da eine Kontextregel nur mit Vater-Objekttypdefinition, Kind-Objekttypdefinition und Kontext existieren kann, wird in der o. g. Methode die Sichtbarkeit der Attribute der Contextrule überprüft (Vgl. Abb. 41). Einige Ausnahmen erfordern den Aufruf der *checkAndPerformVisibility*-Methode. In der Klasse *Modell* sind dies das Anzeigen der Kontexte im Modell-Editor und in der Klasse *EdgeTypeContextRuleRelation* die Kantentypen.

Objektdefinitionen stellen die Grundlage für Objektausprägungen dar. Zu jeder Objektausprägung in einem Modell existiert eine entsprechende Objektdefinition. Die Objektdefinitionen werden nicht explizit aus dem Cache entfernt, da jede ObjectOccurence sein ObjectDefinition-Objekt kennt. Da aber die ObjectOccurences von Anfang an nicht bekannt sind, führt das Eliminieren der ObjectDefinitions zu einem instabilen Programm und Seiteneffekten.

Die ObjectOccurences werden dem Benutzer in einem Modell dargestellt. Das Laden der ObjectOccurences aus der Datenbank erfolgt in dem Modell-Editor durch Anwählen des „expand“ des EditorTreeViews (Baum). Die Selektion erfolgt beim Laden der ObjectOccurences aus der Datenbank. Da für das Laden mehrere Methoden in der Klasse ObjectOccurence verantwortlich sind, ist das Selektieren in der Methode *FilterVisibleOccurences(ArrayList objectOccurences, IDbTransaction transaction)* zusammengefasst, die über die übergebenen *ArrayList objectOccurences* die zu selektierenden Konstrukte herausfiltert. Die *CheckAndPerformVisibility*-Methode ist hier am umfangreichsten, da sich die ObjectOccurences auf unterster Selektionsebene befinden, d. h. abhängig von der Selektion der Übergeordneten Konstrukte wie ObjectDefinitions, ObjectTypeDefinitions, ContextRule und Context sind.

Der Configuration Customizer ist das zentrale Werkzeug um Konfigurationsansatzpunkte zu setzen. Diese Ansatzpunkte dienen entweder der Selektion von Konstrukten oder der Bezeichnungs- und Darstellungsvariation. Wie die Konfigurationsansatzpunkte gesetzt werden, wird im Folgenden erläutert.

Um den Configuration Customizer starten zu können, muss sich der Benutzer angemeldet haben und die entsprechenden Rechte (Funktionsrecht *FunctionRefmodelling*) zum Öffnen des Configuration Customizers besitzen. Zudem ist es notwendig, ein Referenzmodell zu aktivieren. Ist dies erfolgt, kann das Tool über die Menüleiste des H2-Toolsets unter „Ansicht → Configuration Customizer einblenden“ gestartet werden.

***Selektion mit dem Configuration Customizer***

In dem Reiter „Selektion“ werden Ansatzpunkten angezeigt sowie die Möglichkeit bereitgestellt, neue Ansatzpunkte zu setzen. Die Informationen zu den Ansatzpunkten werden dargestellt, sobald das relevante Konstrukt aus dem H2-Toolset ausgewählt wird. Neue Ansatzpunkte können durch eine benutzerfreundliche Drag & Drop Funktionalität gesetzt werden: Diese Funktionalität wird im Folgenden anhand der Selektionsmechanismen genau erklärt.

*Zuweisen von Termen*

Der Reiter Selektion gliedert sich in zwei Bereiche. Zum einen in den Bereich „Zugewiesener Term“ und zum anderen in den Bereich „Terme“ (vgl. Abb. 42).

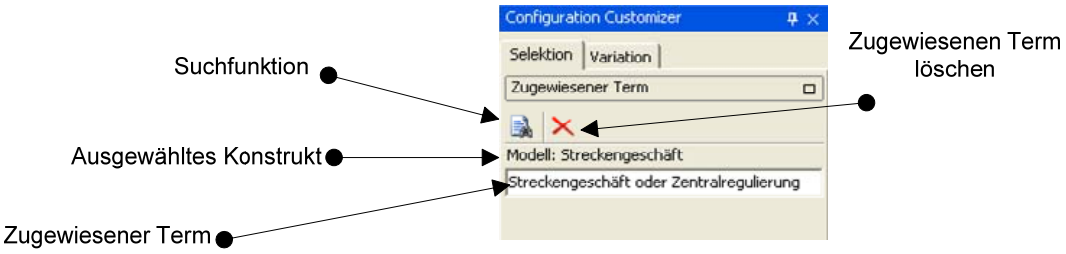



**Abb. 42:** Zuweisen von Termen

Ist ein Modell oder eine Sprache aus dem Database-Explorer, ein Objekttyp, Kontext oder ein Objekttyp in einem Kontext im Sprach-Editor, oder ein Element aus dem Modell-Editor ausgewählt, wird angezeigt, ob und welcher Term hinterlegt ist. Zudem besteht bei einem zugewiesenen Term die Möglichkeit, die Referenz zu dem Konstrukt zu löschen oder sich alle weiteren Verwendungen des Terms anzeigen zu lassen.

*Terme im Configuration Customizer*

Der Bereich „Terme“ enthält ein Listenfeld, in dem alle Terme mit Namen aufgelistet sind, die dem aktivierten Referenzmodell zugeordnet sind. Über das Symbol „leeres Blatt“ kann der Termeditor geöffnet werden, um neue Terme anzulegen. Über das Symbol „Lupe“ können Terme gesucht werden (vgl. Abb. 43).

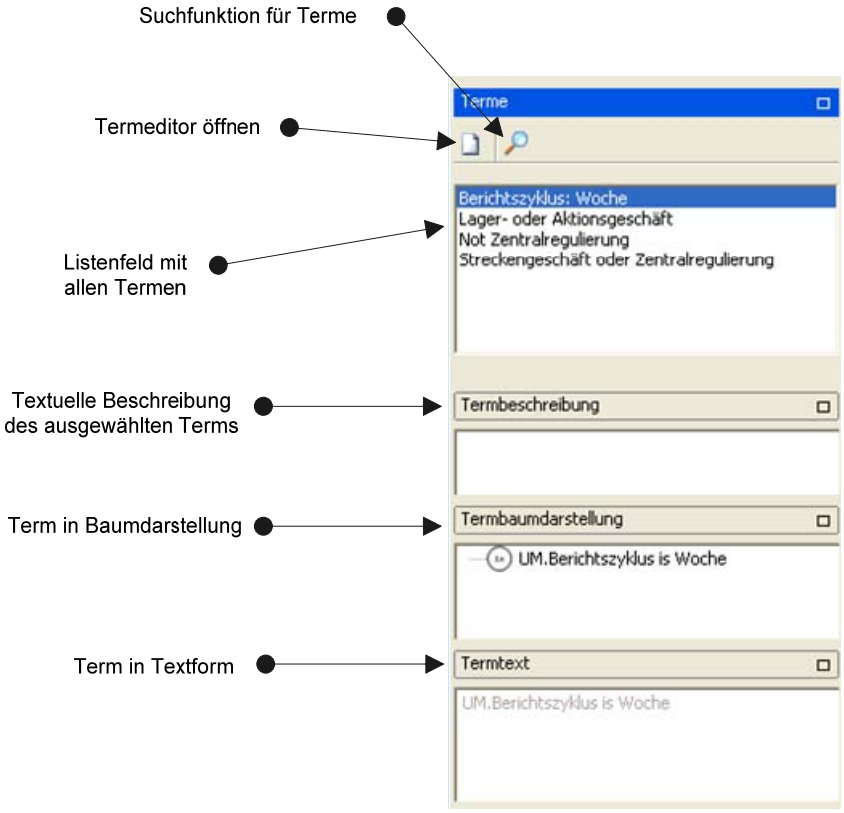



**Abb. 43:** Terme im Configuration Customizer

Wird ein Term aus der Liste angewählt, erscheinen in dem darunter liegenden Feldern zusätzliche Informationen zu dem Term. In dem Feld „Termbeschreibung“ erscheint eine textuelle Erläuterung zu dem Term. In dem Feld „Termbaumdarstellung“ wird der Term in Baumform angezeigt. Zudem wird in dem Feld „Termtext“ der Term in einer Klammernotation angezeigt. Das Editieren des Terms in den Feldern ist nicht möglich.

*Suche von Termen*


**Abb. 44:** Termauswahl

Nach Auswahl des Symbols „Lupe“ öffnet sich das Fenster zum Suchen von Termen. Bei Eingabe des Suchtextes erfolgt eine automatische Suche in der Termliste. Entspricht der Suchtext dem Termnamen, wird der Baum des Terms automatisch angezeigt.

*Sprachselektion, Modellselektion*

Um eine Sprachselektion oder eine Modellselektion durchführen zu können, ist es notwendig, den Datenbank-Explorer und den Configuration Customizer geöffnet zu haben. Im Datenbank-Explorer sind alle im Referenzmodell vorhandenen Sprachen und Modelle in Baumform dargestellt. Um einen Ansatzpunkt zu setzen, ist per Drag & Drop ein Term aus dem Listenfeld mit allen Termen auf die Sprache für eine Sprachselektion oder auf ein Modell für eine Modellselektion zu ziehen. In der oben dargestellten Abbildung ist dies beispielhaft für die Sprache „MW-KID“ und das Modell „Streckengeschäft“ visualisiert. Nachdem der Term auf

das Element gezogen wurde, erscheinen im Bereich „Zugewiesener Term“ Informationen zu dem Konstrukt, auf das der Term gezogen wurde (vgl. Abb. 45).

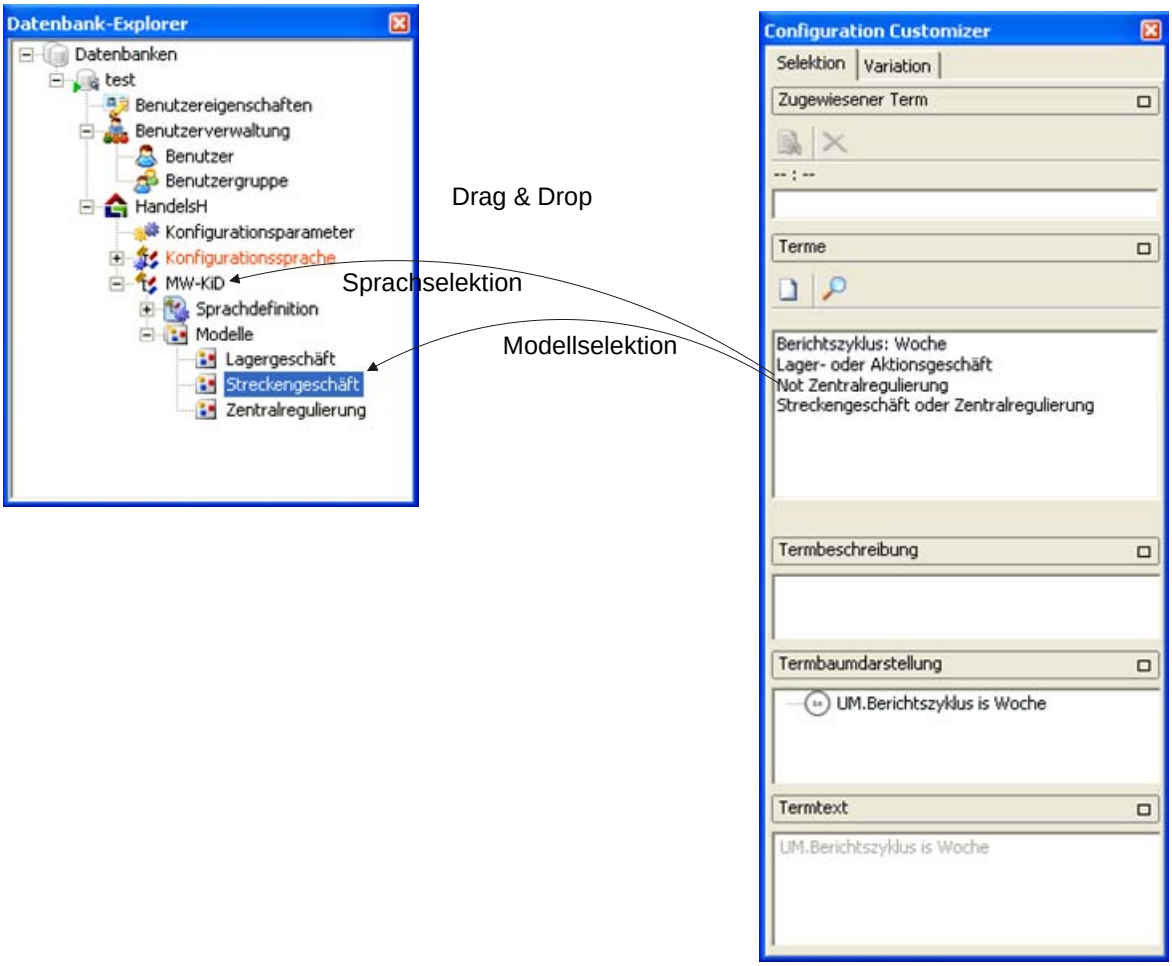



**Abb. 45:** Sprachselektion und Modellselektion

Meldet sich nun ein anderer Benutzer am System an oder wird eine Adaption durchgeführt und erfüllt nicht die Kriterien, damit der Term zu `true` auswertet, ist die Sprache bzw. das Modell nicht für ihn sichtbar. Nicht angezeigte Sprachen führen dazu, dass die abhängigen Modelle ebenfalls nicht angezeigt werden.

*Kontextselektion, Objekttypselektion, kontextabhängige Objekttypselektion*

Um Konfigurationsansatzpunkte auf der Meta-Ebene eines Modells zu setzen, ist es notwendig neben dem Configuration Customizer den Spracheditor zu öffnen. Ein Ansatzpunkt wird gesetzt, indem ein Term aus der „Liste aller Terme“ auf das gewünschte Element gezogen wird. Zur Verfügung stehen hierbei die Objekttypen aus dem Listenfeld „Objekttypen“, Kon-

texte und Kontextabhängige Objekttypen aus dem Baumfeld „Kontexte“. Wird ein Term auf einen Objekttypen gezogen, schlagen sich die Termauswertung auf alle darunter liegenden Elemente durch. Wertet ein Term eines Objekttypen zu `false` aus, werden alle Elemente in dem Modell-Editor, die von diesem Objekttypen sind, nicht angezeigt. Das gleiche gilt für Kontexte und Objekttypen in Kontexten (vgl. Abb. 46).

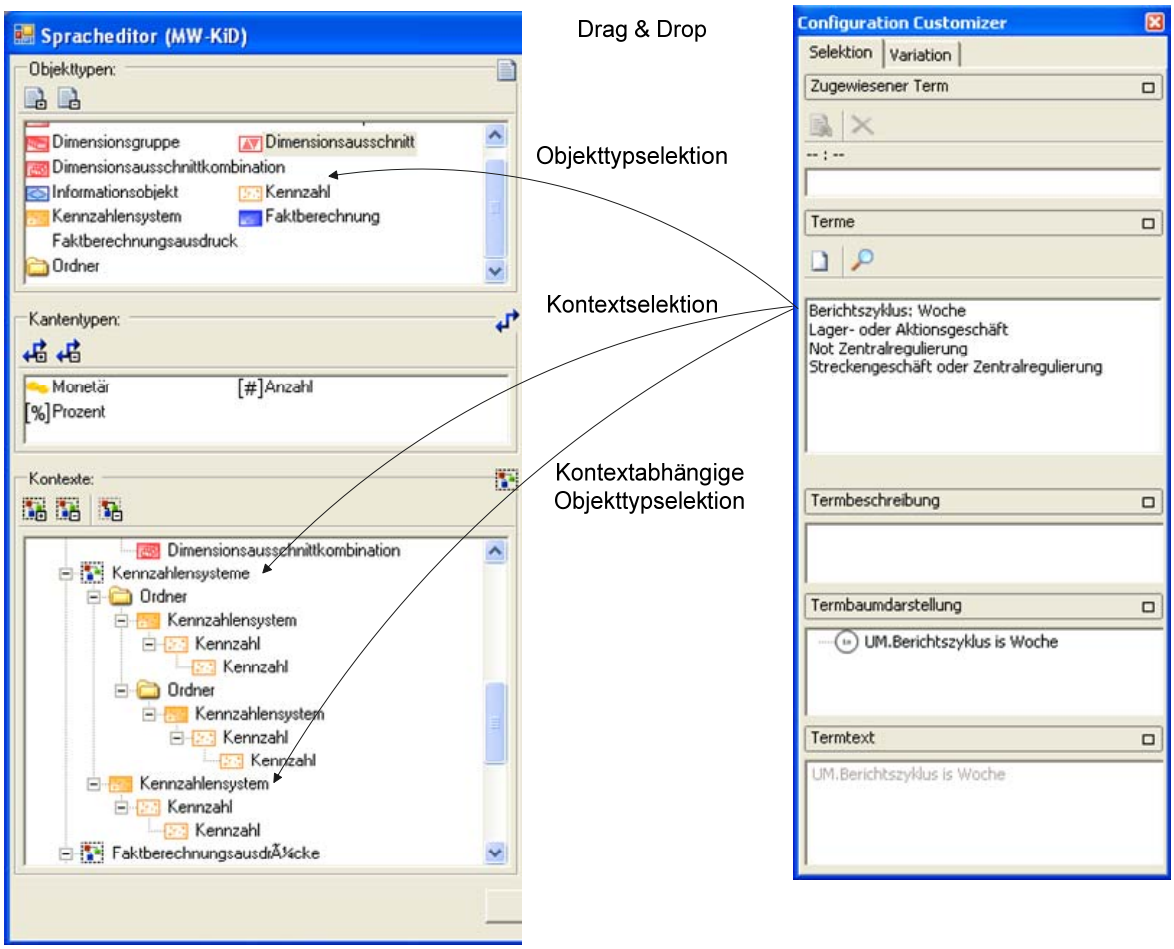



**Abb. 46:** Kontextselektion, Objekttypselektion und kontextabhängige Objekttypselektion

Im Spracheditor selbst werden keine Konstrukte ausgeblendet. Wertet jedoch ein Term in diesem Fenster zu `false` aus, führt das dazu, dass die Sprache von diesem Benutzer nicht editiert werden darf. Hierdurch soll verhindert werden, das Parallel- oder Phantomversionen entstehen.

Bei der Modelladaption ist eine weitere Besonderheit zu beachten. Werten Terme auf Kontexten oder kontextabhängigen Objekttypen zu `false` aus, werden diese Konstrukte trotzdem in das adaptierte Modell übernommen. Dies ist notwendig, da sonst inkonsistente Modelle entstehen würden.

*Elementselektion*

Um eine Elementselektion durchführen, muss der Modell-Editor mit dem entsprechenden Kontext geöffnet sein. Hier kann auch wieder ein Term aus der „Liste aller verfügbaren Terme“ ausgewählt werden und auf das Element gezogen werden. Die Elemente, die hier unterschieden werden können, sind Elementdefinitionen und Elementausprägungen. Wird der Term auf eine Elementdefinition gezogen, wirkt sich die Auswertung des Terms auf all seine Ausprägungen im Modell aus (vgl. Abb. 47).

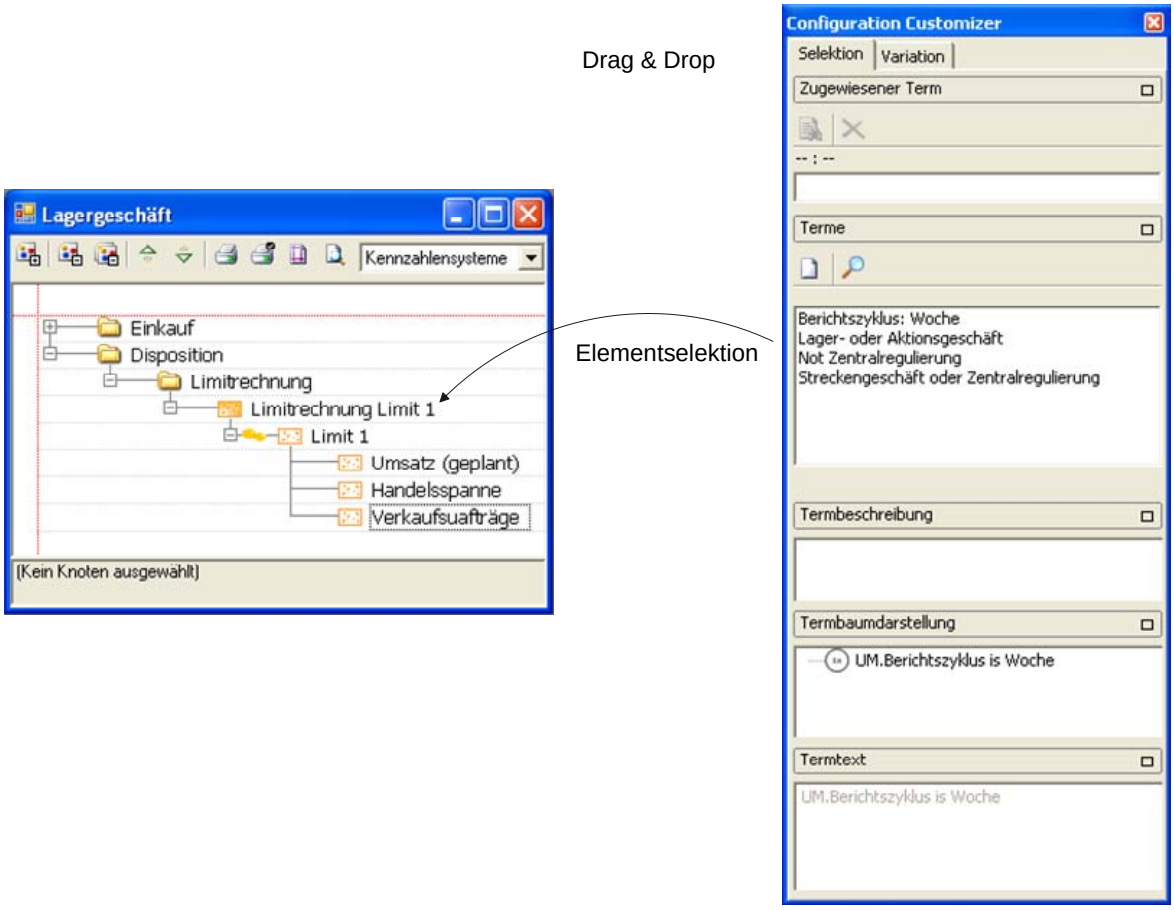



**Abb. 47:** Elementselektion

Wertet ein Term eines Elementes zu `false` aus, werden alle darunter liegenden Elemente zusätzlich ausgeblendet. Hierdurch wird erreicht, dass die Modelle stets konsistent sind.

## 3.6 Erstellung und Verwaltung der Bezeichnungsvariation

### 3.6.1 Fachkonzept

Um je nach Anwendungskontext eines Referenzmodells sich ändernden Namenskonventionen Rechnung zu tragen, sind im H2-Toolset konfigurationsparameterinduziert Bezeichnungen von Modellelementen auszutauschen. Damit die Konsistenz und Einheitlichkeit der Modellierungsergebnisse sichergestellt ist, erfolgt die Bezeichnungsvariation auf Basis eines im Rahmen der Referenzmodellkonstruktion systematisch aufgebauten und im H2-Toolset fest verankerten Basiswortkataloges. Wenngleich Art und Umfang der Bezeichnungsvariation vom Referenzmodellersteller individuell auszugestalten sind, geben die Rahmenbedingungen vor, dass referenzmodellspezifisch Begriffsmengen gleicher Bedeutung (Synonyme) zu Synonymgruppen zusammengefasst und über eine Umbenennungstabelle den Konfigurationsparameterausprägungen zugeordnet werden. Da Modellelementbenennungen mitunter nicht nur aus einzelnen Wörtern, sondern auch aus Textfragmenten bestehen können, ist der Austausch der Bezeichnungen so umgesetzt, dass bei Bedarf nur Teile eines Modellelement-Strings entsprechend kenntlich gemacht und ausgetauscht werden.

Um die Wartbarkeit der Bezeichnungsvariation zu fördern, werden Synonyme nicht direkt den Objektdefinitionen zugeordnet. Stattdessen findet ein modellübergreifendes Umbenennungskonzept Verwendung, das den Bezeichnungsvarianten (Synonymen) der führenden Standardbezeichnungen (Synonymgruppen) Konfigurationsparameterausprägungen mit Hilfe von Termen zuordnet. Im H2-Modell-Editor wird daher mit den Standardbezeichnungen modelliert, die je nach Konfigurationsparameterausprägung, in der ein Modell später aufgerufen wird, aufgrund annotierter Terme automatisch ausgetauscht werden. Dieser Zusammenhang äußert sich im zugehörigen ER-Modell in Abb. 48 in der Einführung des Entititypen *SynonymGroup*, der sämtliche Standardbezeichnungen repräsentiert und diesen ObjektDefinitions zuweist. Hierbei kann einer ObjektDefinition maximal eine Synonymgruppe zugeordnet werden. Da Synonymgruppen immer im Kontext eines Referenzmodells angelegt werden, gehen Entities vom Typ *SynonymGroup* zusätzlich eine Relation mit *ReferenceModel* ein, so dass für jedes Referenzmodell ein eigener Basiswortkatalog gepflegt werden kann. Die eigentliche Konfiguration der Bezeichnungsvariation erfolgt nach demselben Prinzip wie bei den Selektionsmechanismen (vgl. Abschnitt 3.5). Den zu Synonymgruppen zusammengefassten Synonymen *Synonyms* werden Terme *Terms* zugeordnet. Hierbei darf jedem Synonym analog zu den Selektionsmechanismen maximal ein Term zugewiesen werden.

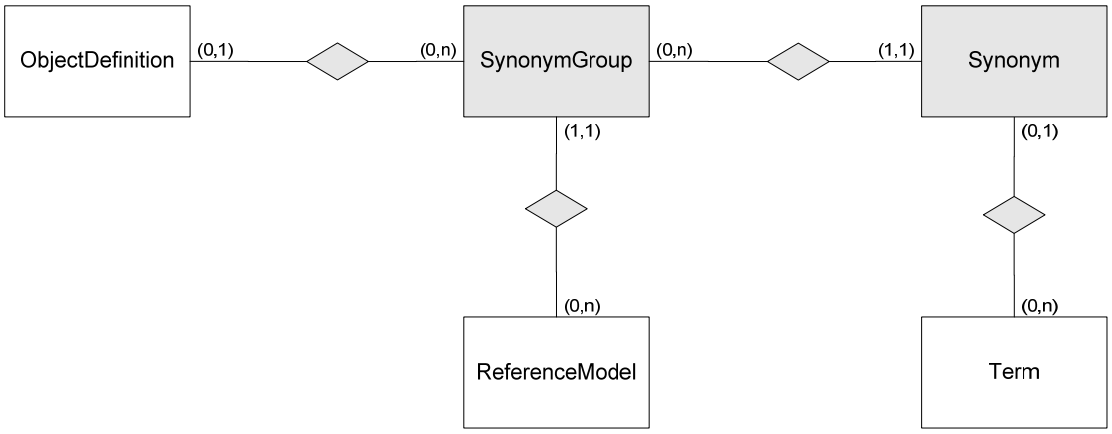



**Abb. 48:** ER-Diagramm der Bezeichnungsvariation

Damit die Bezeichnungsvariation für den Benutzer sichtbar wird, sind für den Fall, dass einer Objektdefinition eine Synonymgruppe zugeordnet ist, von der Anzeigekomponente des Modell-Editors entsprechend die Terme der zugehörigen Synonyme auszuwerten. Bei der Darstellung der Modellelemente wird daraufhin der in der Objektbezeichnung markierte Teilstring entsprechend ersetzt. Falls mehrere Terme zu `true` auswerten, hat der Referenzmodellersteller inkonsistent modelliert und eine entsprechende Fehlermeldung wird ausgegeben.

Für den Modellierungsprozess ergeben sich damit zwei Zuordnungsvorgänge: Einerseits müssen Modellelementen Synonymgruppen, andererseits den Synonymen (einer Synonymgruppe) Terme zugewiesen werden. Die Konfiguration betreffend liegt es nahe, den Editor für den Basiswortkatalog in die GUI-Komponente des Configuration Customizers einzubinden. Die beiden Zuordnungsprobleme lassen sich so per Drag & Drop lösen, indem Synonymgruppen vom Configuration Customizer in den Modell-Editor auf Modellelemente bzw. innerhalb des Configuration Customizers Terme auf Synonymgruppen gezogen werden.

### 3.6.2 DV-Konzept und Implementierung

Der die Bezeichnungsvariation betreffende Ausschnitt des Datenbankschemas umfasst deutlich weniger Tabellen als im zugehörigen ER-Diagramm enthaltene Entity- und Relationshiptypen. Weil Synonymgruppen immer nur genau einem Referenzmodell angehören dürfen, geht der Schlüssel des Referenzmodells direkt in die Tabelle der Synonyme ein und eine mögliche Relation *SG-RM.* entfällt. Aus Performanzgründen sind zudem Synonymgruppen und Synonyme in einer gemeinsamen Tabelle zusammengefasst, wobei die *parentId* sicherstellt, ob es sich bei einem Datensatz um eine Synonymgruppe oder ein Synonym handelt.

*Synonym*

Die den Basiswortkatalog umfassenden Synonyme werden in einer einstufigen Hierarchie gespeichert, wobei Wurzelknoten Synonymgruppen repräsentieren, deren Kindknoten Synonyme sind. Dies äußert sich darin, dass die *parentId* für Synonyme den Vaterknoten, d. h. die zugehörige Synonymgruppe, enthält bzw. im Falle einer Synonymgruppe mit dem Wert „-1“ belegt ist (vgl. Tab. 31).

| Feld | Typ | Beispiel | Erläuterung |
|---|---|---|---|
| Id | int(11) | 1 | ID des Eintrages |
| parentId | Int(11) | -1 | ID des Vaterknotens in der einstufigen Synonymhierarchie (handelt es sich um einen Wurzelknoten, d. h. eine Synonymgruppe, so ist die parentid -1) |
| refmodelId | int(11) | 1 | ID des zugehörigen Referenzmodells |
| synonymName | varchar(50) | Rechnung | String, der die Synonymbezeichnung enthält, die für den Austausch von Modellelementbenennungen vorgesehen ist |

**Tab. 31:** Tabelle Synonym

*Synonymobjectdefinitionrelation*

Die Kombination aus den Primärschlüsseln eines Synonyms und einer Objektdefinition stellt die Zuordnung von alternativen Bezeichnungen zu Modellelementen sicher. Entscheidend ist hierbei, dass die Geschäftslogik gewährleistet, dass ausschließlich Synonymgruppen (Standardbezeichnungen) zugeordnet werden, d. h. Synonyme bei denen die *parentId* mit dem Wert „-1“ belegt ist (vgl. Tab. 32).

| Feld | Typ | Beispiel | Erläuterung |
|---|---|---|---|
| synonymId | int(11) | 1 | ID des Synonyms (nur Synonymgruppen). |
| objectDefinitionId | Int(11) | 1 | ID der zugewiesenen Objektdefinition |

**Tab. 32:** Tabelle SynonymObjectDefinitionRelation

*Termsynonymrelation*

Analog zur vorigen realisiert die Tabelle *TermSynonymRelation* auf ähnliche Weise die Zuordnung von Termen zu Bezeichnungsvariationen. Da ausschließlich die Zuordnung von Termen zu Synonymen und nicht Synonymgruppen zugelassen ist, ist bei dieser Relation von

der Geschäftslogik der konträre Fall einzuschränken: Nur Kindknoten im Basiswortkatalog (`parentid != -1`) dürfen in die Relation aufgenommen werden (vgl. Tab. 33).

| Feld | Typ | Beispiel | Erläuterung |
|---|---|---|---|
| synonymId | int(11) | 1 | ID des Synonyms (keine Synonymgruppen) |
| termId | Int(11) | 1 | ID des zugeordneten Terms |

**Tab. 33:** Tabelle Termsynonymrelation

Die Bezeichnungsvariation wird durch die Klasse *BezVarView* sichergestellt. Sie enthält neben der Standardinitialisierung der Komponenten die Initialisierungsmethode für die verwendete *treeView*, zudem Hinzufüge- Lösch- und Updatefunktionen für die Synonyme bzw. die Synonymgruppen und die obligatorischen Eventhandler.

Bei der Auswertung der Terme für den jeweiligen Benutzer werden die hinterlegten Terme gegen die Konfigurationsparameterausprägungen ausgewertet. Dies geschieht in der Klasse *ObjectFilter* in der Methode *removeOO's*, welcher eine zusätzliche Behandlung von Elementen hinzugefügt wurde, damit Elemente, deren Synonymterme zu `true` auswerten, eine neue Bezeichnung erhalten.

Der Name des Objekts wird durch die Bezeichnung des Synonyms ersetzt, wenn der hinterlegte Term zu `true` auswertet. Ist ein Administrator angemeldet, wird diese Auswertung umgangen; für ihn sind immer die Originalbezeichnungen mit den Tags sichtbar, um die Übersichtlichkeit des Modells zu erhöhen (vgl. Abb. 49).

```
if (!CheckIfAdmin(General.Session.CurrentUser))
{
    // termid ist die Termid des Synonyms dessen term zu true auswertet
    termid = (ObjectFilter.checkSynonymOnObjectDefinition(oo.ObjectDefinition.ID));
    if (termid == -1)
    {
        oo.ObjectDefinition.Name = Tagdelete(oo.ObjectDefinition.Name);
        selection.Add(oo);
    }
    else if (termid == -2)
        MessageBox.Show("Es wurde falsch modelliert, da mehrere Synonyme zur Anwendung kämen.\nBitte
    else
    {
        /*
        // Zur Termid wird nun das Synonym gesucht. Danach muss
        // der Teilstring der Object occurence der zwischen den Tags
        // steht durch den Synonymnamen ausgetauscht werden! WGIDD?
         */
        string termtext = Term.GetTermNameById(termid);
        syn = synodrel.GetSynonymOnObjectDefinitionbytermid(oo.ObjectDefinition.ID, termid);
        string synonymbezeichnung = syn.SynonymName;

        oo.ObjectDefinition.Name = BVstringreplace(oo.ObjectDefinition.Name, synonymbezeichnung);
        selection.Add(oo);
    }
}
else selection.Add(oo);
```

**Abb. 49:** Algorithmus des Variationsmechanismus

Wertet keiner der Synonymterme zu `true` aus, wird die Standardbezeichnung weiterverwendet. Die Auswertung der Terme erfolgt analog zu den anderen Selektionsmechanismen.

*Bezeichnungsvariation mit dem Configuration Customizer*

Um Bezeichnungen von Modellelementen auszutauschen, muss für jedes Referenzmodell ein eigener Basiswortkatalog gepflegt werden. Hierzu existiert im „Configuration Customizer“ der Reiter „Bezeichnungsvariation“. Mit Hilfe zweier Buttons („Synonym hinzufügen“ bzw. „Synonym entfernen“) lassen sich Synonymgruppen sowie Synonyme anlegen und löschen, wobei die Hierarchiestufe im Basiswortkatalog den Ausschlag gibt, ob eine Synonymgruppe oder ein Synonym angelegt bzw. gelöscht wird (vgl. Abb. 50).

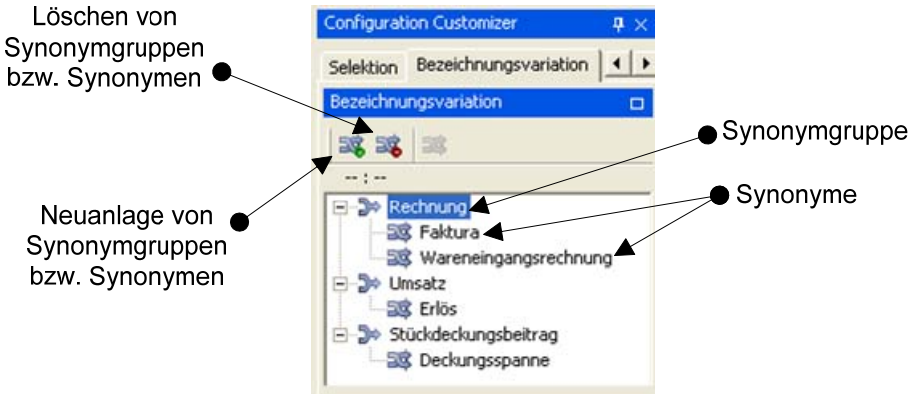



**Abb. 50:** Basiswortkatalog der Bezeichnungsvariation

Die Zuordnung der Terme zu Synonymen ist Abb. 51 zu entnehmen: Per Drag & Drop werden Terme aus der Termübersicht in den Basiswortkatalog gezogen. Für das markierte Synonym weist ein Textfeld nach erfolgreicher Zuordnung unter dem Katalog anschließend den zugeordneten Term aus.

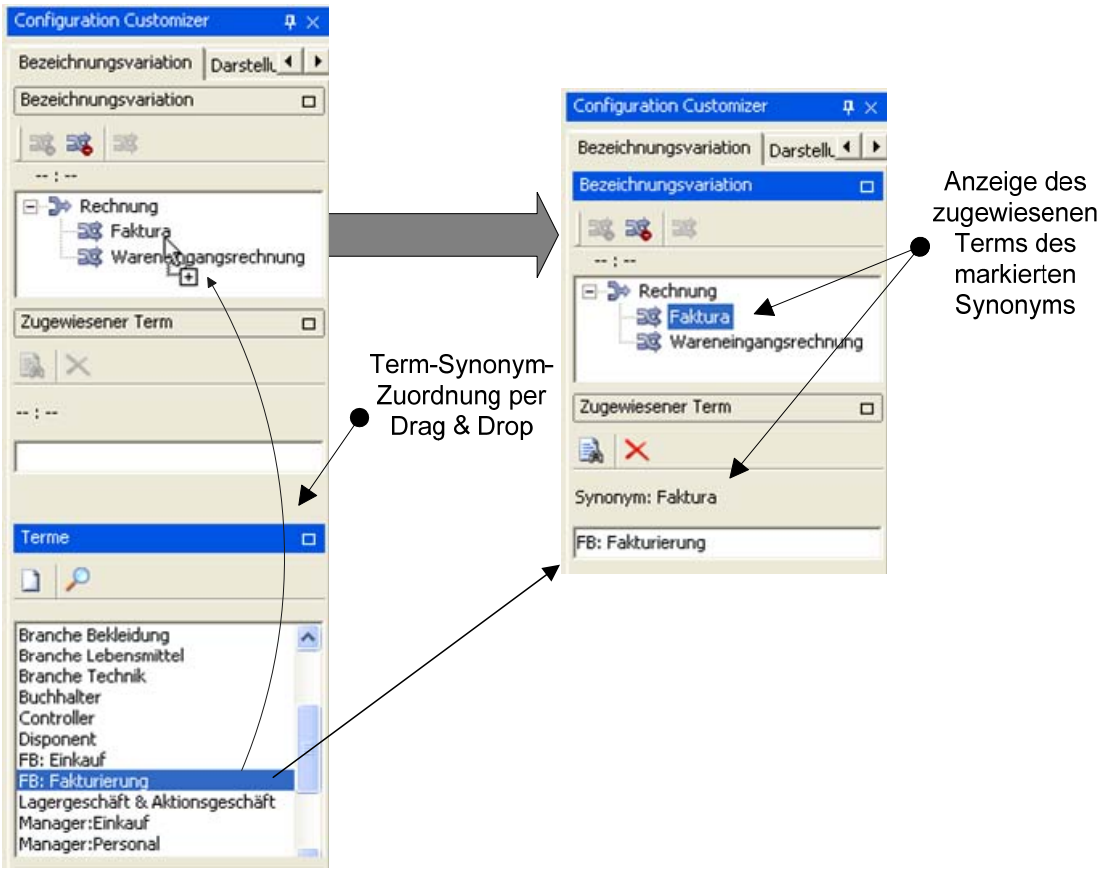



**Abb. 51:** Zuweisung von Termen zu Synonymen

Nachdem die Konfiguration des Austausches von Synonymgruppenbezeichnungen durch Synonyme mittels der Termzuordnung abgeschlossen ist, sind Synonymgruppen abschließend Modellelementen zuzuordnen. Abb. 52 stellt die einzelnen Arbeitsschritte dar: Die jeweilige

Synonymgruppe ist wie gewohnt per Drag & Drop auf das zu variierende Modellelement zu ziehen, woraufhin ein Fenster aufgeht, in dem das auszutauschende Textfragment zu markieren ist. Bei Erfolg der Zuordnung ist die zugeordnete Synonymgruppe des im Modell-Editor markierten Elements im Configuration Customizer abzulesen.

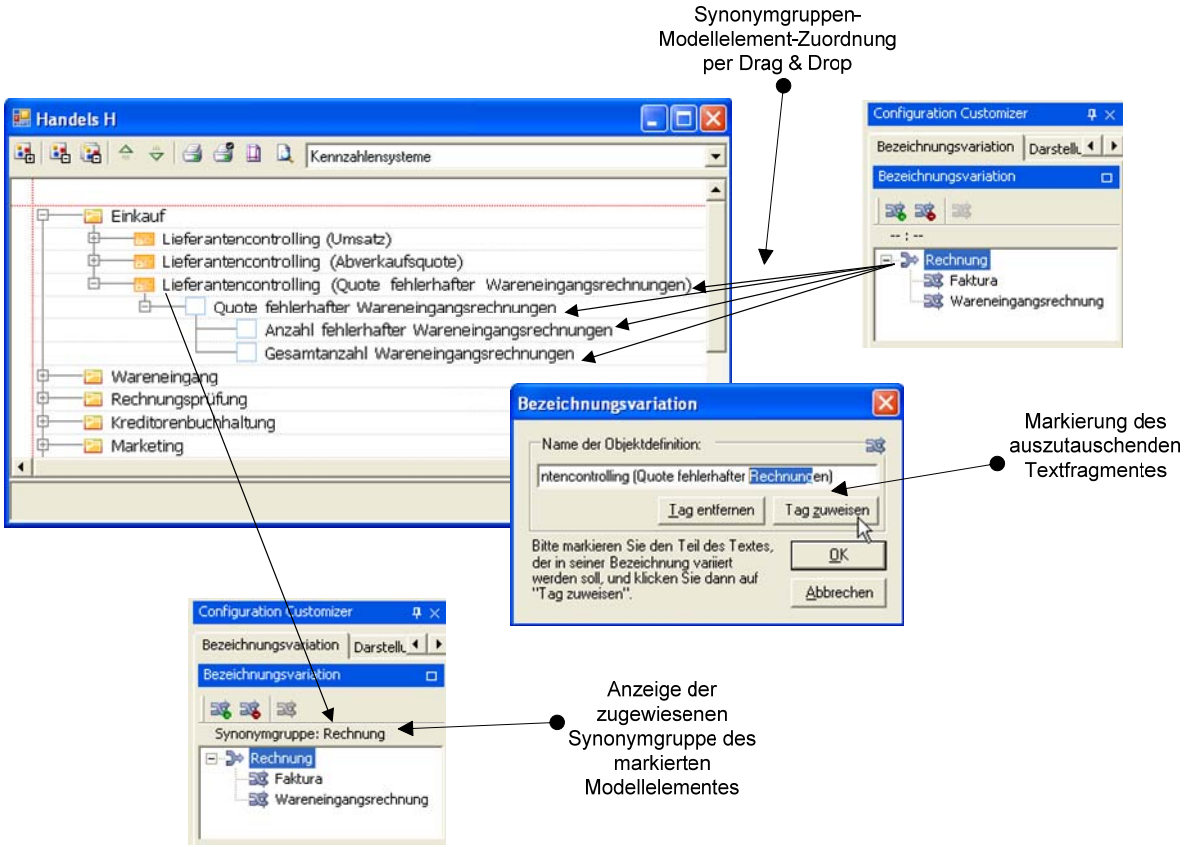



**Abb. 52:** Zuweisung von Synonymgruppen zu Modellelemente

Sobald sich anschließend ein Nutzer mit zugewiesenen Konfigurationsparametern anmeldet, die den Term eines Synonym zu `true` auswerten lassen, ändert sich die Bezeichnung eines Modellelementes im Modell-Editor. Abb. 53 zeigt die Auswirkung der Bezeichnungsvariation. Sind bereits Synonyme auf dem Element hinterlegt, können keine weiteren Synonyme gedroppt werden. Für Angehörige der Benutzergruppe der Administratoren sind von der Bezeichnungsvariation betroffene Modellelementbezeichnungen nach wie vor anhand der Tags `<BV>[Textfragment]</BV>` ersichtlich. Werten zwei oder mehr Synonyme gleichzeitig zu `true` aus, wird eine Fehlermeldung wegen Modellinkonsistenz ausgeworfen.

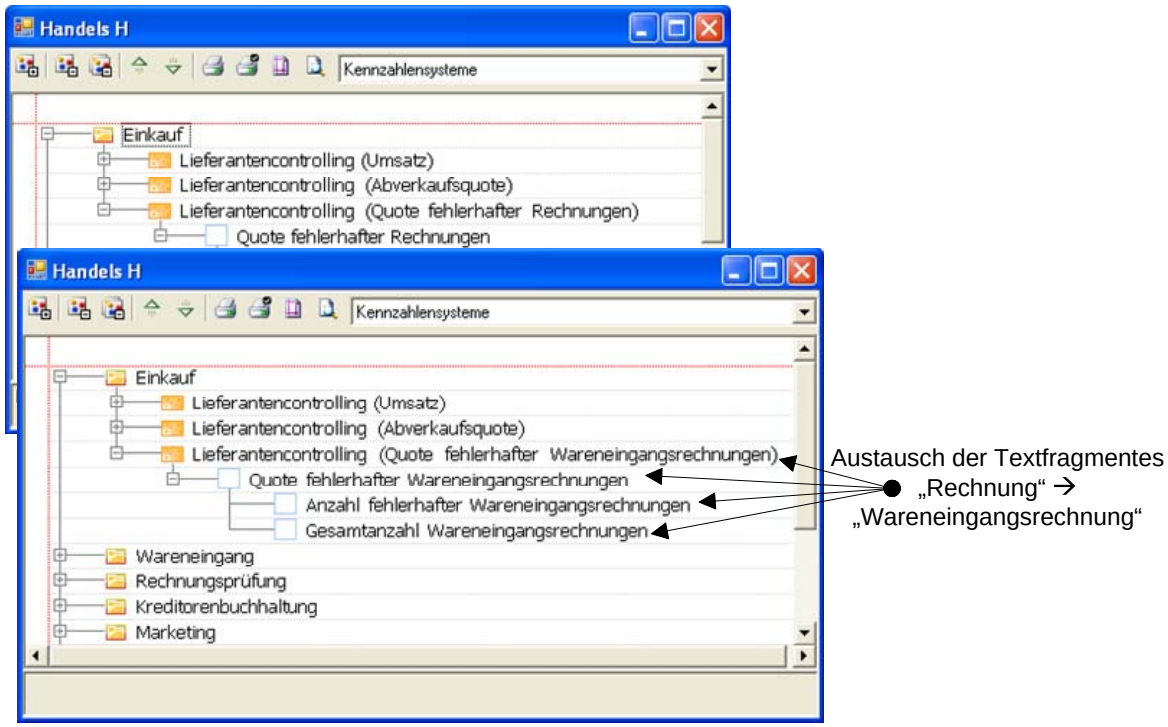



**Abb. 53:** Auswirkung der Bezeichnungsvariation

## 3.7 Darstellungsvariation der Symbole

### 3.7.1 Fachkonzept

Abhängig vom Anwendungskontext der Referenzmodelle sind die Standards bezüglich der grafischen Darstellung von Modellen unterschiedlich. Um diesen Unterschieden gerecht zu werden, ist es nötig, die Darstellung von Modellen durch eine Zuweisung von Konfigurationsparameterkombinationen verändern zu können.

Im H2-Toolset wurden Symbolmengen, so genannte Imagesets, eingeführt, mit denen die Symbole der Objekttypen einer Sprache ausgetauscht werden können. Diesen Imagesets können Terme zugeordnet werden, welche die Variation steuern. Bei der Darstellungsvariation im H2-Toolset werden dem zufolge nur ganze Imagesets ausgetauscht und nicht einzelne Symbole. Bei der Definition eines Imagesets muss nicht zwangsläufig jedem Objekttyp ein neues Symbol zugeordnet werden. Ein Imageset wird aktiviert, wenn dem angemeldeten Benutzer die Konfigurationsparameter zugeordnet wurden, welche bewirken, dass der dem Set zugewiesene Term zu `true` auswertet. Der Referenzmodellierer hat dafür Sorge zu tragen, dass von allen Termen, welche den Imagesets einer Sprache zugewiesen wurden höchstens ein Term bei jeder möglichen Konfigurationsparameterkombination zu `true` auswertet. Missachtet er diese Regel, so wird das erste Imageset aktiviert, dessen Term zu `true` auswertet.

Ist ein Imageset aktiv, so werden im Modell-Editor die Symbole der Objekte ausgetauscht, für deren Objekttyp im Set ein variiertes Symbol eingetragen ist. Für alle anderen Objekte wird das Standardsymbol des Objekttypen verwendet.

Abb. 54 veranschaulicht diesen Sachverhalt.

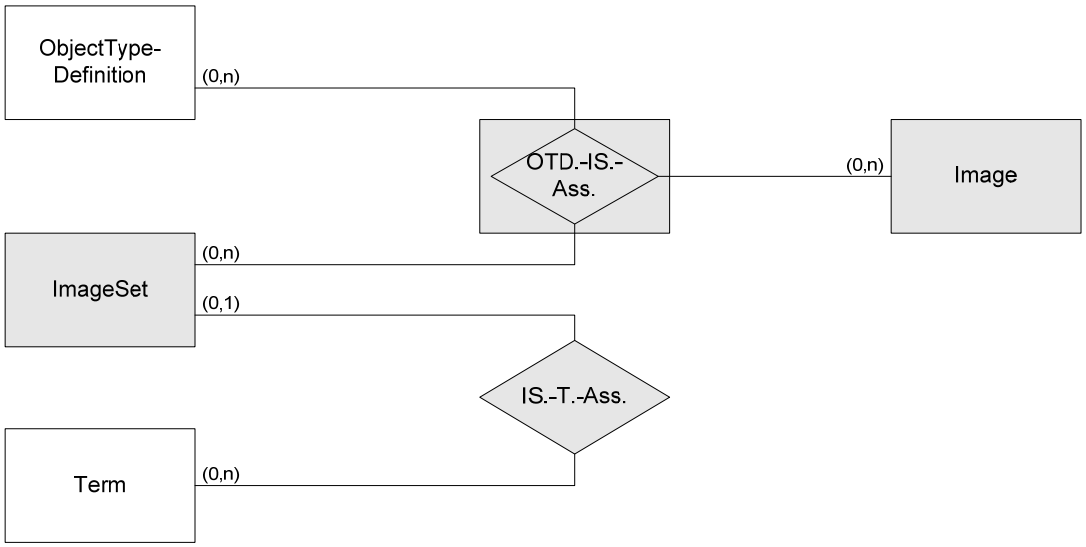



**Abb. 54:** ER-Modell für die Darstellungsvariation der Symbole

### 3.7.2 DV-Konzept und Implementierung

Die für die Darstellungsvariation benötigten Daten werden in den vier Tabellen *Image*, *ImageSet*, *ImageSetImageObjectTypeDefinitionRelation* und *TermImageSetRelation* gespeichert.

*Image*

Die Symbole werden in einer eigenen Tabelle gespeichert, um sie in mehreren Sets verwenden zu können. Die Daten werden als Bitmap gespeichert. Wenn Teile eines Symbols im Modell-Editor transparent dargestellt werden sollen, müssen die entsprechenden Flächen in der Farbe „magenta“ gespeichert werden. Ein Symbol gehört immer eindeutig zu einer Sprache, aus diesem Grund wird die *languageId* aus der Tabelle *Language* als Fremdschlüssel ebenfalls gespeichert (vgl. Tab. 34).

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| imageId | int(11) | 1 | Id des Image |
| data | Blob | <BLOB> | Bilddaten als Bitmap |
| languageId | int(11) | 1 | Sprache, in der das Bild verwendet wird |

**Tab. 34:** Tabelle Image

*Imageset*

Die Tabelle *ImageSet* ist analog zu Tabelle *Image* aufgebaut (vgl. Tab. 35).

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| imagesetId | Int | 1 | Id des Image |
| name | Varchar | graue Images | Name des Imagesets |
| languageId | Int | 1 | Sprache, in der das Imageset verwendet wird |

**Tab. 35:** Tabelle ImageSet

*ImageSetImageObjectTypeDefinitionRelation*

In der Tabelle *ImageSetImageObjectTypeDefinitionRelation* wird die Zuordnung von Symbolen zu Objekttypen innerhalb der Imagesets gespeichert. Dabei kann jeder Kombination aus einem Imageset und einem Objekttyp höchstens ein Symbol zugewiesen werden (vgl. Tab. 36).

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| imagesetId | Int | 1 | Id eines imagesets aus der Tabelle „imageset“ |
| imageId | Int | 1 | Id eines Symbols aus der Tabelle imageid |
| objectTypeDefinitionId | Int | 1 | Id einer Objekttypdefinition aus der Tabelle „objecttypedefinition“ |

**Tab. 36:** Tabelle Imagesetimageobjecttypedefinitionrelation

*Termimagesetrelation*

Über die Zuordnung von Termen zu Imagesets kann beim Öffnen des Modell-Editors geprüft werden, ob der Term zum Austausch des Symbols führt. Die Zuweisung erfolgt über die Primärschlüssel der jeweiligen Tabellen.

| Feld | Typ | Beispiel | Erläuterungen |
|---|---|---|---|
| termId | Int | 1 | Id des Terms |
| imageId | Int | 1 | Id des Images |

**Tab. 37:** Tabelle Termimagesetrelation

*Darstellungsvariation mit dem Configuration Customizer*

Im H2-Toolset ist es möglich, die Symbole der Objekte eines Modells in Abhängigkeit der Konfigurationsparameterkombination auszutauschen, welche dem angemeldeten Benutzern zugewiesen sind. Hierzu ist es nötig, so genannte Imagesets anzulegen. In einem Imageset sind Alternativsymbole definiert, welche die Standardsymbole der Objekttypen ersetzen sobald ein hinterlegter Term zu `true` auswertet (vgl. Abb. 55).

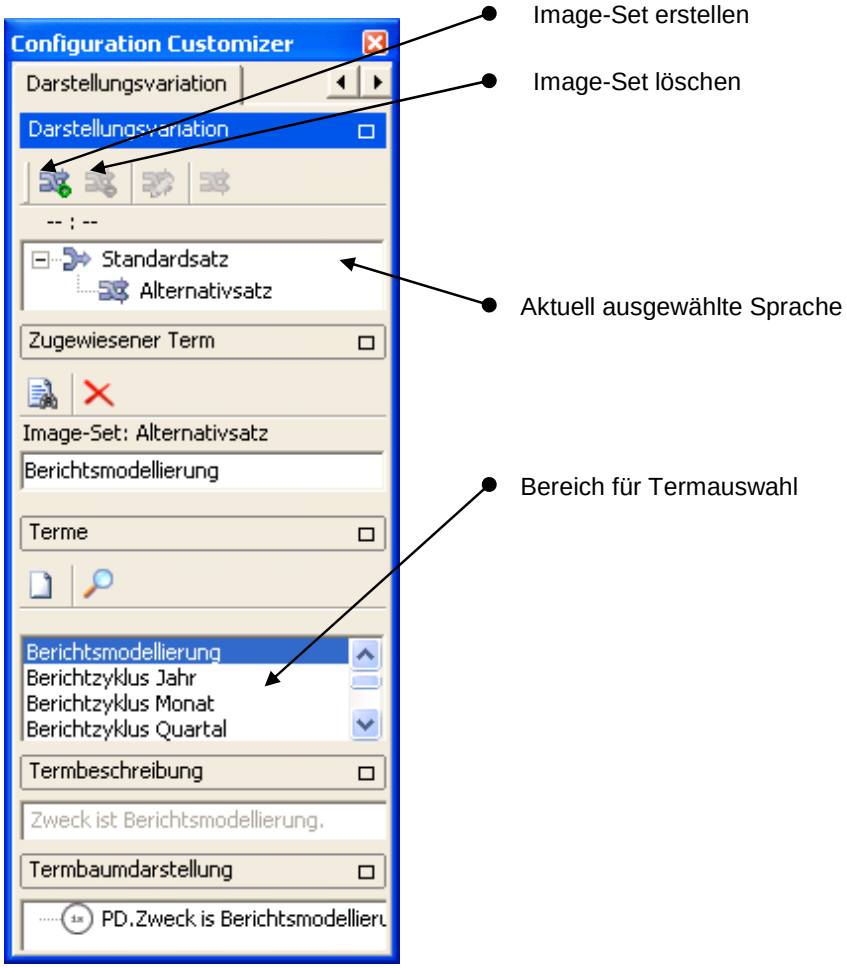



**Abb. 55:** Darstellungsvariation

Um ein Imageset anzulegen, muss zuerst eine Sprache ausgewählt werden, für die das Imageset erstellt werden soll. Anschließend muss im Configuration Customizer der Reiter Darstellungsvariation ausgewählt werden. Durch das Klicken auf den Button „Imageset erstellen“ wird der Editor zum Anlegen der Imagesets geöffnet (vgl. Abb. 56).

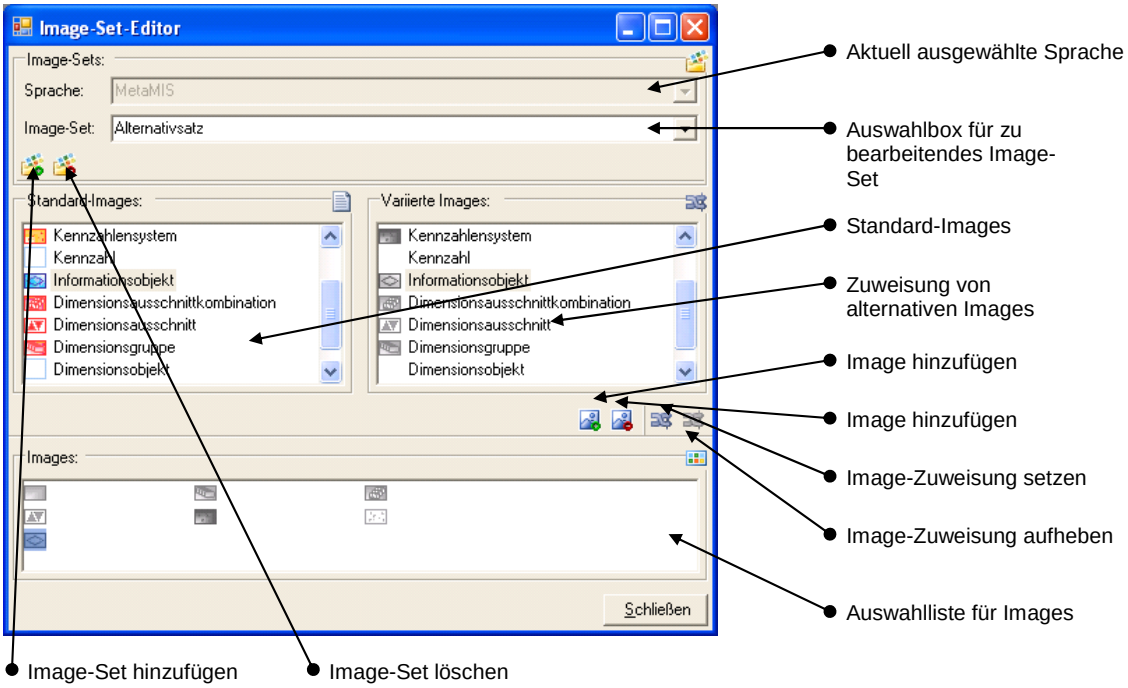



**Abb. 56:** Anlegen eines Image-Sets

Im linken oberen Teil des Fensters werden die in der Sprache enthaltenen Objekttypen mit ihren Standardsymbolen dargestellt. Im rechten oberen Teil sind ebenfalls die in der Sprache enthaltenen Objekttypen zu sehen. Da bis jetzt keine Alternativsymbole definiert wurden, ist hier noch kein Symbol eingetragen.

Im unteren Teil des Fensters ist eine Liste von Alternativsymbolen, die durch Drag & Drop den Objekttypen des rechten Fensterbereichs zugewiesen werden können. Es ist auch möglich, diese Zuweisung durch Markierung jeweils einer Objekttypdefinition im rechten oberen Bereich des Fensters und eines Symbols und einen darauf folgenden Klick auf „Image-Zuweisung setzen“ vorzunehmen. Ist das gewünscht Symbol noch nicht in der Auswahlliste, muss es zuerst angelegt werden. Dazu wird der Button „Image hinzufügen“ geklickt und eine Bilddatei ausgewählt.

Nicht mehr benötigte Symbole werden markiert und durch einen Klick auf den Button „Image löschen“ gelöscht. Nicht mehr benötigte Symbol-Objekttyp-Verknüpfungen können entfernt werden, indem der entsprechende Eintrag im rechten oberen Fensterbereich markiert und anschließend mit dem Button „Image-Zuweisung aufheben“ gelöscht wird.

Wurden alle gewünschten Alternativsymbole den Objekttypen im rechten Fensterbereich zugewiesen, ist das Imageset vollständig definiert und kann gespeichert werden. Nach dem Schließen des Editors kann nun analog zu den anderen Selektions- und Variationsmechanismen ein Term im Configuration Customizer auf das erstellte Imageset gezogen werden um

ihn dem Set zuzuweisen. Es muss darauf geachtet werden, dass die Terme der unterschiedlichen Imagesets einer Sprache niemals gleichzeitig zu `true` auswerten, da ansonsten nicht definiert ist, welches Imageset verwendet wird, und daher das erste Imageset verwendet wird das zu `true` auswertet.

Ein im Configuration Customizer markiertes Imageset kann durch einen Klick auf den Button „Imageset löschen“ gelöscht werden. Durch einen Klick auf den Button „Imageset umbenennen“ kann ein markiertes Imageset umbenannt werden.

## 3.8 Konfiguration und Zuordnung von Konfigurationsparametern zu Benutzern

### 3.8.1 Fachkonzept

Wurde das Referenzmodell vollständig modelliert und mit Termen hinterlegt, kann es anhand einer Auswahl von Konfigurationsparametern angepasst werden.

Eine Anpassung eines Referenzmodells kann sowohl durch den Einsatz in einem konkreten Unternehmen erforderlich sein, als auch durch die Anforderungen der unterschiedlichen Betrachter und Modellierer dieses Modells sinnvoll sein. Wird ein Referenzmodell an die Gegebenheiten eines Unternehmens angepasst, kann die Auswahl der Konfigurationsparameter als Auswahl der vorhandenen und relevanten Unternehmensmerkmale des zu modellierenden Betrachtungsgegenstands verstanden werden. Werden Konfigurationsparameter einem Benutzer zugeordnet so erhält dieser Einblick in den betreffenden Bereich. Die Ausblendung der übrigen Elemente kann aus Gründen der Übersichtlichkeit oder der Rechteverwaltung erforderlich sein.

Wird ein Referenzmodell an ein Unternehmen angepasst, wird in dieser Arbeit von Konfiguration gesprochen. Anhand der ausgewählten Konfigurationsparameter werden die hinterlegten Terme ausgewertet und je nachdem, ob der Term zu `true` oder `false` auswertet, wird das Element übernommen bzw. verändert oder nicht übernommen. Das daraus entstehende neue Modell wird zusätzlich zu dem ursprünglichen Referenzmodell gespeichert. Es ist komplett losgelöst und unabhängig, d. h. Veränderungen des ursprünglichen Referenzmodells haben keine Auswirkungen auf das neue Modell und vice versa.

Wird ein Referenzmodell an die Bedürfnisse eines Benutzers angepasst, werden Elemente zur Laufzeit verändert. Meldet sich ein Benutzer an, wird sowohl seine Arbeitsumgebung an die ihm zugeteilten Rechte angepasst (siehe Kap. 3.2), als auch die Modelle aufgrund der ihm zugeteilten Konfigurationsparametern verändert dargestellt. Genau wie bei der Konfiguration

werden die Terme mit den Konfigurationsparametern abgeglichen und die Modelle auf diese Weise verändert. Im Gegensatz zur Konfiguration arbeitet der Benutzer aber immer auf dem gesamten Referenzmodell (vgl. Abb. 57).

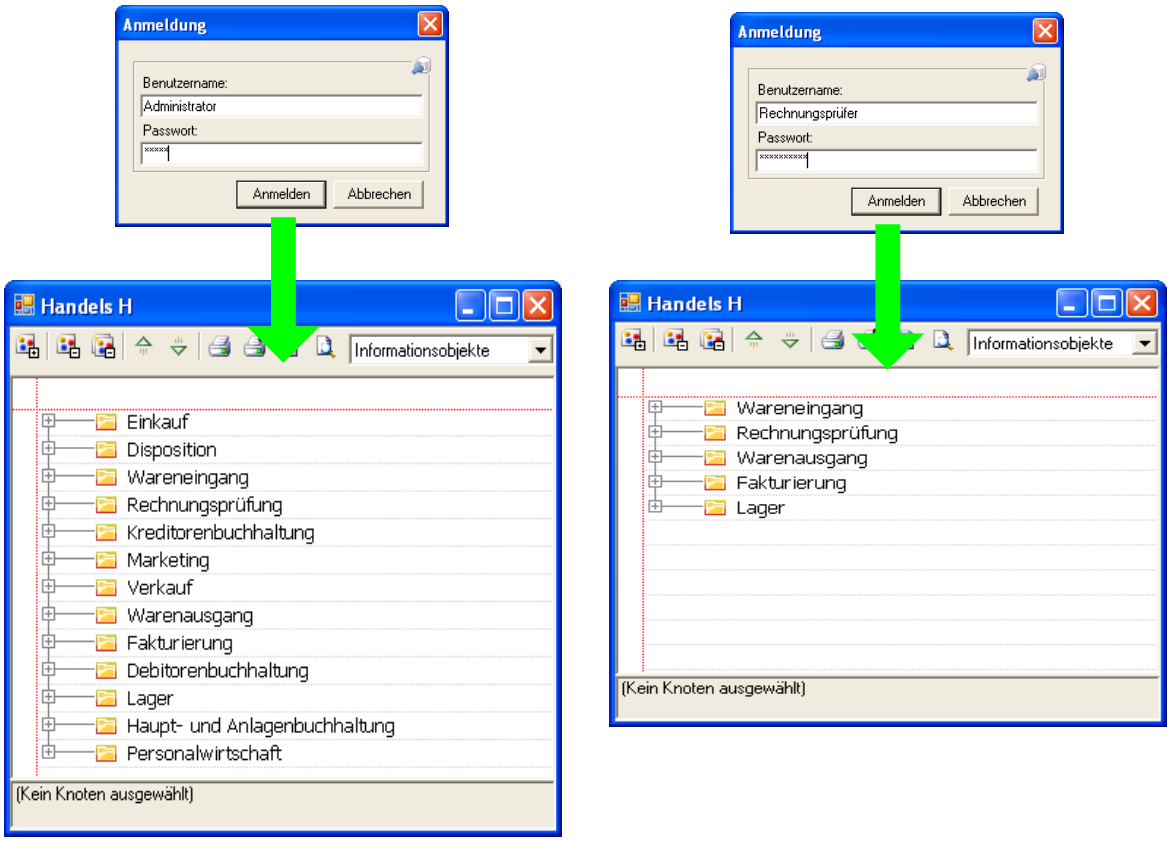



**Abb. 57:** Perspektivenwechsel bei einem Referenzmodell

### 3.8.2 DV-Konzept und Implementierung

*Konfigurationsparameter-Benutzer-Zuordnung*

Die Zuordnung von Konfigurationsparametern zu Benutzern wird durch die Relationstabelle *userconfparamrelation* zwischen der Tabelle *Usergrouprefmodelrelation* und *confparamdefinition* realisiert. Das ER-Diagramm in Abb. 58 dient zur Einordnung dieser Relationstabelle.

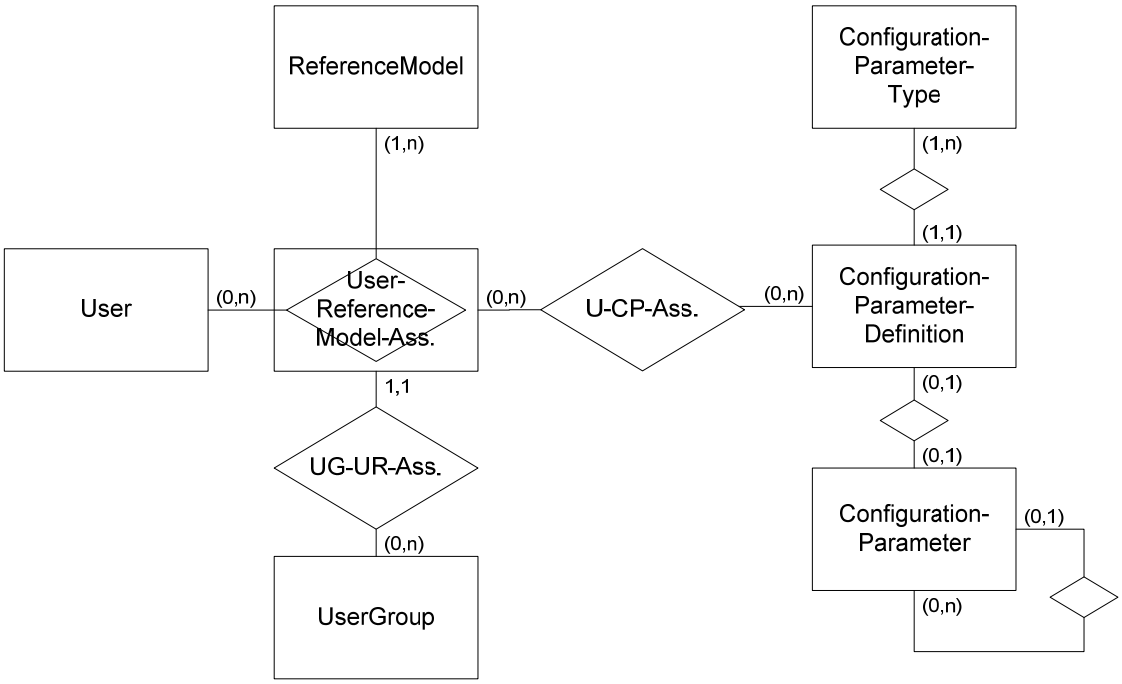



**Abb. 58:** ER-Diagramm der Zuweisung von Konfigurationsparameterdefinitionen zu Benutzern

Das Konstrukt der Konfigurationsparameter wurde schon im Abschnitt 3.3 beschrieben. Folglich sind die Tabellen *Confparam*, C*onfparamdefinition* und C*onfparamtype* bereits bekannt. Auch wurde im Abschnitt 3.2 auf die Tabellen *Usergroup*, *Userlist*, *Refmodel* und *Usergrouprefmodelrelation* eingegangen. Die Relationstabelle *Userconfparamrelation* bildet die Zuweisung von Konfigurationsparametern zu Benutzern bezogen auf ein konkretes Referenzmodell ab. Das Schlüsselattribut *id* aus der Tabelle *Userlist*, das Schlüsselattribut *id* aus der Tabelle *Refmodel* und das Schlüsselattribut *objectDefinitionId* aus der Tabelle *Confparamdefinition* gehen als Fremdschlüssel in die Tabelle *Userconfparamrelation* ein. Gemeinsam bilden sie einen zusammengesetzten Schlüssel.

| **Feld** | **Typ** | **Beispiel** | **Erläuterungen** |
|---|---|---|---|
| userid | Int(11) | 1 | Id eines Benutzers |
| refmodelId | Int(11) | 1 | Id eines Referenzmodells |
| confparamDefinitionId | Int(11) | 1 | Id einer Confparamdefinition |

**Tab. 38:** Tabelle Userconfparamrelation

*Konfigurationsparameter Benutzern zuordnen*

Die Klasse *UserConfparamAssignment* bildet die grafische Benutzerschnittstelle zur Eingabe und Pflege der Zuordnung von Konfigurationsparametern zu Benutzern in bestimmten Refe-

renzmodellen. In dieser GUI kann jeder Benutzer, der dem aktivierten Referenzmodell zugeordnet wurde, ausgewählt werden. Für diese Benutzer können nun alle dem Referenzmodell zugeordneten Konfigurationsparameter zugewiesen werden. Die Zuweisung ist mit Checkboxen realisiert. Welche Konfigurationsparameter einem Benutzer zugeordnet wurden, wird in einem Cache in der Klasse *UserGroupRefmodelRelation* gespeichert, welche für die Aktualisierung der Datenbank verantwortlich ist.

## 3.9 Erstellung und Verwaltung von Kantentypen

### 3.9.1 Fachkonzept

In einem Modell können Objekte in Bezug zueinander gesetzt werden. Die Verbindung der Objekte zueinander bezeichnet man als Kante. Dabei besteht die Möglichkeit, die Beziehung der Objekte zueinander zu typisieren und hierdurch zusätzliche Informationen zu der Art der Verbindung der Objekte zu machen. Es ist also möglich, unterschiedliche Formen der Verbindung abbilden zu können.

Das H2-Toolset wurde dahingehend erweitert, den Typ der Beziehung zwischen zwei Objekten angeben zu können. Dabei ist es möglich, zwischen zwei Objekten mehrere Beziehungen zu spezifizieren, um den Informationsgehalt der Verbindung beliebig erweitern zu können und flexibel zu halten. Werden mehrere Kantentypen einer Beziehung zugeordnet, so wird über ein separates Symbol eine Gruppe von Kantentypen deutlich gemacht und die Übersichtlichkeit des Modells beibehalten. Die Erstellung eines Modells und damit die Objekttypen, die eine Beziehung eingehen können, erfolgt anhand definierter Regeln. Hierdurch wird erreicht, dass ein stets syntaktisch richtiges Modell erzeugt wird. Die Erweiterung um Kanten weicht von diesem Konzept nicht ab. Jeder definierten Regel lassen sich ausgewählte Kantentypen zuordnen, so dass der Modellierer in zwei Hinsichten unterstützt wird. Dadurch bleibt das Modell stets syntaktisch korrekt und der Modellierer wird in seiner Arbeit durch eine eingeschränkte Auswahl unterstützt.

Das ER-Diagramm in Abb. 59 zeigt durch die Entitytypen *Edgetype* und *ContextRule* sowie den verbindenden Relationshiptypen *Edgetypecontextrule* die möglichen Kantentypen, die der Beziehung zweier Objekttypen zur Verfügung stehen. Über den Relationshiptypen *Edge* erfolgt die Datenhaltung der Kanten zwischen den Objektausprägungen in einem Modell. Da alle Modelle im H2-Toolset Baumstrukturen haben, hat jede Objektausprägung nur eine eingehende Kante. Hier reicht also die Zuordnung der Kantentypen zu der Objektausprägung der unteren Hierarchiestufe. Durch die n:n-Zuordnung wird die Zuordnung von mehreren Kantentypen zu einer Ausprägung erreicht.

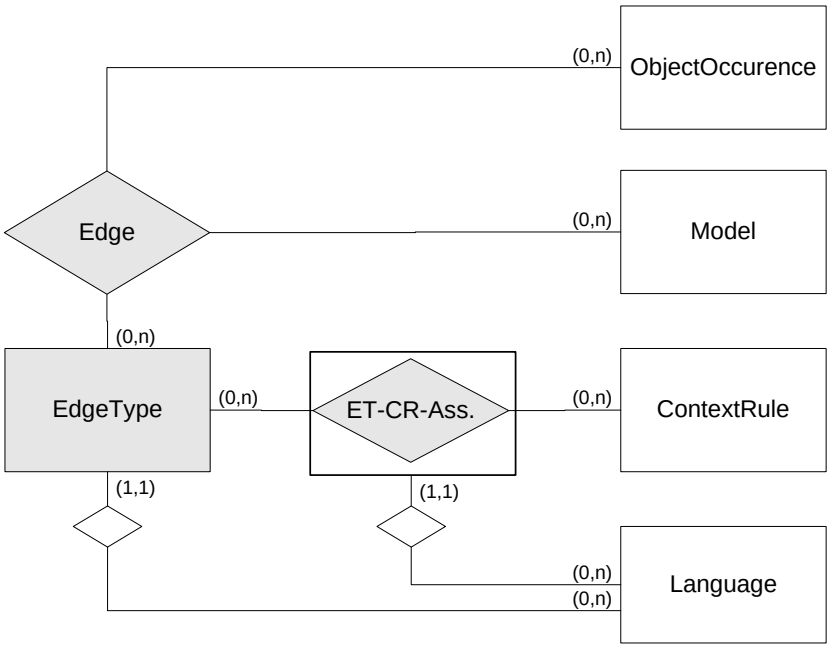



**Abb. 59:** ER-Diagramm der Kanten

Für die Realisierung sind Änderungen an folgenden Paketen und Dateien gemacht worden:

- DatabaseMapping::Language::Edgetype.cs
- DatabaseMapping::Language::Edgetypecontextrulerelation.cs
- DatabaseMapping::Language::Contextrule.cs
- DatabaseMapping::Language::Language.cs
- DatabaseMapping::Model::Edge.cs
- DatabaseMapping::Model::Model.cs
- DatabaseMapping::Model::ObjectOccurence.cs
- ModelEditing::Editor.cs
- ModelEditing::EditorServices.cs
- ModelEditing::EdgeWindow.cs
- LanguageEditing::LanguagEditor.cs
- Configuration::Adaption::Adaptionexporter.cs

Eine Übersicht über die beteiligten Klassen ist Abb. 60 zu entnehmen.

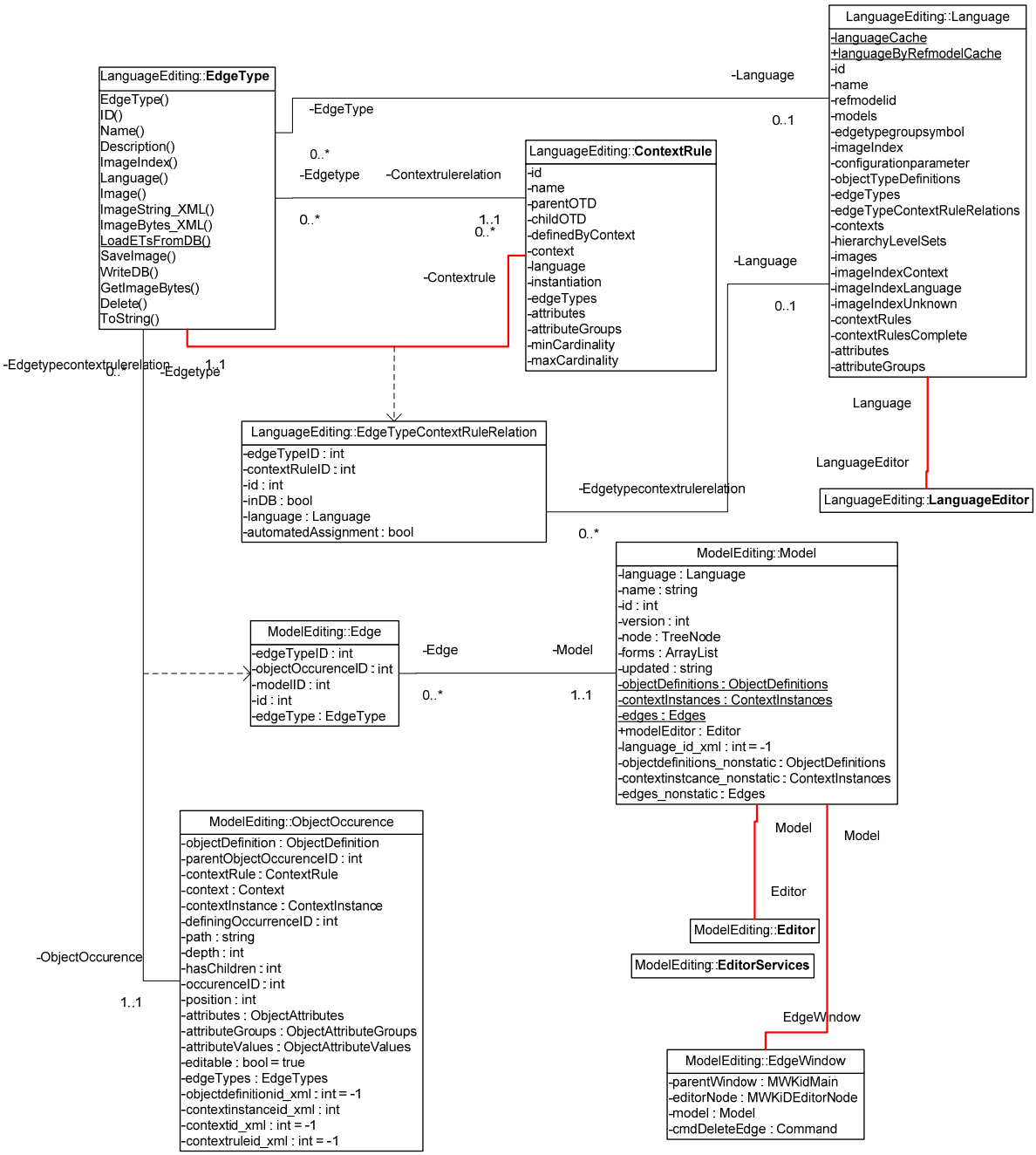



**Abb. 60:** Klassendiagramm der Kanten

### 3.9.2 DV-Konzept und Implementierung

Die Daten in den Tabellen *EdgeType*, *EdgeTypeContextRuleRelation* und *Edge* werden initial in die Klassen der Pakete *DatabaseMapping::Language* und *DatabaseMapping::Model* abgebildet. Die Klassen sind *EdgeType*, *EdgeTypeContextRuleRelation* und *Edge*.

Über das Laden der ContextRule werden die Kantentypen zur jeweiligen ContextRule ebenfalls geladen.

*Datenbankschema*

Die Tabelle *EdgeTypeContextRuleRelation* enthält die Zuordnung von Kantentypen zu den ContextRules (vgl. Tab. 39).

| **Feld** | **Typ** | **Beispiel** | **Erläuterungen** |
|---|---|---|---|
| edgetype_Id | int(11) | 1 | Id des Kantentypen |
| language_Id | int(11) | 1 | Id der Sprache |
| automatedAssignment | tinyint(1) | 0 | Bit, ob eine automatische Zuweisung von Kantentypen erfolgen soll bei Anlage der Objektausprägung |
| contextrule_Id | int(11) | 1 | Id der Kontextregel |

**Tab. 39:** Tabelle EdgeTypeContextRuleRelation

Die Tabelle *Edge* gibt die Zuordnung des Kantentyps zu einer Objektausprägung an. Die *model_Id* wird aus Performanzgründen gespeichert (vgl. Tab. 40).

| **Feld** | **Typ** | **Beispiel** | **Erläuterungen** |
|---|---|---|---|
| id | int(11) | 1 | Id der Kante |
| objectoccurence_Id | int(11) | 1 | Zugehörige Objektausprägung. Hiermit ist die eingehende Kante gemeint. |
| model_Id | int(11) | 1 | Id des Modells |
| edgetype_Id | int(11) | 1 | Id des Kantentypen |

**Tab. 40:** Tabelle Edge

In der Tabelle *Edgetype* werden alle im Spracheditor erzeugten Kantentypen angelegt. Das hier angegebene Bild wird im Modell angezeigt (vgl. Tab. 41).

| **Feld** | **Typ** | **Beispiel** | **Erläuterungen** |
|---|---|---|---|
| id | Int(11) | 1 | Id des Kantentypen |
| name | Varchar(100) | kennt | Name |
| description | Varchar(255) | übergeordnetes Element kennt untergeordnetes | Beschreibung |
| languageId | Int(11) | 1 | Id der Sprache |
| image | Blob | <BLOB> | Bild des Kantentypen |

**Tab. 41:** Tabelle EdgeType

***Nutzung von Kantentypen und Kanten***

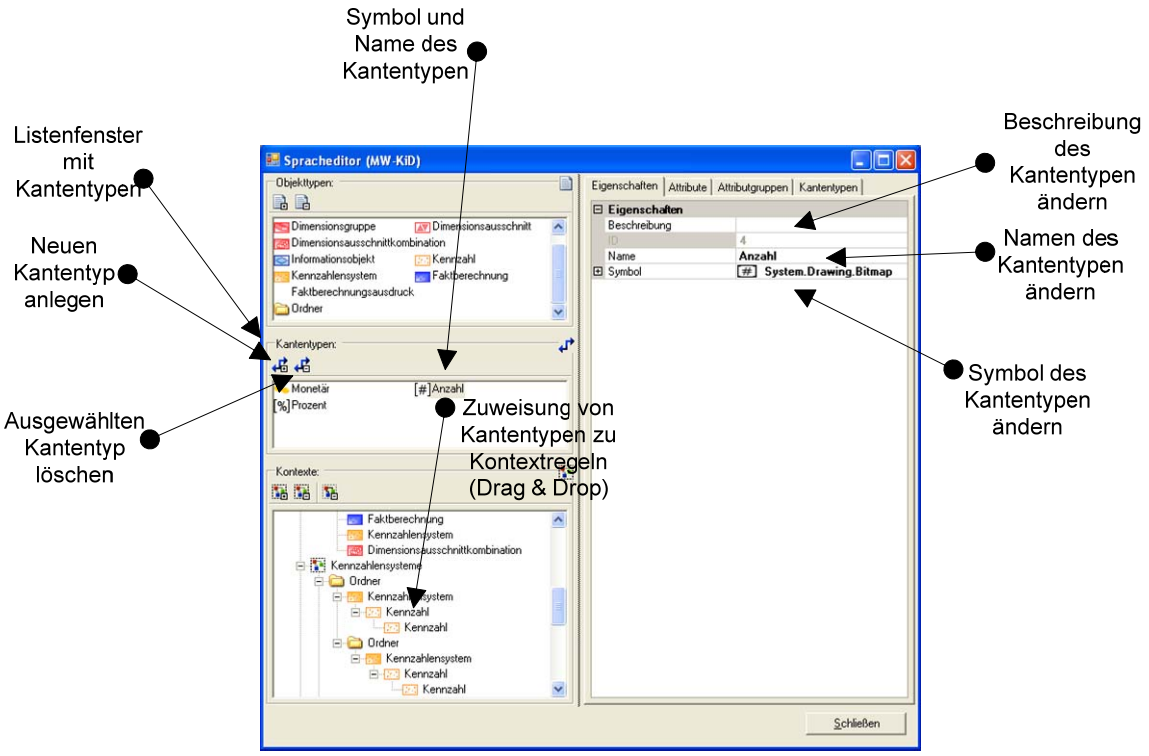



**Abb. 61:** Nutzung von Kantentypen

*Verwalten von Kantentypen*

Im Gruppenfeld „Kantentypen“ können Kantentypen gelöscht und hinzugefügt werden. Zudem enthält es eine Liste der im System angelegten Kantentypen.

Durch das Auswählen des mit einem „plus“ versehenen Buttons wird ein neuer Kantentyp erzeugt. Dieser wird mit dem initialen Namen „unbenannt“ angelegt und enthält kein Symbol. Im Reiter „Eigenschaften“ werden dann zusätzliche Informationen zu den Kantentypen angezeigt. Über dieses Dialogfeld kann dann zudem der Name editiert werden, eine Beschreibung vergeben und das Symbol für den Kantentypen ausgewählt werden.

Durch das Auswählen des zweiten Buttons („minus“) kann ein im Listenfeld ausgewählter Kantentyp gelöscht werden.

*Zuordnung von Kantentypen zu Kontextregeln*

Um die Kantentypen in einem Modell an Elemente annotieren zu können, ist es notwendig, die Kantentypen den Kontextregeln zuzuordnen. Dies geschieht, indem ein Kantentyp ausgewählt wird und per Drag and Drop auf die entsprechende Kontextregel gezogen wird. Zusätz-

liche Informationen zu den Kantentypen können durch die Auswahl des Reiters „Kantentypen“ angezeigt werden. In dem Gruppenfeld „zugewiesene Kantentypen“ wird angezeigt, welche Kantentypen der Kontextregel zugewiesen wurden. Die Anzeige des Inhaltes dieses Fensters erfolgt durch die Auswahl einer Kontextregel.

*Gruppensymbol für mehrere Kanten*

Im Reiter „Kantentypen“ unter dem Gruppenfeld „Standardsymbole“ kann das Symbol für die Gruppierung von Kantentypen angegeben werden. Diese findet Verwendung, wenn zu einem Element mehrere Kantentypen zugewiesen werden. Hierüber erfolgt im Endeffekt der Vermerk, dass das Element mehrere Kantentypen zugewiesen bekommen hat.

*Automatische Zuweisung von Kanten*

Die Zuweisung von Kantentypen zu Elementen kann durch das Auswahlfeld „Automatisch zuweisen“ vereinfacht werden. Ist das Auswahlfeld angewählt, wird durch das Anlegen eines Elementes im Modell die der Kontextregel zugewiesenen Kantentypen automatisch zugewiesen.

*Verwalten von Kanten im Modell*

Das Anlegen von Kanten in einem Modell erfordert Kontextregeln, denen Kantentypen zugewiesen wurden und Elemente, die nach diesen Kontextregeln aufgebaut wurden.

Durch die Auswahl des Elementes mit der rechten Maustaste öffnet sich ein Menüfeld. Der Eintrag „+ Kantentyp >“ enthält alle Kantentypen, die dem Element zugewiesen werden können. Durch die Auswahl eines Kantentypen erfolgt die Zuweisung und das Kantensymbol wird angezeigt. Neben dem Kantensymbol weist zudem das „PropertyWindow“ „Kanten“ darauf hin, welche Kanten dem Element zugeordnet wurden. Über dieses „PropertyWindow“ können angelegte Kanten auch wieder gelöscht werden (vgl. Abb. 62).

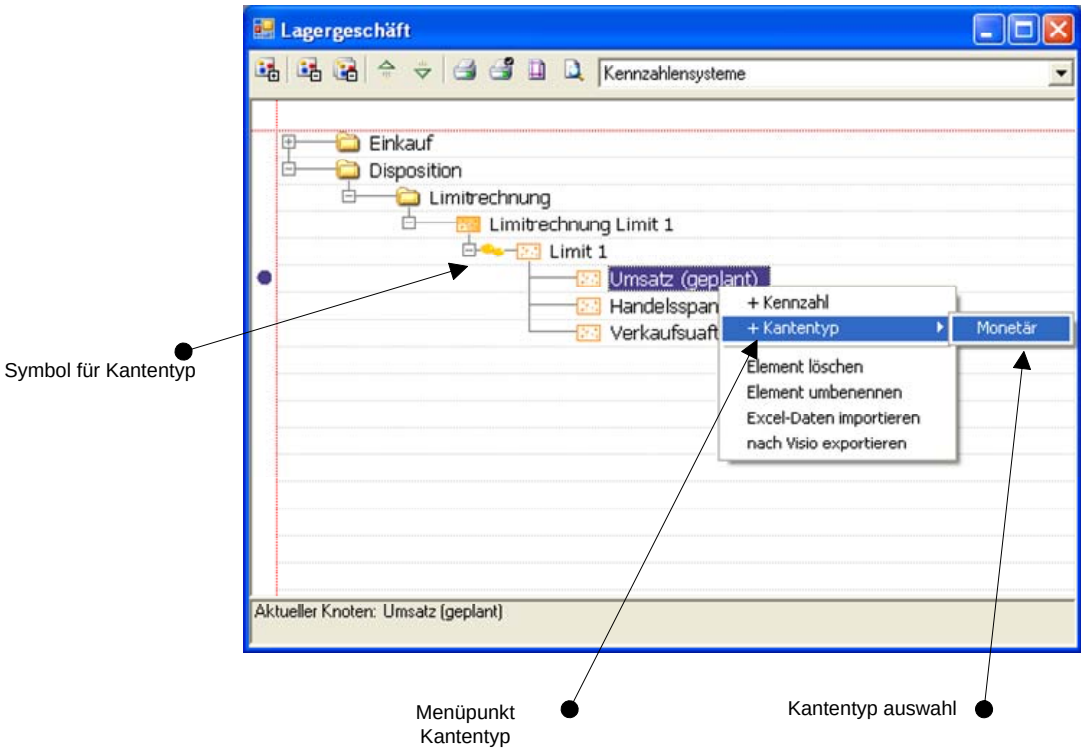



**Abb. 62:** Anlegen von Kanten

# 4 Status quo und Entwicklungsperspektiven konfigurativer Referenzmodellierung

Referenzmodelle beinhalten umfassendes Wissen für eine Klasse von Problemen. In der Praxis ist dieses Wissen jedoch zumeist nicht unmittelbar nutzbar, da sich die Spezifika eines individuellen Unternehmens signifikant von denen eines allgemeingültigen Modells unterscheiden. Die Anpassung von Referenzmodellen an die Anforderungen bereitet oftmals Probleme, da Inkonsistenzen und Modellierungsfehler die Qualität der angepassten Modelle reduzieren.

Das Prinzip der konfigurativen Referenzmodelle folgt einer domänenspezifischen Anpassung nach bestimmten Regeln. Die Modellvarianten des Referenzmodells werden ex ante durch den Modellierer festgelegt. Um die Anwendung der Konfigurationsmechanismen möglichst effizient zu gestalten, ist eine Differenzierung der Konfigurationsmechanismen in unterschiedliche Granularitätsgrade sinnvoll.

Die *Modelltypselektion* trägt dem Umstand Rechnung, dass die Relevanz ganzer Modelltypen perspektivenspezifisch variiert. Aus Sicht des Modellanwenders werden die für seine Perspektive nicht relevanten Modelltypen ausgeblendet.

Modelltypen sind durch die in ihnen enthaltenen Elementtypen und ihren Beziehungen untereinander charakterisiert. Mit der *Elementtypselektion* lassen sich zu einzelnen Modelltypen Varianten bilden, indem die perspektivenspezifisch zugewiesene Relevanz eines Elementtyps zu einer Modellvariante überprüft wird.

Eine differenziertere Selektionsmöglichkeit ermöglicht die *Elementselektion*. Diese operiert nicht mehr auf Elementtypen, sondern auf Instanzen dieser Typen.

Die Möglichkeit, dass identische Sachverhalte in unterschiedlichen Kontexten unterschiedliche Bezeichnungen besitzen, wird über den Konfigurationsmechanismus der *Bezeichnungsvariation* abgebildet. Dieser ermöglicht einen von den Konfigurationsparameterausprägungen abhängigen Austausch von Begriffen.

Im Gegensatz zu den vorangegangenen vier (die konzeptionellen Sprachaspekte betreffenden) Konfigurationsmechanismen ermöglicht die *Darstellungsvariation* die Modifikation der repräsentationellen Sprachaspekte des Modells.

Das Konzept der konfigurativen Referenzmodellierung existierte lange Zeit nur auf fachkonzeptioneller Ebene. Die prototypische Umsetzung der oben genannten Konfigurationsmechanismen im H2-Toolset beweist die Umsetzbarkeit dieses Konzepts. So wird in der Literatur

oftmals die Anwendung konfigurativer Referenzmodellierungsmechanismen in einem Satz mit der Notwendigkeit einer computergestützten Werkzeugunterstützung genannt. Diese wurde im H2-Toolset umgesetzt. Es stellt eine adäquate Implementierung der aufgeführten Mechanismen bereit und kann seine Praxistauglichkeit im Projekteinsatz unter Beweis stellen. Obwohl die Entwicklung und Optimierung der konfigurativen Komponenten noch nicht abgeschlossen sind, eröffnet diese Umsetzung bereits weitreichende Anwendungsperspektiven. Gleichwohl ist darauf hinzuweisen, dass in der Weiterentwicklung der Konfigurationsmechanismen noch weiterer Forschungsbedarf besteht.

## Literaturverzeichnis

Becker, J.; Delfmann, P. und Knackstedt, R. (2004). Konstruktion von Referenzmodellierungssprachen: Ein Ordnungsrahmen zur Spezifikation von Adaptionsmechanismen für Informationsmodelle. Wirtschaftsinformatik, 46 (4), 251-264.

Becker, J.; Delfmann, P.; Knackstedt, R. und Kuropka, D. (2002). Konfigurative Referenzmodellierung. In Wissensmanagement mit Referenzmodellen - Konzepte für die Organisations- und Anwendungssystemgestaltung (Hrsg.: J. Becker and R. Knackstedt), Physica, Heidelberg.

Becker, J.; Delfmann, P. und Rieke, T. (2004). RefMod06 - Wiederverwendung fachkonzeptioneller Softwaremodelle für kleine und mittlere Softwareunternehmen durch adaptive, komponentenorientierte Referenzmodellierung. In Proceedings of the Eröffnungskonferenz Software Engineering 2006, Berlin.

Becker, J.; Janiesch, C.; Seidel, S. and Brelage, C. (2006). A Framework for Situational and Evolutionary Language Adaption. In Proceedings of the 15th International Conference on Information Systems Development (ISD 2006), Budapest.

Becker, J.; Janiesch, C.; Seidel, S.; Brelage, C. und Crisandt, J. (2005). Specifying Modeling Languages with H2. In Proceedings of the 1st Workshop on Meta-Modelling and Corresponding Tools (WoMM 2005) (Hrsg.: U. Frank, J. Jung and L. Kirchner), S. 9-11, Essen.

Becker, J. und Knackstedt, R. (2003). Konstruktion und Anwendung fachkonzeptioneller Referenzmodelle im Data Warehousing. In Wirtschaftsinformatik 2003/Band II. Medien - Märkte - Mobilität. (Hrsg.: W. Uhr, W. Esswein and E. Schoop), Heidelberg, S. 415 - 434.

Becker, J. und Knackstedt, R. (2004). Referenzmodellierung im Data-Warehousing - State-of-the-Art und konfigurative Ansätze für die Fachkonzeption. Wirtschaftsinformatik, 46 (1), S. 39-49.

Crawford, C. (2003). Core Components Technical Specification - Part 8 of the ebXML Framework.

Holten, R. (2003). Specification of Management Views in Information Warehouse Projects. Information Systems, 28 (7), 709-751.

Schütte, R. (1998). Grundsätze ordnungsmäßiger Referenzmodellierung. Konstruktion konfigurations- und anpassungsorientierter Modelle,. Dissertation. Gabler, Wiesbaden.

vom Brocke, J. (2003). Referenzmodellierung: Gestaltung und Verteilung von Konstruktionsprozessen. Dissertation. Berlin.

# Anhang: Implementierungstechnische Details

*Objektbaum*

Zur Vereinfachung der Datenbankspeicherung sowie der XML Serialisierung wurde besonderer Wert darauf gelegt, die Objekte in einer Baumstruktur mit einem Referenzmodell-Objekt als Wurzel zu verbinden. Um die Assoziationen zwischen den Objekten zu erstellen, wurden den jeweiligen Klassen entsprechende Instanzvariablen zur Aufnahme eines oder mehrerer Objekte hinzugefügt Bei der Auswahl der Datentypen wurde insbesondere der Aufnahme mehrerer Objekte eine hohe Bedeutung beigemessen.

Hierbei wurden zwei unterschiedlich Ansätze verfolgt: Teilweise finden sich in den Klassen eigene, vom Typ Hashtable abgeleitete Collection-Klassen (z. B. *Languages*). Diese Klassen bieten u. a. eine *GetById*-Methode, um aus dem zugrunde liegenden Hashtable direkt ein Objekt des spezifizierten Typs (z. B. *Language*) zu extrahieren. Diese Implementierungsweise ist identisch mit der alten Toolset-Version, da dieses mit dem .Net Framework 1.1 realisiert wurde und dort noch keine generischen Datentypen zur Verfügung standen. Die erweiterte Version des Toolsets basiert jedoch auf dem .Net 2.0 Framework, in welchem u. a. als Neuerung generische Datentypen integriert sind.

Als Nachteil der eigenen, abgeleiteten Collection-Klassen ist zu nennen, dass die *GetAll*-Methode eine untypisierte ArrayList zurückliefert, so dass bei dem Zugriff auf diese Liste keine Typsicherheit garantiert ist. Da die Klassen vom Typ Hashtable abgeleitet sind, kann weiterhin direkt mit dem Key-Accessor-Operator ein Objekt aus der Collection ausgelesen werden, welches untypisiert ist. Deshalb wurde im Verlaufe der Implementierung dazu übergangen, die generischen Datentypen Dictionary und List zu verwenden. Je nach Verwendungskontext (iterativer Durchlauf bzw. direkter Zugriff per Key) wurde hier entschieden, ob eine Liste oder ein Dictionary zum Einsatz kommt.

*Caching-Konzept*

Als Caches werden statische, auf die jeweiligen Objekte typisierte Dictionaries verwendet. Als Schlüssel zum Auslesen der Objekte dienen die Integer-Ids aus der Datenbank. Ein Cache ist von außerhalb der jeweiligen Klasse nicht zugreifbar. Zugriffe auf den Cache finden nur in entsprechenden Methoden der Klassen statt. Hierbei ist zu beachten, dass bei allen Datenbank-Operationen wie Lesen, Schreiben, Aktualisieren und Löschen auch der Cache aktuell gehalten wird. Wird von dem Programm ein Objekt angefordert (z. B. beim Aktivieren eines Referenzmodells eine Referenzmodell-Objekt), so prüft die aufgerufene Methode zuerst den

Cache auf dieses Objekt anhand der ID. Ist das Objekt nicht im Cache vorhanden, so wird es aus der Datenbank geladen und dem Dictionary-Cache hinzugefügt. Ein analoges Vorgehen ist bei den restlichen Datenbank Zugriffsmethoden realisiert.

Tab. 42 enthält typische Methoden einer Klasse, welche auf den jeweiligen Cache zugreifen, sowie die Zugriffsart auf den Cache:

| **Methodenname** | **Zugriffsart** |
|---|---|
| loadRefModelsFromDB | read / write |
| deleteFromDb | write |
| writeDb | write |
| getAll | read |
| getById | read |

**Tab. 42:** Cachezugriff von Methoden

*Divergenzen zwischen Entitytypen und Tabellennamen*

| **Entitytyp** | **Tabellenname** |
|---|---|
| ReferenceModel | refmodel |
| FR-UG-Ass. | Userfunction |
| ConfigurationParameterDefinition | Confparamdefinition |
| User | Userlist |
| UserReferenceModel-Ass. | Usergrouprefmodelrelation |

**Tab. 43:** Divergenzen zwischen Entitytypen und Tabellennamen

*Gesamt-ERM der neuen Komponenten*

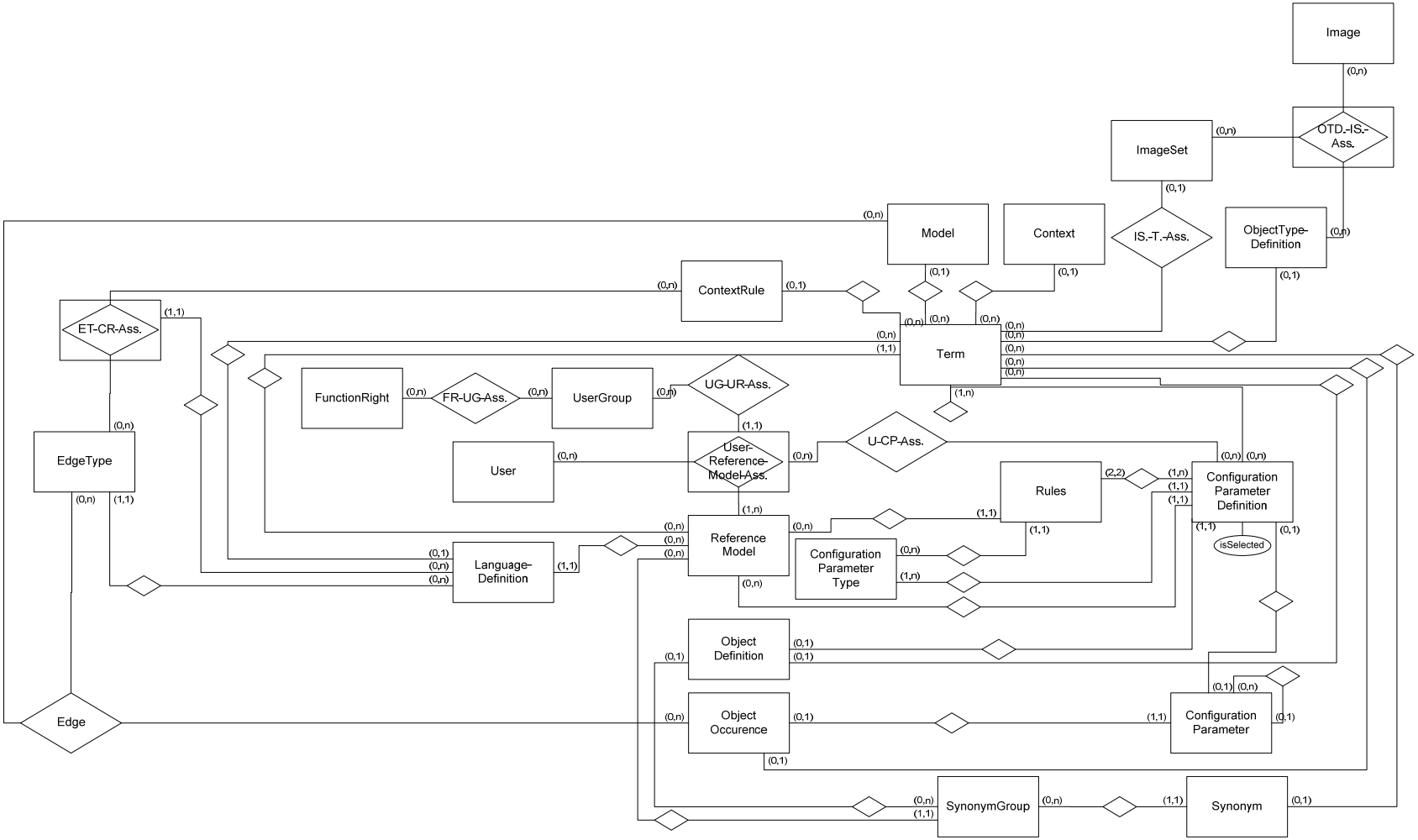



**Abb. 63:** ER-Modell der neuen Komponenten

*Gesamt-ERM*

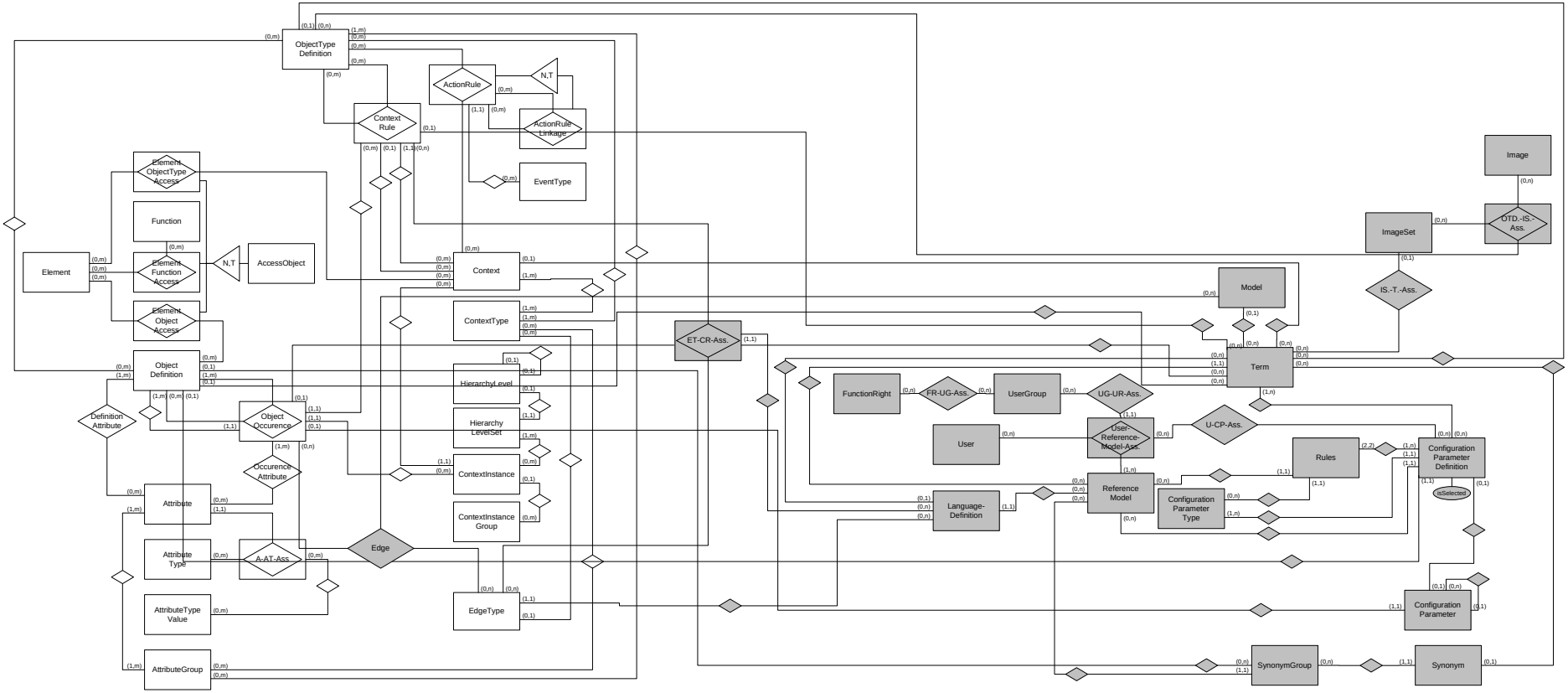



**Abb. 64:** Gesamt ER-Modell

*Klassendiagramm*

**Abb. 65:** H2 Klassendiagramm


Institut für Wirtschaftsinformatik
Leonardo-Campus 3
48149 Münster
Http://www.wi.uni-muenster.de/

ISSN 1438-3985