JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

Erweiterung von Protégé zur intuitiven Multi-Level Modellierung

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

(MAG.RER.SOC.OEC.)

im Diplomstudium

WIRTSCHAFTSINFORMATIK

Eingereicht von:
Alois Diwold

Angefertigt am:
Institut für Wirtschaftsinformatik - Data & Knowledge Engineering

BeurteilerIn:
o. Univ.-Prof. Dr. Michael Schrefl

Mitbetreuung:
Mag. Dr. Bernd Neumayr, Mag. Dr. Christoph Schütz

Linz, September 2015

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.
Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, September 2015 _____

(Alois Diwold)

Danksagung

Ich möchte mich hier an dieser Stelle bei allen Menschen bedanken, die mich bei der Erstellung der Diplomarbeit und Bewältigung meines Studiums moralisch und fachlich unterstützt haben.

Allen voran bedanke ich mich bei Herrn o. Univ.-Prof. Dr. Michael Schrefl, Herrn Mag Dr. Bernd Neumayr und Herrn Mag. Dr. Christoph Schütz, die durch ihre fachliche Kompetenz wesentlich zur Erstellung meiner Diplomarbeit beigetragen haben. Insbesondere möchte ich mich noch bei Mag. Dr. Bernd Neumayr für sein Engagement und seine Geduld bedanken.

Ein großer Dank gilt auch meiner Familie insbesondere meiner Mutter, die mich stets moralisch unterstützt hat und mir überhaupt erst das Studium ermöglichte.

Des Weiteren möchte ich mich noch bei meinen Geschwistern und Freunden bedanken, mit denen ich viele schöne Stunden verbracht habe und so abschalten konnte, um die nötige Kraft für das Studium aufzubringen.

Mein letzter Dank gilt meinen Onkel Wilhelm Mayrhofer, der viel auf unserem landwirtschaftlichen Betrieb gearbeitet hat, um mich zu entlasten, so dass ich Zeit für mein Studium aufbringen konnte.

Kurzfassung

Mit Hilfe der Multi-Level Modellierung können Objekte der realen Welt auf verschiedenen Abstraktionsebenen dargestellt werden. Um solche Modelle erstellen zu können, wurden in den letzten Jahren Techniken wie *materialization*, *powertypes* und *deep instantiation* verwendet. Aufbauend auf diese genannten Techniken wurden am Institut für Data & Knowledge Engineering an der Johannes Kepler Universität Linz Multi-Level-Objects (M-Objects) und Multi-Level-Relationships (M-Relationships) eingeführt, die einen neuen Ansatz zur Modellierung von Multi-Level Objekten und deren Beziehungen darstellen.

Kern dieser Arbeit ist die Umsetzung von M-Objects und M-Relationships in Protégé-Frames, einer Open-Source-Plattform, die Werkzeuge zur Erstellung von Modellen bereitstellt. Für diese Umsetzung wird ein Mapping erstellt, das beschreibt, wie M-Objects und M-Relationships in Protégé realisiert werden können. Konkret definiert das Mapping, durch welche Elemente von Protégé-Frames M-Objects, M-Relationships, die verschiedenen Abstraktionsebenen, Attribute, etc. dargestellt werden können. Dieses Mapping bildet die Grundlage für die Erweiterung von Protégé und ist somit ein zentraler Bestandteil dieser Arbeit.

Aufbauend auf das Mapping werden die Vorgehensweise bei der Implementierung und die Implementierung an sich beschrieben, wobei die Umsetzung von M-Objects und M-Relationships in Protégé-Frames durch ein Plug-In realisiert wurde. Im Anschluss daran wird das Plug-In selbst, dessen Darstellung, die Navigation und die verfügbaren Operationen genauer betrachtet. Den Abschluss bildet das Benutzerhandbuch, in dem anhand eines Beispiels der Umgang mit dem Plug-In näher erläutert wird.

Abstract

Using multi-level modeling real world objects can be viewed at different levels of abstraction. To create these models, techniques such as *materialization*, *power types* and *deep instantiation* have been used in recent years. Building on these techniques multi-level objects (m-objects) and multi-level relationships (m-relationships) were introduced at the Johannes Kepler University Linz, Department of Business Informatics - Data & Knowledge Engineering, as a new approach for modeling multi-level objects and their relationships.

The aim of this thesis is the implementation of m-objects and m-relationships in Protégé-Frames, an open source platform, which contains tools for creating models. For this implementation it is necessary to create a mapping which describes how m-objects and m-relationships can be integrated in Protégé. More specifically the mapping defines how to represent m-objects, m-relationships, the different levels of abstraction, attributes and so on with the modelling elements of Protégé-Frames. This mapping is the basis for the extension of Protégé and is thus a central part of this work.

Based on the mapping, this thesis describes the implementation of m-objects and m-relationships in Protégé-Frames as well as the underlying design principles of the plug-in. Furthermore, the plug-in itself, its presentation, the navigation and the available operations will be considered in more detail. At the end of this thesis is a user manual with an example which illustrates the handling of the plug-in.

Inhaltsverzeichnis

1. Einleitung 13
1.1 Aufgabenstellung und Zielsetzung 13
1.2 Durchgängiges Beispiel in Anlehnung an [NGS09] 13
1.3 Aufbau der Arbeit 15
Teil A – Grundlagen 17
2. Multi-Level Modellierung mit M-Objects und M-Relationships 18
2.1 Grundlagen M-Objects 19
2.2 Grundlagen M-Relationships 21
2.3 M-Objects und M-Relationships am konkreten Beispiel 23
2.4 Zusammenfassung M-Objects und M-Relationships 26
3. Grundlagen Protégé-Frames 27
3.1 Was ist Protégé-Frames? 27
3.2 Knowledge-Model von Protégé 28
3.2.1 Klassen und Instanzen 28
3.2.2 Slots 28
3.2.3 Facets 29
3.2.4 Darstellung in Protégé-Frames 30
Teil B – Konzept der Umsetzung von M-Objects und M-Relationships in Protégé 31
4. Strukturelles Mapping 32
4.1 Mapping Grundlagen 32
4.2 Strukturelles Mapping der M-Objects 34
4.2.1 Metaklasse: MObjectClass 35
4.2.2 Klasse: MObject 36
4.2.3 Klasse: MLevel 37
4.2.4 Beispiel Mapping M-Objects 37
4.3 Strukturelles Mapping der M-Relationships 42
4.3.1 Klasse: MRelationship 43
4.3.2 Klasse: MConnectionLevel 45
4.3.3 Beispiel Mapping M-Relationships 46
5. Operationales Mapping 48
5.1 Operationen auf M-Objects 48
5.1.1 Erstellen von M-Objects 49

5.1.2 Löschen von M-Objects ... 51
5.1.3 Umbenennen von M-Objects ... 53
5.1.4 Verändern der Konkretisierungshierarchie eines M-Objects ... 54
5.1.5 Erstellen eines Levels ... 55
5.1.6 Löschen eines Levels ... 61
5.1.7 Umbenennen eines Levels ... 62
5.1.8 Erstellen, Ändern und Löschen eines Attributes ... 63
5.2 Operationen auf M-Relationships ... 65
5.2.1 Erstellen von M-Relationships ... 65
5.2.2 Löschen eines M-Relationships ... 68
5.2.3 Umbenennen eines M-Relationships ... 69
5.2.4 Erstellen, ändern und löschen eines ConnectionLevels ... 70
Teil C – Implementierung ... 72
6. Implementierung in Protégé-Frames ... 73
6.1 Vorbereitungen für die Implementierung ... 73
6.2 M-Object Browser ... 75
6.3 M-Object Editor ... 79
6.4 M-Relationship Browser ... 86
6.5 M-Relationship Editor ... 90
6.6 Restriktionen der Implementierung ... 93
7. Benutzerhandbuch ... 96
7.1 Erstellen von Objekten im M-Object Browser und M-Object Editor ... 97
7.2 Erstellen von Objekten im M-Relationship Browser und M-Relationship Editor ... 109
7.3 Abschließende Bemerkungen ... 112
8. Zusammenfassung ... 114
Literaturverzeichnis ... 115
9. Anhang ... 117
A.1 Installationsanleitung ... 117
A.2 Verwendete Programmversionen ... 118
2.1 Eclipse SDK Version 4.2.1 ... 118
2.2 Protégé-Frames Version 3.5 ... 120
A.3 Ausgewählte Teile der Implementierung ... 121

Abkürzungen

bspw.	beispielsweise
bzw.	beziehungsweise
d.h.	das heißt
EPL	Eclipse Public License
etc.	et cetera
http	Hypertext Transfer Protocol
JAR	Java Archive
Javadoc	ist ein Software-Dokumentationswerkzeug
JDBC	Java Database Connectivity
lt.	Laut
M-Object	Multi-Level Object
M-Relationship	Multi-Level Relationship
SPARQL	SPARQL Protocol and RDF Query Language.
SWRL	Semantic Web Rule Language
OWL	Web Ontology Language
UML	Unified Modelling Language
usw.	und so weiter
vgl.	vergleiche
vs.	versus
WWW	World Wide Web
z.B.	zum Beispiel

Abbildungsverzeichnis

Abbildung 1: Durchgängiges Beispiel (1) nach [NGS09] 14
Abbildung 2: Durchgängiges Beispiel (2) nach [NGS09] 14
Abbildung 3: Durchgängiges Beispiel (3) nach [NGS09] 15
Abbildung 4: Darstellung von M-Objects [NGS09] 20
Abbildung 5: Darstellung von M-Relationships [NGS09] 22
Abbildung 6: Beispiel M-Objects [NGS09] 23
Abbildung 7: Beispiel M-Relationships [NGS09] 25
Abbildung 8: Metamodell M-Object 34
Abbildung 9: Darstellung der Umsetzung von M-Objects in Protégé (1) 38
Abbildung 10: Darstellung der Umsetzung von M-Objects in Protégé (2) 40
Abbildung 11: Darstellung der Umsetzung von M-Objects in Protégé (3) 41
Abbildung 12: Darstellung der Umsetzung von M-Objects in Protégé (4) 42
Abbildung 13: Metamodel M-Relationship 43
Abbildung 14: Darstellung von M-Relationships in Protégé 46
Abbildung 15: Erstellen von M-Objects (1) 49
Abbildung 16: Erstellen von M-Objects (2) 50
Abbildung 17: Löschen von M-Objects 52
Abbildung 18: Umbenennen von M-Objects 53
Abbildung 19: Verändern der Konkretisierungshierarchie eines M-Objects 54
Abbildung 20: Erstellen eines Levels (1) 56
Abbildung 21: Erstellen eines Levels (2) 57
Abbildung 22: Erstellen eines Levels (3) 59
Abbildung 23: Löschen eines Levels 61
Abbildung 24: Umbenennen eines Levels 63
Abbildung 25: Erstellen, Ändern und Löschen von Attributen 64
Abbildung 26: Erstellen von M-Relationships (1) 66
Abbildung 27: Erstellen von M-Relationships (2) 67
Abbildung 28: Löschen eines M-Relationships 68
Abbildung 29: Umbenennen von M-Relationships 70
Abbildung 30: Darstellung von M-Objects und M-Relationships in Protégé 75
Abbildung 31: M-Object Browser 76
Abbildung 32: Kontextmenü im M-Object Browser 77

Abbildung 33: Dialog zum Erstellen von M-Objects 78
Abbildung 34: Dialog zum Erstellen eines M-Objects und eines Levels 78
Abbildung 35: Dialog für die Auswahl von übergeordneten M-Objects 79
Abbildung 36: M-Object Editor 79
Abbildung 37: Kontextmenü für Attribute im M-Object Editor 81
Abbildung 38: Dialog zur Eingabe eines neuen Attributes 82
Abbildung 39: Dialog zum Ändern eines Attributes 83
Abbildung 40: Dialog zum Erstellen eines neuen Levels 84
Abbildung 41: Auswirkung des Erstellens eines neuen Levels auf den M-Object Browser 85
Abbildung 42: Dialog zum Löschen eines Levels 85
Abbildung 43: M-Relationship Browser 86
Abbildung 44: Kontextmenü im M-Relationship Browser 88
Abbildung 45: Dialog zum Erstellen eines M-Relationships 89
Abbildung 46: Dialog zum Löschen eines M-Relationships 90
Abbildung 47: M-Relationship Editor 90
Abbildung 48: Kontextmenü für ConnectionLevels im M-Relationship Editor 92
Abbildung 49: Dialog zum Erstellen eines neuen ConnectionLevels 92
Abbildung 50: Schritt 1 – Erstellen von *Produkt*, *Katalog*, *Kategorie*, *Modell*, *PhysischesObjekt* 98
Abbildung 51: Schritt 2 – Erstellen von *bezeichner*, *steuersatz*, *listenpreis*, *serienNr* 99
Abbildung 52: Schritt 3 – Erstellen von *Auto*, *maxGeschwindigkeit*, *kilometerstand* 100
Abbildung 53: Schritt 4 – Erstellen von *Buch*, *autor* 101
Abbildung 54: Schritt 5 – Erstellen von *Marke*, *markteinführung* 103
Abbildung 55: Schritt 6 – Erstellen von *Porsche911*, *HarryPotter1*, *porsche911Club* 104
Abbildung 56: Schritt 7 – Erstellen von *Porsche911CarreraS*, *Porsche911GT3* 106
Abbildung 57: Schritt 8 – Erstellen von *meinPorsche911CarreraS*, *meinHarryPotter1* 107
Abbildung 58: Schritt 9 – Erstellen von *Konzern*, *Root*, *Industriesektor*, *Unternehmen*, *Fabrik* 108
Abbildung 59: Erstellen von *Autohersteller*, *PorscheLtd*, *PorscheZuffenhausen* 109
Abbildung 60: *Erstellen von Produkt-produziertVon-Konzern* 110
Abbildung 61: Erstellen von *Kategorie-Industriesektor*, *Modell-Unternehmen*, *PhysischesObjekt-Fabrik* 111
Abbildung 62: Erstellen von *Auto-produziertVon-Autohersteller* 112
Abbildung 63: Modell einer Produkthierarchie in Protégé 113

Tabellenverzeichnis

Tabelle 1: Attributwerte beim Anlegen von M-Objects (1) 49
Tabelle 2: Attributwerte beim Anlegen von M-Objects (2) 51
Tabelle 3: Löschen von M-Objects 52
Tabelle 4: Attributwerte beim Anlegen eines Levels (1) 56
Tabelle 5: Attributwerte beim Anlegen eines Levels (2) 58
Tabelle 6: Attributwerte beim Anlegen eines Levels (3) 60
Tabelle 7: Löschen eines Levels 62
Tabelle 8: Javadoc - Methode *createMObject* 122
Tabelle 9: Javadoc - Methode *createMObjectAndLevel* 122
Tabelle 10: Javadoc - Methode *deleteMObject* 122
Tabelle 11: Javadoc - Methode rename*MObject* 123
Tabelle 12: Javadoc - Methode *createMRelationship* 123
Tabelle 13: Javadoc - Methode *deleteMRelationship* 123
Tabelle 14: Javadoc - Methode *renameMRelationship* 124
Tabelle 15: Javadoc - Methode *createLevel* 124
Tabelle 16: Javadoc - Methode *deleteLevel* 125
Tabelle 17: Javadoc - Methode *renameLevel* 125
Tabelle 18: Javadoc - Methode *createConnectionLevel* 125
Tabelle 19: Javadoc - Methode *deleteConnectionLevel* 126
Tabelle 20: Javadoc - Methode *createOwnAttribute* 126
Tabelle 21: Javadoc - Methode createTemplateAttribute 126
Tabelle 22: Javadoc - Methode *deleteAttribute* 127
Tabelle 23: Javadoc - Methode *changeAttribute* 127
Tabelle 24: Javadoc - Methode *changeParentMObject* 128
Tabelle 25: Javadoc - Methode *createImportantClasses* 128

1. Einleitung

1.1 Aufgabenstellung und Zielsetzung

Aufgabenstellung bzw. Kern dieser Arbeit ist die Entwicklung eines Konzeptes zur Umsetzung des M-Object- und M-Relationship-Ansatzes in Protégé und weiterführend die Umsetzung dieses Konzeptes in eine Anwendung, die durch ein Plug-In in Protégé realisiert wurde. Dieses Plug-In soll auf möglichst einfache und übersichtliche Weise sämtliche Aspekte des M-Object- und M-Relationship-Ansatzes und deren Beziehungen untereinander veranschaulichen und dem Benutzer die Möglichkeit geben, Modelle im Sinne des M-Object- und M-Relationship-Ansatzes zu erstellen, zu ändern und zu löschen.

Ziel dieser Arbeit ist die vollständige Umsetzung des M-Object- und M-Relationship-Ansatzes für Domain-Modellierung nach [NGS09] in Protégé und damit zu zeigen, dass sich dieser Ansatz als Anwendung realisieren lässt und somit einen Mehrwert für die Benutzer bringt, die dadurch komfortabel ein Modell mit Hilfe des M-Object- und M-Relationship-Ansatzes erstellen bzw. verwalten können.

1.2 Durchgängiges Beispiel in Anlehnung an [NGS09]

Für die Erklärung des M-Object bzw. M-Relationship Ansatzes und die Umsetzung in Protégé wird zur besseren Veranschaulichung ein Beispiel verwendet. Dieses soll vor allem der einfacheren Darstellung und dem besseren Verständnis der Umsetzung in Protégé dienen.

Beispiel:

Ein Online-Store kauft und verkauft verschiedene Produkte wie bspw. Bücher, Autos, etc. Diese Produkte werden auf verschiedenen Abstraktionsebenen beschrieben. In diesem Fall sind diese Ebenen Kategorie, Modell und PhysischesObjekt. Jedes Produkt besitzt auf der Ebene Kategorie ein Attribut Steuersatz, auf der Ebene Modell ein Attribut Listenpreis und auf der Ebene PhysischesObjekt ein Attribut Seriennummer.

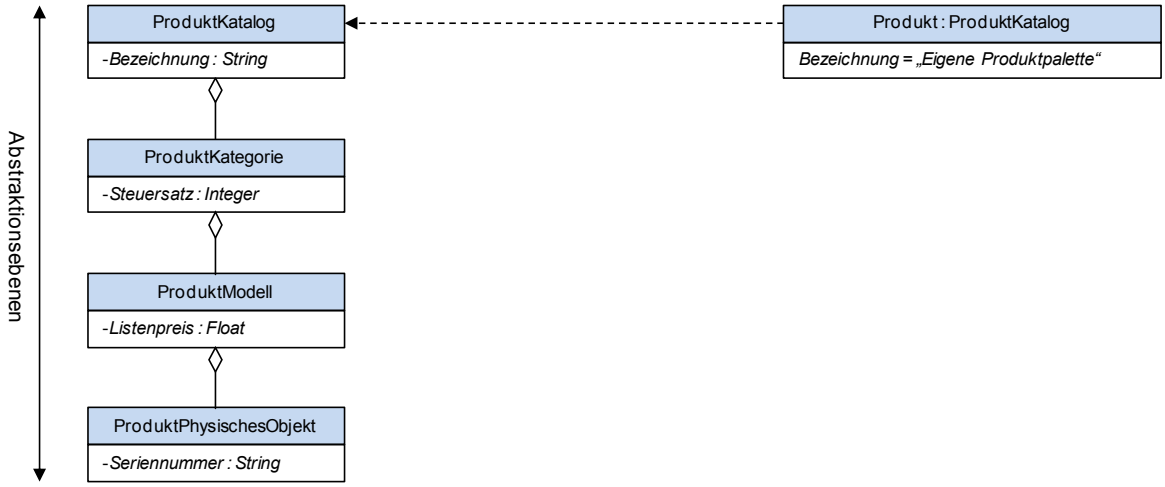

Abbildung 1: Durchgängiges Beispiel (1) nach [NGS09]

Bücher sind auf der Ebene Kategorie angesiedelt, wobei Bücher zusätzlich auf der Ebene Modell ein Attribut Autor besitzen.

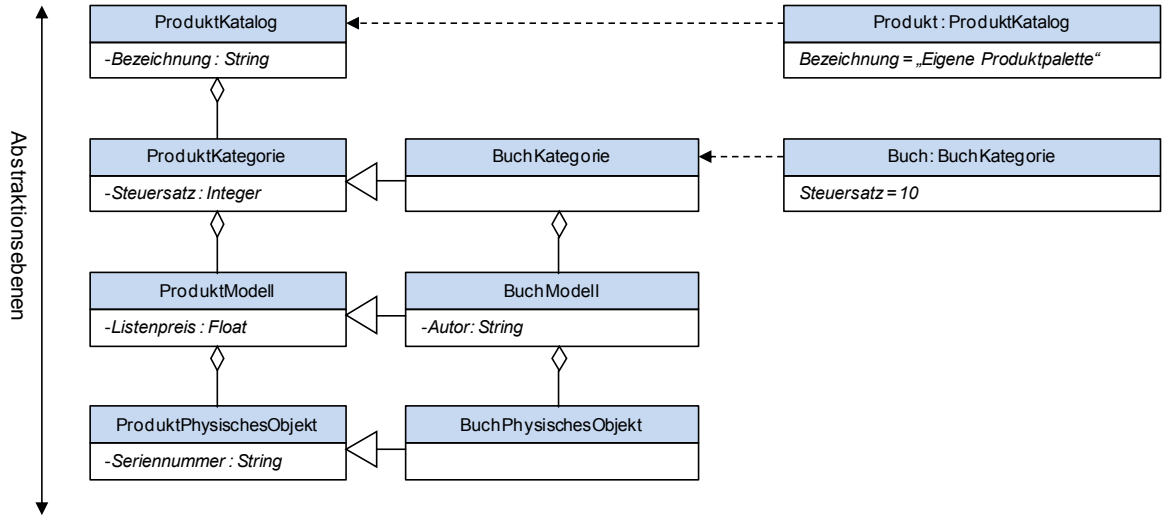

Abbildung 2: Durchgängiges Beispiel (2) nach [NGS09]

Autos liegen ebenfalls auf der Ebene Kategorie benötigen aber noch eine zusätzliche Ebene Marke, zu der wiederum Attribute wie Höchstgeschwindigkeit auf der Ebene Model und Kilometerstand auf der Ebene PhysischesObjekt eingeführt werden.

Weiters spezialisiert sich der Online-Store auf den Kauf und Verkauf einer bestimmten Automarke nämlich den Porsche911, wobei gespeichert werden soll, ob ein konkretes Auto

Mitglied im Posche911Club ist. Hierzu wird auf der Ebene PhysischesObjekt das Attribut Porsche911Club eingeführt.

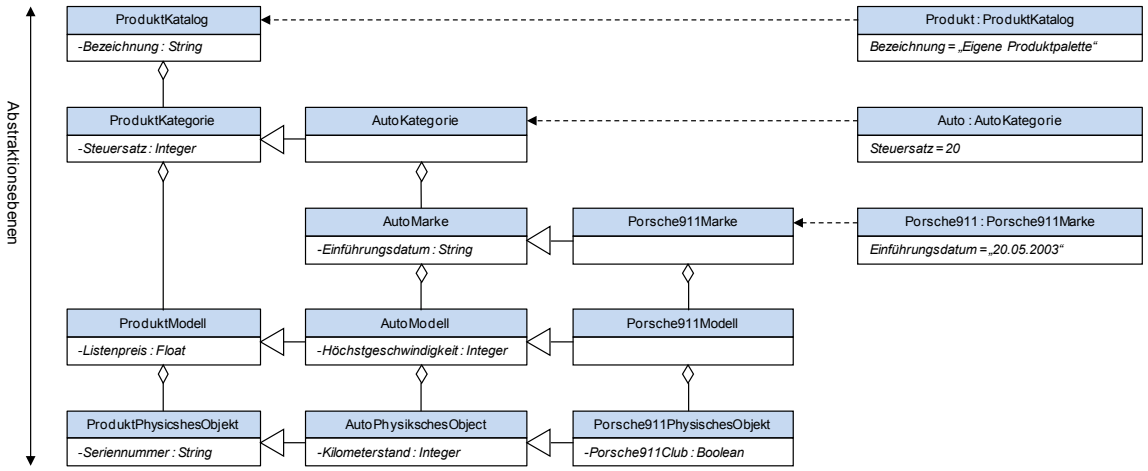

Abbildung 3: Durchgängiges Beispiel (3) nach [NGS09]

Für die Logistik und das Qualitätsmanagement ist eine enge Verknüpfung mit den Zulieferfirmen notwendig. Diese Firmen werden ebenfalls auf den verschiedenen Abstraktionsebenen Industriesektor, Unternehmen und Fabrik dargestellt. So gehören bspw. Produzenten von Autos einem speziellen Industriesektor an, in diesem Fall der Autoindustrie. Um die Qualität zu verbessern und Qualitätsproblemen vorzubeugen soll jedes physische Objekt der Kategorie Auto in Beziehung zur Fabrik stehen, in der es tatsächlich produziert wurde. Natürlich muss dann auch diese Fabrik zu dem Unternehmen gehören, das dieses Automodell auch erzeugt.

1.3 Aufbau der Arbeit

Die Arbeit ist in drei wesentliche Teile gegliedert, die im Folgenden kurz beschrieben werden.

Teil A - Grundlagen

Kapitel 2 und 3 beschreiben die Grundlagen der Multi-Level Modellierung sowie die Grundlagen von Protégé. Die Grundlagen dienen im Wesentlichen zur Vertiefung in die

Materie der Multi-Level Modellierung und tragen in weiterer Folge zum besseren Verständnis der Arbeit bei. Dazu wird zunächst genauer auf die Multi-Level Modellierung an sich eingegangen. In weiterer Folge werden die Kernthemen M-Objects und M-Relationships sowie die Entwicklungsumgebung Protégé betrachtet, die als Basis für die weiterführende Arbeit unabdingbar sind.

Teil B - Konzept der Umsetzung von M-Objects und M-Relationships in Protégé

In Kapitel 4 wird das strukturelle Mapping von M-Objects und M-Relationships beschrieben. Nach den allgemeinen Grundlagen zum Mapping wird in diesem Kapitel genau erläutert, durch welche in Protégé vorhandenen Konstrukte (Class, Instance, Slot, usw.) M-Objects, M-Relationships und deren Verknüpfungen untereinander dargestellt werden können. Schlussendlich ergibt sich daraus ein Schema, das das Konzept der M-Objects und M-Relationships vollständig in Protégé beschreibt.

Kapitel 5, das Operationale Mapping, beschäftigt sich wie der Name schon sagt mit den Operationen, die auf M-Objects und M-Relationships durchgeführt werden können. Es wird beschrieben, welche Veränderungen die verschiedenen Operationen auf M-Objects und M-Relationships in Protégé hervorrufen.

Teil C – Implementierung

Kapitel 6 befasst sich mit der tatsächlichen Implementierung von M-Objects und M-Relationships in Protégé. Es wird beschrieben wie bei der Entwicklung der Umsetzung vorgegangen wurde. Unter anderem werden in diesen Kapiteln auch die Probleme bei der Implementierung sowie die Einschränkungen des Plug-Ins in Protégé für den Benutzer veranschaulicht. Schlussendlich wird die graphische Darstellung des Plug-In beschrieben, in der näher auf die einzelnen grafischen Elemente, die Navigation und die Operationen eingegangen wird.

In Kapitel 7 wird durch ein Benutzerhandbuch das Arbeiten mit dem Plug-In beschrieben. Es wird die Erstellung eines Modells mittels des Plug-Ins in Bezug auf das vorhin erwähnte durchgängige Beispiel Schritt für Schritt genau erklärt und so dem Benutzer der Umgang damit näher gebracht.

Teil A – Grundlagen

2. Multi-Level Modellierung mit M-Objects und M-Relationships 18

3. Grundlagen Protégé-FramesGrundlagen 27

2. Multi-Level Modellierung mit M-Objects und M-Relationships

Mithilfe der Multi-Level Modellierung können Objekte auf verschiedenen Abstraktionsebenen durch Klassifizierung, Aggregation und Generalisation dargestellt werden. Durch diese Konkretisierungen der Objekte erhöhen sich aber oftmals die Komplexität und die Wartung solcher Modelle. [NGS09] [NST08]

Objekte werden im Alltag oft in Hierarchien auf verschiedenen Ebenen dargestellt. Die Modellierung solcher Hierarchien ist noch vergleichsweise einfach, solange Objekte auf jeder Hierarchieebene einheitlich gestaltet sind. Allerdings kommt es vor, dass vorhandene Informationssysteme wachsen oder in andere integriert werden, wobei dann nicht ausgeschlossen werden kann, dass Objekte auf gleicher Abstraktionsebene in verschiedenen Subhierarchien voneinander abweichen. In weiterer Folge können sich auch Subhierarchien zweier Objekte auf derselben Abstraktionsebene unterscheiden. [NGS09]

Verschiedene Techniken zur Vermeidung der Komplexität in der Darstellung von Modellen auf mehreren Abstraktionshierarchien wurden in den letzten Jahren verwendet wie bspw. *materialization*, *powertypes* und *deep instantiation* (siehe [NST08], [DPZ02], [NJSS14]). All diese Techniken vereinfachen die Modellierung auf verschiedenen Abstraktionshierarchien und unterstützen die tiefe Charakterisierung. Allerdings haben sie den Nachteil, dass keine der genannten Techniken die Modelle unterstützt, bei denen die Subhierarchien der einzelnen Objekte voneinander abweichen. Multi-Level Objects und Multi-Level Relationships bauen auf die Ergebnisse der vorher genannten Techniken auf und unterstützen eine einfache, natürliche Darstellung der Konkretisierung von Objekten und deren Beziehungen auf verschiedenen Abstraktionsebenen. Damit soll vor allem die Lesbarkeit verbessert, die Wartung erleichtert und die Komplexität verringert werden. [NGS09] [Sch10]

Der M-Object- und M-Relationship-Ansatz wurde am Institut für Data & Knowledge Engineering an der Johannes Kepler Universität ins Leben gerufen und stellt ein Kernthema

dieser Arbeit dar. Aus diesem Grund wird im Folgenden auf die Grundlagen der M-Objects und M-Relationships eingegangen und anhand des in Kapitel 1.2 beschrieben durchgängigen Beispiels näher erläutert.

2.1 Grundlagen M-Objects

Die Modellierung von Objekten auf unterschiedlichen Abstraktionsebenen hat in den letzten Jahren in verschiedenen Bereichen immer mehr an Bedeutung gewonnen. Objekte werden dabei oft in Hierarchien organisiert, welche aus mehreren Ebenen sogenannten Levels bestehen. Beispiele für solche Hierarchien sind Produkthierarchien, Dimensionshierarchien, Taxonomien im Allgemeinen, usw. [NGS09]

M-Objects beschreiben Objekte der realen Welt auf verschiedenen Abstraktionsebenen. Ein konkretes Beispiel wäre, wie erwähnt, eine Produkthierarchie. Ein Produktkatalog könnte drei verschiedene Abstraktionsebenen beinhalten nämlich *Kategorie*, *Modell* und *PhysischesObjekt*. Instanzen dieser Ebenen wären bspw. *Auto*, *Porsche911CarreraS*, *meinPorsche911CarreraS* oder *Buch*, *HarryPotter1*, *meinHarryPotter1*. [NGS09] [Sch10]

Besitzen diese Objekte eine einheitliche Struktur bzw. sind auf jeder Hierarchieebene gleich aufgebaut, ist die Modellierung solcher Konzepte noch relativ einfach. Allerdings können Objekte, die auf dem gleichen Level angesiedelt sind, sich in deren Hierarchien unterscheiden, was die Modellierung wesentlich komplexer macht. Nimmt man die beiden Objekte *Auto* und *Buch* her, so besitzen diese auf der Ebene *Kategorie* ein gemeinsames Attribut *steuersatz*, das für beide Produkte Gültigkeit hat. Auf der Ebene *Modell* hingegen können sich diese Produkte deutlich unterscheiden. Das Objekt *Buch* besitzt auf dieser Hierarchieebene das Attribut *autor*, wohingegen das Objekt *Auto* ein neues Attribut *maxGeschwindigkeit* einführt. Somit unterscheiden sich diese beiden Objekte in ihren Eigenschaften. In weiterer Folge können sich Objekte auf gleicher Hierarchieebene auch dadurch unterscheiden, indem in der Subhierarchie ein weiterer Level eingefügt wird, der das Objekt genauer konkretisiert. Im konkreten Anwendungsfall wird für das Objekt *Auto* ein neuer Level *Marke* und ein neues Objekt *Porsche911* eingeführt, das das Objekt *Auto*

konkretisiert. Der Level *Marke* taucht allerdings nur in der Subhierarchie von *Auto* auf. Die Subhierarchie von *Buch* bleibt unberührt. [NGS09]

In UML können solche Hierarchien durch eine Kombination von Aggregation, Generalisation und Klassifikation dargestellt werden. Werden neue Konzepte eingeführt, erhöht sich allerdings die Komplexität enorm und führt dazu, dass sich die Darstellung solcher Modelle nur schwer umsetzen lässt. Für jedes Konzept, das neu angelegt wird, müssen eine neue Instanz und eine Vielzahl neuer Klassen in Verbindung mit Aggregation, Generalisation und Instanzierungsbeziehungen angelegt werden. Die daraus sich ergebenden Redundanzen und Fragmentierung führen dazu, dass sich die Wartung und Erweiterung solcher Modelle als schwierig gestaltet. [NGS09]

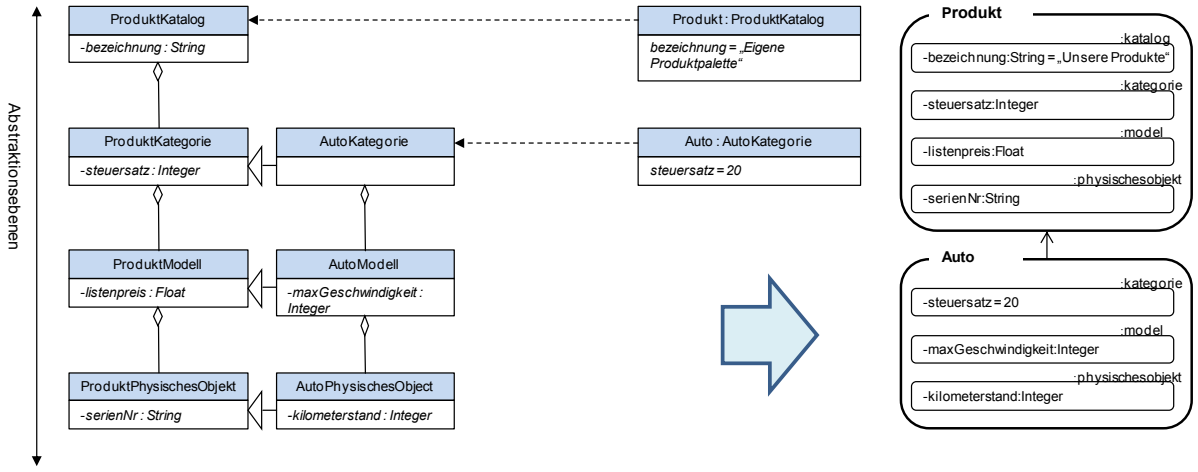

Abbildung 4: Darstellung von M-Objects [NGS09]

Multi-Level Objekte, kurz M-Objects, kapseln mehrere Abstraktionsebenen, sogenannte Levels, und sind in Abbildung 4 auf der rechten Seite dargestellt. Diese Hierarchien besitzen eine Ordnung von der abstraktesten Ebene bis hin zur konkretesten Ebene. Das M-Object *Produkt* bspw. beinhaltet die Level *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt*.
Eigenschaften für das M-Object selbst können festgelegt werden, indem auf dem Top-Level, bei dem das M-Object angelegt wurde, Werte gesetzt werden. Bei dem M-Object *Produkt* auf der Ebene *Katalog* wird ein Attribut *beschreibung* eingeführt und mit dem Wert *„Unsere Produkte“* belegt.

M-Objects können aber auch Attribute auf Subebenen beschreiben. Das M-Object *Produkt* führt bspw. auf der Ebene *Kategorie* ein Attribut s*teuersatz* ein. Wird nun ein M-Object durch ein anderes M-Object konkretisiert, so übernimmt das untergeordnete M-Object alle Level mit Ausnahme des Top-Levels des übergeordneten M-Objects. Damit verbunden erhält das neue M-Object auch sämtliche Attribute, die beim übergeordneten M-Object für die zweite Ebene angelegt wurden. Im konkreten Beispiel erbt das M-Object *Auto* das Attribut *steuersatz* vom M-Object *Produkt,* das beim *Produkt* auf der zweiten Ebene *Kategorie* angelegt wurde. Der Wertebereich dieses vererbten Attributes ist allerdings bereits vom übergeordneten M-Object *Produkt* fest vorgegeben.

Weiters werden für jede der vererbten Ebenen des untergeordneten M-Objects neue Klassen angelegt und diese stellen Subklassen des entsprechenden Levels des übergeordneten M-Objects dar. Die Klasse *AutoKategorie* ist bspw. eine Subklasse von *ProduktKategorie*, die Klasse *AutoModell* ist eine Subklasse von *ProduktModell*, usw. [NGS09]

2.2 Grundlagen M-Relationships

M-Relationships beschreiben Beziehungen zwischen M-Objects auf verschiedenen Ebenen und können dabei unterschiedliche Rollen einnehmen, je nachdem auf welcher Ebene die M-Objects in Verbindung stehen. Es stehen also nicht nur direkt die M-Objects in Beziehung zueinander, sondern es besteht auch eine Abhängigkeit zwischen den einzelnen Levels der verschiedenen M-Objects. Wie in Abbildung 5 dargestellt, verbindet demnach das M-Relationship *Produkt-produziertVon-Konzern* nicht nur die beiden M-Objects *Produkt* und *Konzern*, sondern stellt auch die Beziehungen zwischen den Abstraktionsebenen der beiden M-Objects her. Konkret werden hierzu die Beziehungen der Level *Kategorie* und *Industriesektor*, *Modell* und *Unternehmen* sowie *PhysischesObjekt* und *Fabrik* angeführt. [NGS09]

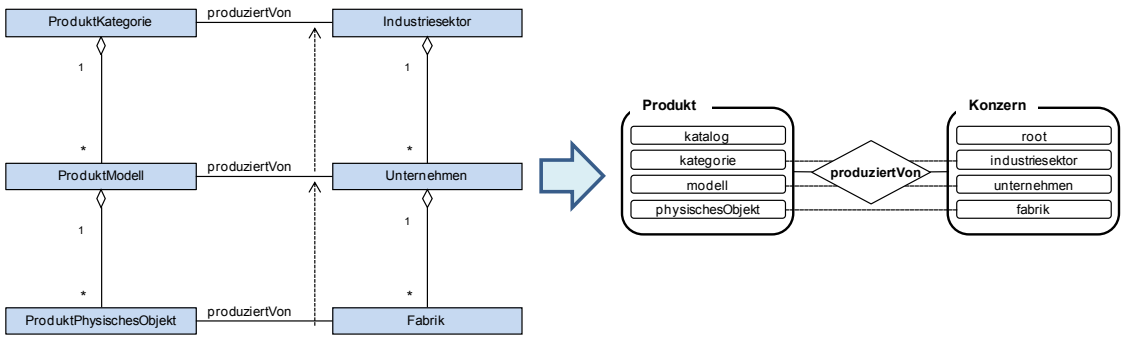

Abbildung 5: Darstellung von M-Relationships [NGS09]

Wie erwähnt können M-Relationships verschiedene Rollen einnehmen, die im Folgenden kurz angeführt werden.

M-Relationship Rollen

- **Relationship class:** beschreibt ein M-Relationship, das ein M-Object mit einem anderen M-Object verbindet, wobei die Verbindung der beiden nicht auf der höchsten Abstraktionsebene beruht.
- **Relationship occurrence***:* beschreibt ein M-Relationship, das zwei M-Objects auf der höchsten Abstraktionsebene miteinander in Beziehung stellt.
- **Shared relationship:** beschreibt ein M-Relationship, bei dem ein M-Objekt auf einer untergeordneten Abstraktionsebene mit einem M-Object auf der höchsten Abstraktionsebene, also auf dem Top-Level, in Verbindung steht. [NeSc09]

Je nachdem welche Rolle ein M-Relationship einnimmt, bestimmt welche Ebenen der Abstraktionshierarchien miteinander in Beziehung stehen. Um die Konsistenz zu wahren, dürfen dabei auch nur solche Abstraktionsebenen in Beziehung gestellt werden, die sowohl im Source M-Object als auch im Target M-Object vorhanden sind. Zusätzlich muss die relative Ordnung von Source–Levels und Target–Levels einheitlich sein. Damit ist gemeint, dass sich Verbindungen der Beziehungen zwischen den Abstraktionsebenen zweier M-Objects nicht kreuzen dürfen. Dies wäre der Fall, wenn in Abbildung 5 das M-Object *Produkt* auf der Ebene *Modell* mit dem M-Object *Konzern* auf der Ebene *Industriesektor* und wiederum das M-Object *Produkt* auf der Ebene *Kategorie* mit dem M-Object *Konzern* auf der Ebene *Unternehmen* in Beziehung stehen würde. [NGS09]

2.3 M-Objects und M-Relationships am konkreten Beispiel

Im Folgenden werden M-Objects und M-Relationships anhand eines konkreten Beispiels beschrieben, das bereits in Kapitel 1.2 näher erläutert wurde. Es soll helfen den M-Object- und M-Relationship-Ansatz besser zu verstehen und einen tieferen Einblick in diese Modellierungsmethode geben.

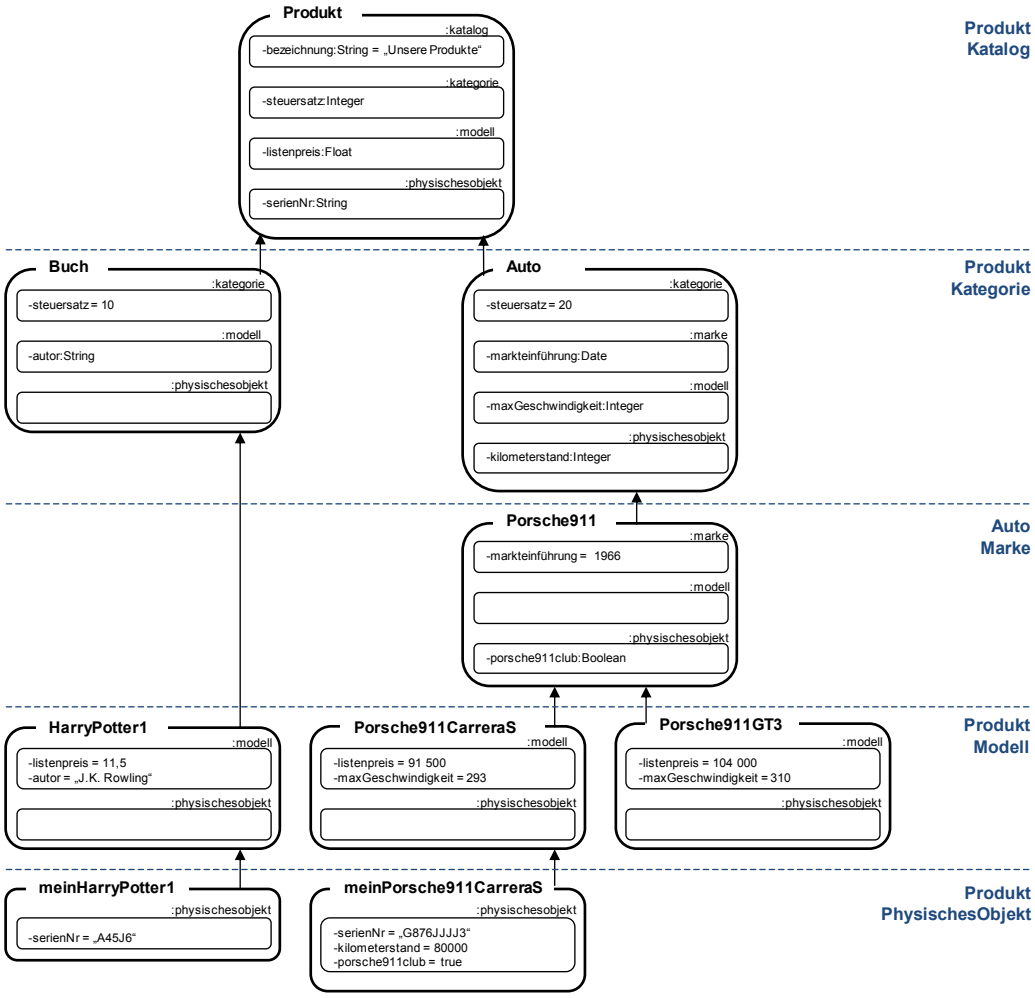

Abbildung 6: Beispiel M-Objects [NGS09]

Die Abbildung 6 beschreibt eine Produkthierarchie wie sie auch in der Praxis vorkommen könnte. Das M-Object *Produkt* besitzt vier Abstraktionsebenen *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt*, wobei es selbst auf der obersten Ebene *Katalog* angesiedelt ist. Weiters werden für jede dieser Abstraktionsebenen Attribute eingeführt. Dies sind auf der Ebene *Katalog* das Attribut *bezeichnung*, auf der Ebene *Kategorie* das Attribut *steuersatz*,

auf der Ebene *Modell* das Attribut *listenpreis* und auf der Ebene *PhysischesObjekt* das Attribut *serienNr*. Das M-Objekt *Produkt*, das auf der obersten Ebene *Kategorie* angesiedelt ist, besitzt für das Attribut *bezeichnung* den Wert *„Unsere Produkte“.* [NeSc09]

In dem Beispiel wird das M-Object *Produkt* konkretisiert durch die M-Objects *Auto* und *Buch*. Diese beiden M-Objects erben somit alle Abstraktionsebenen von *Produkt*, abgesehen von der obersten Ebene *Katalog*. Sie sind sozusagen Instanzen der Ebene *Kategorie* des M-Objekts *Produkt*. Beide M-Objects besitzen das Attribut *steuersatz* auf der Ebene *Kategorie,* allerdings unterscheiden sie sich in deren Wert. Zusätzlich führt das M-Object *Buch* auf der Ebene *Modell* ein neues Attribut *autor* ein, wohingegen das M-Object *Auto* auf demselben Level das Attribut *maxGeschwindigkeit* und auf dem Level *PhysischesObjekt* ein Attribut *kilometerstand* aufweist. Weiters spezialisiert das M-Objekt *Auto* das M-Objekt *Produkt*, indem es einen neuen Level *Marke* und gleichzeitig ein neues Attribut für diesen Level nämlich *markteinführung* einführt. [NeSc09]

Das M-Object *Porsche911* stellt eine Konkretisierung von dem M-Object *Auto* dar und erbt wiederrum alle Abstraktionsebenen mit Ausnahme des Top-Levels von M-Object *Auto*. Auf der Ebene *PhysischesObjekt* wird ein Attribut *porsche911club* eingeführt. Da das M-Object *Porsche911* eine Konkretisierung von *Auto* und *Auto* eine Konkretisierung von *Produkt* darstellt, konkretisiert *Porsche911* indirekt auch *Produkt*. Ausgehend davon konkretisiert das M-Object *meinPorsche911CarreraS* direkt bzw. indirekt *Porsche911CarreraS*, *Porsche911*, *Auto* und *Produkt*. [NeSc09]

Wie in diesem Beispiel zu erkennen ist, bündeln und ordnen M-Objects die verschiedenen Abstraktionseben beginnend von der abstraktesten bis hin zur konkretesten. Dabei beschreiben sie sich selbst und die gemeinsamen Eigenschaften der Objekte auf den verschiedenen Konkretisierungsebenen. M-Objects vererben die Abstraktionsebenen an die direkt untergeordneten M-Objects weiter. Eine Ausnahme dabei bildet die oberste Ebene, die nicht vererbt wird. Sie beschreibt das M-Object durch Attribute selbst, was im Fall des M-Objects *Buch* das Attribut *steuersatz* mit dem Wert *10* darstellt. Alle weiteren Attribute der

anderen Abstraktionsebenen beschreiben gemeinsame Attribute von untergeordneten M-Objects. [NST10]

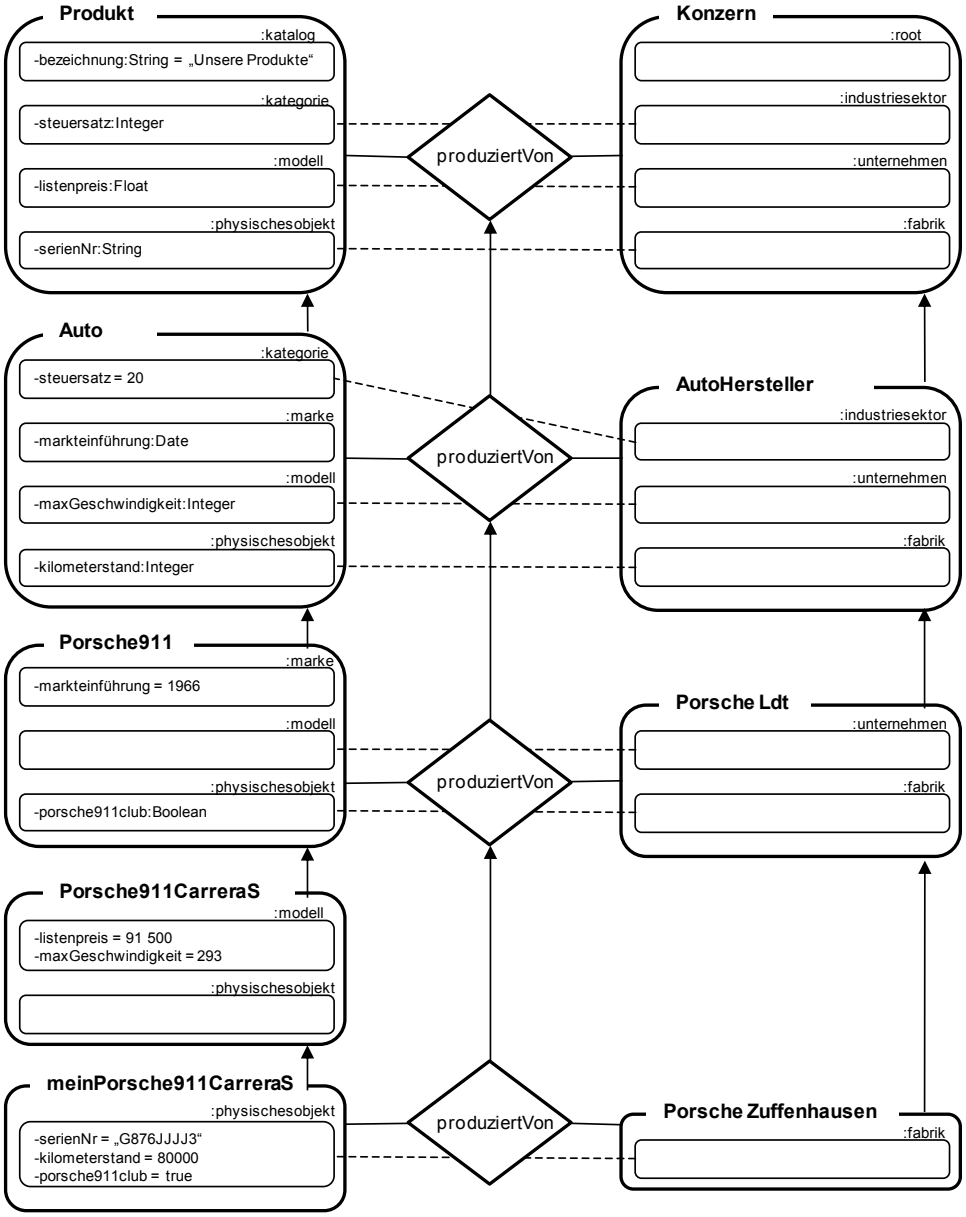

Abbildung 7: Beispiel M-Relationships [NGS09]

In Abbildung 7 werden typische M-Relationships dargestellt. Dabei beschreibt ein M-Object die Beziehungen auf den verschiedenen Abstraktionsebenen zu einem anderen M-Object und dessen Abstraktionsebenen. Das M-Relationship *Produkt-produziertVon-Konzern* stellt somit Beziehungen der Abstraktionsebenen *Kategorie* und *Industriesektor*, *Modell* und *Unternehmen* und *PhysischesObjekt* und *Fabrik* her. Betrachtet man nun das M-Relationship

Auto-produziertVon-Autohersteller, so stellet dies eine Konkretisierung des M-Relationships *Produkt-produziertVon-Konzern* dar und beschreibt, dass Autos nur von Autoherstellern produziert werden können. In weiterer Folge muss jedes Automodell von einem Unternehmen produziert werden, das zu dem Industriesektor Autohersteller gehört. Genauso dazu muss auch jedes konkrete Auto (z.B.: *meinPorsche911CarreraS*) von einer Fabrik produziert worden sein, die dem Industriesektor Autohersteller angehört. [NeSc09]

2.4 Zusammenfassung M-Objects und M-Relationships

Mittels M-Objects und M-Relationships ist es also möglich, komplexe Modelle, wie die als Beispiel angeführte Produkthierarchie, einfach und vollständig darzustellen. Dazu erfüllen M-Objects und M-Relationships folgende Kriterien, an der die Unterstützung für die Multi-Level Modellierung gemessen werden kann.

- **Verschiedene Abstraktionsebenen**: Objekte können auf unterschiedlichen Abstraktionsebenen beschrieben werden.
- **Erweiterbarkeit**: Es soll hinsichtlich der Erweiterbarkeit möglich sein, für Objekte unterschiedliche Subhierarchien darzustellen. Dies ist dann der Fall, wenn zusätzliche Ebenen, Attribute oder Beziehungen eingeführt werden.
- **Vermeidung von Fragmentierung und Redundanz**: Jede Information, die nur ein bestimmtes Objekt betrifft, soll auch nur lokal bei diesem Objekt gespeichert werden um Fragmentierung und Redundanzen zu vermeiden.
- **Beziehungen**: Es soll möglich sein, Beziehungen zwischen Objekten auf den verschiedenen Abstraktionsebenen darzustellen.
- **Abfragen**: Multi-Level Modelle sollen Abfragemöglichkeiten besitzen, sowie eine einfache Navigation zwischen den verschiedenen Abstraktionsebenen gewährleisten. [NGS09]

3. Grundlagen Protégé-Frames

Ein wesentlicher Teil dieser Arbeit bestand darin, ein Plug-In für Protégé zu entwickeln, weshalb in den nachfolgen Kapiteln ein kurzer Überblick über Protégé-Frames gegeben wird.

3.1 Was ist Protégé-Frames?

Protégé wurde entwickelt vom *Standford Center for Biomedical Informatics Research*. Es ist eine freie Open-Source Plattform, die ein Paket von Werkzeugen für die Erstellung von Modellen und wissensbasierten Anwendungen bereitstellt. Dabei unterstützt Protégé die Erstellung, Visualisierung und die Manipulation von Ontologien in verschiedenen Darstellungsformaten. Abgesehen davon sind Erweiterungen von Protégé durch die Plug-In Architektur und das *Java-based Application Programm Interface (API)* möglich. [Stan15b] [Stan15d]

Protégé stellt zur Modellierung zwei verschiedene Varianten zur Verfügung:

- **Protégé-Frames Editor:** Stellt eine Benutzerschnittstelle zur Verfügung, die den Benutzer beim Erstellen von frame-basierten Ontologien, deren Speicherung und Verwaltung unterstützt. [Stan15c]
- **Protégé-OWL Editor:** Ist eine Erweiterung von Protégé, die die Erstellung und Entwicklung von Ontologien mittels OWL ermöglicht. [Stan15f]

Die Umsetzung von M-Objects und M-Relationships in Protégé in dieser Arbeit basiert auf Protégé-Frames, weshalb im nachfolgenden nicht weiter auf Protégé-OWL eingegangen wird. Mehr zu Protégé-OWL wird in einer anderen Arbeit [Hub11] vorgestellt, wo M-Objects und M-Relationships in Protégé-OWL umgesetzt wurden. Aufgrund der Tatsache, dass diese Arbeit, wie bereits erwähnt, auf Protégé-Frames aufbaut, wird deshalb Protégé weitgehend als Synonym für Protégé-Frames verwendet.

3.2 Knowledge-Model von Protégé

Die Wissensbasis von Protégé baut auf Frames auf, wobei Frames die prinzipiellen Bausteine darstellen. Eine Ontologie in Protégé besteht aus Klassen, Slots, Facets und Axiomen. Klassen sind Konzepte in einer bestimmten Domäne, Slots beschreiben Eigenschaften von Klassen und Facets wiederum beschreiben Eigenschaften von Slots. Axiome spezifizieren zusätzliche Bedingungen.

3.2.1 Klassen und Instanzen

Klassen werden in Protégé in einer taxonomischen Hierarchie dargestellt. Das bedeutet, wenn die Klasse B eine Subklasse der Klasse A ist, dann ist jede Instanz von B auch eine Instanz von A. Ist die Klasse *Autor* eine Subklasse von der Klasse *Angestellten*, so ist auch jede Instanz der Klasse *Autor* eine Instanz der Klasse *Angestellten*. Solche Beziehungen der Klassen untereinander werden in Protégé immer als Baum abgebildet, wobei die Wurzel dieser Hierarchie immer die Klasse :THING darstellt. [NFM00]

Protégé unterstützt außerdem auch noch Mehrfachvererbung, was bedeutet, dass eine Klasse mehrere Superklassen haben kann. Ein Beispiel hierfür wäre, dass die Klasse *Redakteur* sowohl eine Subklasse von *Mitarbeiter* als auch von *Autor* sein kann, wenn der *Redakteur* sowohl *Angestellter* als auch *Autor* bei einer Zeitung ist. Weiters können in Protégé sowohl Instanzen als auch Klassen Instanzen einer Klasse sein. Solche Klassen deren Instanzen wiederum Klassen sind nennt man Metaklassen. [NFM00]

3.2.2 Slots

In Protégé beschreiben Slots Eigenschaften von Klassen und Instanzen, wie z.B. den Namen eines *Angestellten* oder den Namen eines *Autors*. Slots sind direkt vom Grundbaustein Frame abgeleitet und sind damit unabhängig von irgendeiner Klasse definiert. Wenn ein Slot zu einem Objekt in einem Modell angehängt wird, so beschreibt er die Eigenschaften dieses Objektes. Als Beispiel wird ein Slot *name* erstellt und zur Klasse *Zeitung* und zur Klasse *Autor* hinzugefügt, wo er dort den Namen der Zeitung (bspw. „Daily News“) und des Autors („Mark

Allen“) repräsentiert. Ein erstellter Slot kann somit für mehrere Klassen verwendet werden und muss nicht immer neu erstellt werden. [NFM00]
Protégé definiert zwei verschiedene Arten von Slots, wobei der Unterscheid in der Vererbung der Slots liegt. Diese werden als Template Slots und Own Slots bezeichnet. [NFM00]

Own Slots

Own Slots, die an eine Klasse oder Instanz angehängt werden, beschreiben deren Eigenschaften. Allerdings werden Own Slots nicht vererbt, weder an die Instanzen noch an die Subklassen. Als Beispiel für einen solchen Slot wäre der Slot *Synonym* zu nennen, der zu der Klasse *Autor* hinzugefügt wird. In Zusammenhang mit einer Zeitung würde der Wert für den Slot *Synonym* bei der Klasse *Autor* „Quelle“ sein. Allerdings würde dieser Wert für eine Instanz der Klasse *Autor* bspw. „Mark Allen“ keinen Sinn ergeben. [NFM00]

Template Slots

Template Slots können nur zu Klassen hinzugefügt werden, wobei die Subklassen, im Gegensatz zu den Own Slots, die Template Slots erben. Ein Template Slot einer Klasse wird zu einem Own Slot bei der Instanz dieser Klasse. Ein Beispiel für ein Template Slots wären die Slots *name* oder *einstellungsdatum* bei der Klasse *Herausgeber*. Instanzen dieser Klasse würden diese Slots erben und sie mit Werte belegen („Mark Allen“, „12-12-2008“). [NFM00]

3.2.3 Facets

Mittels Facets können Restriktionen für einen Slot gesetzt werden. Mit ihnen kann bspw. festgelegt werden, wie viele Werte ein Slot annehmen kann bzw. welchen Datentyp (Integer, String, etc.) die entsprechenden Werte aufweisen müssen. Für numerische Werte kann ein Wertebereich also ein Minimum und ein Maximum gesetzt werden. Facets erteilen nur Restriktionen den Slots in Zusammenhang mit einem konkreten Objekt. Zum Beispiel kann der Slot *gehalt* für die Klasse *Autor* den Minimumwert *15000* aufweisen. Wird der Slot *Gehalt* allerdings der Klasse *Herausgeber* hinzugefügt, kann der Minimumwert bspw. auf *20000* gesetzt werden. [NFM00]

3.2.4 Darstellung in Protégé-Frames

Die Darstellung in Protégé erfolgt auf Basis von Formularen sogenannten „knowledge-acquisition forms“. Sie erlauben die einfache Eingabe und Verifizierung von Informationen und unterstützen somit den Benutzer im bestmöglichen Umfang. Wenn ein Benutzer bspw. eine Klasse generiert und dieser Klasse Template Slots zuordnet, erstellt Protégé anschließend automatisch ein Formular, das es dem Benutzer ermöglicht Instanzen dieser Klasse anzulegen. Dabei bestimmt der Slot an sich, dessen Kardinalität und der Wertebereich wie das Formular gestaltet wird, indem das standardmäßige Erscheinungsbild und der Inhalt angepasst wird. Als Beispiel hierbei ist anzuführen, dass ein Slot der den Wertebereich *Boolean* hat, als Checkbox dargestellt wird, wohin gegen ein Wertebereich S*tring* ein einfaches Textfeld benötigt.

Aber nicht nur die verschiedenen Slots haben Auswirkungen auf das Formular. Auch der Benutzer hat Einfluss auf das Layout des Formulars und teilweise auch den Inhalt, der angezeigt werden soll. Er kann bspw. die für ihn wichtigeren Informationen weiter oben auf dem Formular positionieren oder eine andere Darstellung für die Eingabe von Werten der Slots wählen. [NFM00]

Teil B – Konzept der Umsetzung von M-Objects und M-Relationships in Protégé

4 Strukturelles Mapping 32

5 Operationales Mapping 48

4. Strukturelles Mapping

Ein wesentlicher Teil dieser Arbeit ist die Umsetzung von M-Objects und M-Relationships in Protégé. Um dies zu gewährleisten wird ein Konzept bzw. in weiterer Folge ein Modell benötigt, das beschreibt, wie M-Objects bzw. M-Relationships und im weiteren Sinne auch Levels, Attribute, etc. in Protégé dargestellt werden können.

Da es sich dabei um ein Kernstück dieser Arbeit handelt und es für die spätere Implementierung unabdingbar war, wurde das Mapping bereits in einer sehr frühen Phase entwickelt und unterlag ständigen Änderungen, die sich bspw. aufgrund von zuvor nicht berücksichtigten bzw. nicht erkannten Restriktionen und Spezialfällen ergaben. Schlussendlich entstand dann das fertige Konzept, das die M-Objects und M-Relationships in Protégé darstellt und in den nachfolgenden Kapiteln beschrieben wird.

4.1 Mapping Grundlagen

Zur Vereinfachung der Darstellung und des Verständnisses des Mappings werden in diesem Zusammenhang einige generelle Grundlagen, die auch bereits teilweise in vorigen Kapiteln beschrieben wurden, nochmals erläutert.

Das Wissensmodell von Protégé verfügt über zahlreiche Konstrukte, die zur Darstellung von M-Objects und M-Relationships verwendet werden können. Jedes dieser Konstrukte in Protégé besitzt standardmäßig ein Attribut *name*. Der Wert dieses Attributes ist systemweit eindeutig, also ein sogenannter „unique name“ und somit kann jede Klasse, Instanz, usw. über dieses Attribut identifiziert werden. Der Lesbarkeit halber wird dieses Attribut nicht explizit angegeben, sondern nur mehr vom Namen gesprochen.

In der nachfolgenden Auflistung wird gezeigt, wie die Bausteine *class*, *instance* und *slot* für die Umsetzung der M-Objects und M-Relationships in Protégé verwendet werden.

M-Object-Klassen

Diese werden in Protégé mittels des Konstrukts *class* dargestellt. Sie sind Instanzen der Metaklasse `MObjectClass` und erben somit alle Attribute. Weiters sind M-Object-Klassen auch Subklassen der Klasse `MObject`. Auf die Metaklasse `MObjectClass` wird zu einem späteren Zeitpunkt noch genauer eingegangen. Der Name einer M-Object-Klasse setzt sich zusammen aus dem Namen eines M-Objects und der Abstraktionsebene, die sie repräsentiert (bspw. *AutoKategorie*).

M-Object

M-Objects werden in Protégé durch das Konstrukt *instance* dargestellt. M-Objects stellen direkte Instanzen einer konkreten M-Object-Klasse dar und erben somit auch deren Attribute. Beim Anlegen von M-Objects wird zumindest eine M-Object-Klasse erzeugt und zwar jene, von der das M-Object eine direkte Instanz darstellt. Der Name des M-Objects wird vom Benutzer vergeben (bspw. *Produkt*).

M-Relationship

Die Darstellung erfolgt in Protégé ebenfalls mittels des Konstruktes *instance*. Ein M-Relationship ist eine direkte Instanz der Klasse M-Relationship, auf die nachfolgend noch genauer eingegangen wird. Der Name des M-Relationships entsteht durch die Namen der beiden M-Objects, die es miteinander in Beziehung setzt und dem Namen der Beziehung, die vom Benutzer vergeben wird (bspw. *Produkt-produziertVon-Konzern*).

Level

Level sind Abstraktionsebenen und werden in Protégé durch das Konstrukt *instance* veranschaulicht. Die Bezeichnung eines Levels wird durch den Benutzer vergeben (bspw. *Kategorie*), der systemweit eindeutige Name wird automatisch generiert.

ConnectionLevel

Ein ConnectionLevel verbindet zwei Abstraktionsebenen miteinander und wird mittels des Konstrukts *instance* dargestellt. Ausschlaggebend für die Bezeichnung des ConnectionLevels

sind die beiden Abstraktionsebenen, die in Beziehung gebracht werden (bspw. *Kategorie-Industriesektor*), der systemweit eindeutige Name wird ebenfalls automatisch generiert.

Attribut

Attribute werden in Protégé durch das Konstrukt *slot* dargestellt. Diese Slots und deren Darstellung können vom Benutzer individuell zugeschnitten werden. Die Attribute für das Mapping enthalten einen Namen, eine Bezeichnung, einen Wertebereich, eine Beschreibung, einen Wert und einen Default-Wert. Der Name identifiziert das Attribut eindeutig, die Bezeichnung dient der Darstellung des Attributes und für den Wertebereich stehen die Typen *Boolean*, *String*, *Integer* und *Float* zur Verfügung. Optional dazu kann auch noch eine Beschreibung beigefügt werden. Die Bezeichnung des Attributes wird vom Benutzer vergeben, wobei der eindeutige Name automatisch aus der Bezeichnung und einer Zeichenkette generiert wird.

4.2 Strukturelles Mapping der M-Objects

Wie bereits erwähnt ist für die Umsetzung der M-Objects ein Modell nötig. Abbildung 8 zeigt schematisch wie die M-Objects in Protégé realisiert wurden.

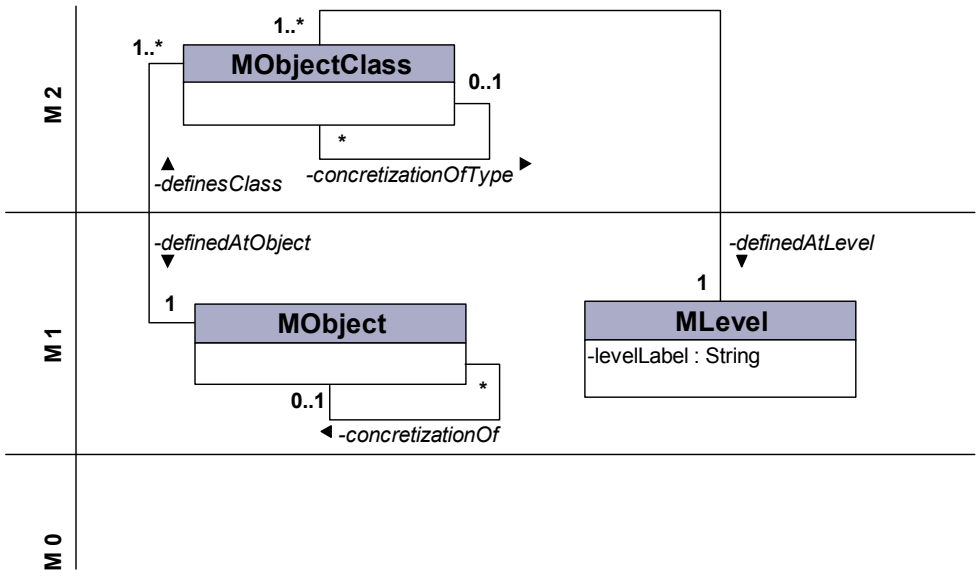

Abbildung 8: Metamodell M-Object

Grundlage des dargestellten Metamodells bilden die drei Klassen `MObjectClass`, `MObject` und `MLevel`, die mit den Attributen und den Beziehungen untereinander, die Basis für die Umsetzung der M-Objects in Protégé schaffen.

4.2.1 Metaklasse: MObjectClass

Die `MObjectClass` ist eine Metaklasse und somit sind ihre Instanzen wiederum Klassen, wobei diese als M-Object-Klassen bezeichnet werden. Sie enthält drei wesentliche Attribute `definedAtLevel`, `definedAtObject` und `concretizationOfType`.

Attribut: `definedAtLevel`

Das Attribut `definedAtLevel` definiert das Top-Level auf dem die Klasse und in weiterer Folge auch die direkten Instanzen dieser Klasse angesiedelt sind. Der Wertebereich von `definedAtLevel` sind die Instanzen der Klasse `MLevel`, wobei jede M-Object-Klasse mittels `definedAtLevel` genau eine Instanz von `MLevel` identifiziert. Ein so genannter Level kann jedoch mindestens einer oder auch mehreren M-Object-Klassen zugewiesen werden. Die Ausprägung des Attributes `definedAtLevel` der M-Object-Klasse *ProduktModell* ist eine Referenz auf das Level *Modell*.

Attribut: `concretizationOfType`

Das Attribut `concretizationOfType` definiert von welcher Klasse die aktuelle Klasse eine Konkretisierung darstellt. Zulässig als Wertebereich für das Attribut `concretizationOfType` sind alle Klassen, die als Metaklasse die `MObjectClass` besitzen. Dies trifft für alle M-Object-Klassen zu, da diese Instanzen der Metaklasse `MObjectClass` sind.

Jede M-Object-Klasse konkretisiert mittels `concretizationOfType` keine oder genau eine weitere M-Object-Klasse. Andererseits können auch mehrere Konkretisierungen einer M-Object-Klasse existieren.

Die M-Object-Klasse *ProduktKategorie* konkretisiert die M-Object-Klasse *ProduktKatalog*, weshalb die Ausprägung des Attributes `concretizationOfType` der M-Object-Klasse *ProduktKategorie* auf die M-Object-Klasse *ProduktKatalog* verweist.

Attribut: definedAtObject

Das Attribut definedAtObject verweist auf eine Instanz einer M-Object-Klasse, also auf ein konkretes M-Object, bei dem die M-Object-Klasse eingeführt wurde. Der Wertebereich von definedAtObject beinhaltet alle Instanzen der M-Object-Klassen, also alle konkreten M-Objects.

Jede M-Object-Klasse verweist mittels definedAtObject genau auf ein M-Object. Andererseits verweist eine Instanz der M-Object-Klasse auf ein oder mehrere M-Object-Klassen, wobei diese Beziehung durch das Attribut definesClass ausgedrückt wird und somit eine inverse Beziehung zu definedAtObject darstellt.

Die M-Object-Klassen *ProduktKatalog*, *ProduktKategorie*, *ProduktModell* und *ProduktPhysischesObjekt* verweisen alle mit dem Attribut definedAtObject auf das M-Object *Produkt.* Im Gegenzug dazu referenziert das Attribut definesClass des M-Objects *Produkt* auf alle 4 zuvor genannten M-Object-Klassen.

4.2.2 Klasse: MObject

Die Klasse MObject enthält zwei Attribute concretizationOf und definesClass. Eine Subklasse der Klasse MObject wird als M-Object-Klasse bezeichnet. Eine Instanz einer solchen M-Object-Klasse ist ein konkretes Objekt und heißt M-Object.

Der Name einer M-Object-Klasse setzt sich zusammen aus dem Namen des M-Objects und dem Namen der Abstraktionsebene. Die M-Object-Klasse *ProduktKategorie* setzt sich bspw. zusammen aus dem Namen des M-Objects *Produkt* und der Abstraktionsebene *Kategorie*.

Attribut: concretizationOf

Das Attribut concretizationOf definiert von welchem M-Object die konkrete Instanz einer M-Object-Klasse, also wiederum ein M-Object, eine Konkretisierung ist. Der Wertebereich von concretizationOf besteht aus der Menge aller M-Objects, wobei jedes M-Object mittels des Attributes concretizationOf auf kein oder genau ein anderes M-Object verweist. Andererseits kann jedoch ein M-Object keine oder mehrere Konkretisierungen besitzen.

Das Attribut `concretizationOf` spiegelt somit die Konkretisierungshierarchie wieder. Das M-Object *Auto* ist eine Konkretisierung des M-Objects *Produkt*, somit referenziert das Attribut `concretizationOf` des M-Objects *Auto* auf das M-Object *Produkt*. Das M-Object *Produkt* konkretisiert kein weiteres M-Object mehr, weshalb das Attribut `concretizationOf` keinen Wert besitzt.

Attribut: `definesClass`

Das Attribut `definesClass` ist wie bereits erwähnt, ein inverses Attribut zu `definedAtObject`, woraus sich ergibt, dass der Wertebereich von `definesClass` die Menge aller M-Object-Klassen ist. Ein konkretes M-Object identifiziert mittels `definesClass` mindestens ein oder auch mehrere M-Object-Klassen.
Das M-Object *Buch* bspw. referenziert mittels des Attributes `definesClass` die M-Object-Klassen *BuchKategorie*, *BuchModell* und *BuchPhysischesObjekt*.

4.2.3 Klasse: MLevel

Die Klasse `MLevel` enthält ein Attribut `levelLabel`. Jede Instanz dieser Klasse ist ein konkretes Objekt und wird als Level bezeichnet. Beispiele für Level sind *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt*.

Attribut: `levelLabel`

Das Attribut `levelLabel` definiert einen Bezeichner für die konkrete Instanz der Klasse `MLevel`, also eines Levels. Der Wertebereich von `levelLabel` beschränkt sich auf eine Zeichenkette.

4.2.4 Beispiel Mapping M-Objects

Alle nachfolgenden Abbildungen zeigen eine Darstellung eines konkreten Beispiels, das fortlaufend erweitert wird. Es beschreibt wie M-Objects in Protégé realisiert wurden. Alle in den Abbildungen enthaltenen Klassen sind Instanzen der Metaklasse `MObjectClass` und werden als M-Object-Klassen bezeichnet. Die Objekte links im Bild stellen Instanzen der

Klasse `MLevel` dar und heißen, wie erwähnt, Levels. Im Gegensatz dazu zeigen die Objekte rechts in der Darstellung Instanzen der M-Object-Klassen, die als M-Objects bezeichnet werden.

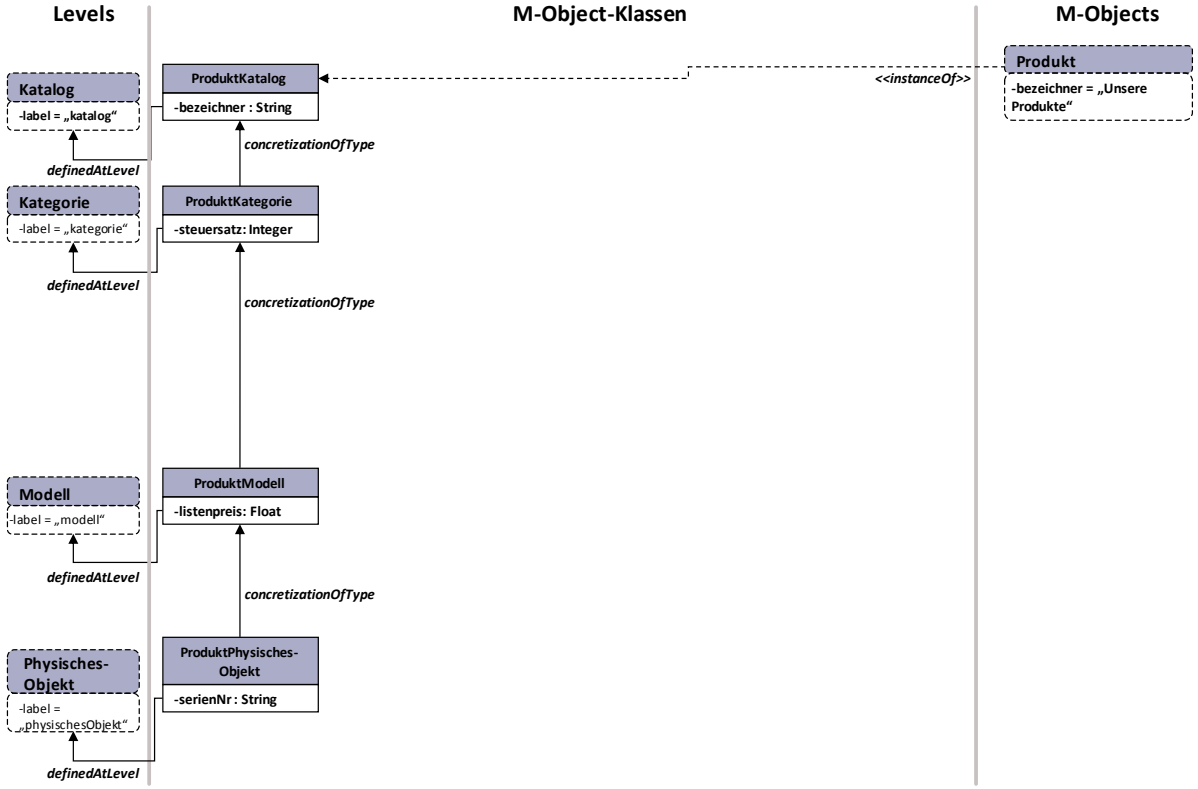

Abbildung 9: Darstellung der Umsetzung von M-Objects in Protégé (1)

Das M-Object *Produkt* wird in Abbildung 9 durch vier Abstraktionsebenen beschreiben. Dies sind *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt*. Um dies darzustellen werden die M-Object-Klassen *ProduktKatalog*, *ProduktKategorie*, *ProduktModell* und *ProduktPhysischesObjekt* erzeugt, deren Name sich aus dem M-Object und der Abstraktionsebene ergibt, aufgrund dessen sie erzeugt wurden. Das M-Object *Produkt* selbst ist eine Instanz der Klasse *ProduktKatalog*, die mit dem Attribut `definedAtLevel` auf die oberste Abstraktionsebene verweist. *ProduktKategorie*, *ProduktModell* und *ProduktPhysischesObjekt* referenzieren mit dem Attribut `definedAtLevel` auf die jeweilige Ebene auf der sie angelegt wurden. Somit zeigt das Attribut `definedAtLevel` der M-Object-Klasse *ProduktKategorie* auf das Level *Kategorie*, das Attribut

definedAtLevel der M-Object-Klasse *ProduktModell* auf das Level *Modell* und das Attribut definedAtLevel der M-Object-Klasse *ProduktPhysischesObjekt* auf das Level *PhysischesObjekt*.

Auf den verschiedenen Abstraktionsebenen werden auch Attribute eingeführt, die die M-Objects näher beschreiben. Auf der Ebene *Katalog* befindet sich das Attribut *bezeichner*, auf der Ebene *Kategorie* das Attribut *steuersatz*, auf der Ebene *Modell* das Attribut *listenpreis* sowie auf der Ebene *PhysischesObjekt* das Attribut *serienNr*. Das Attribut *bezeichner* der M-Object-Klasse *ProduktKatalog* beschreibt direkt eine Instanz dieser Klasse, in diesem Fall das M-Object *Produkt* mit dem Wert „Unsere Produkte“.

Die Hierarchie der einzelnen Abstraktionsebenen wird festgehalten durch das Attribut concretizationOfType der einzelnen M-Object-Klassen, wobei der Wert des Attributes concretizationOfType der M-Object-Klasse *ProduktKatalog* undefiniert ist, da diese Klasse auf der obersten Abstraktionsebene angesiedelt ist. Das Attribut concretizationOfType der M-Object-Klasse *ProduktKategorie* verweist auf die M-Object-Klasse *ProduktKatalog*, das Attribut concretizationOfType der M-Object-Klasse *ProduktModell* verweist auf die M-Object-Klasse *ProduktKategorie* und das Attribut concretizationOfType der M-Object-Klasse *ProduktPhysischesObjekt* verweist auf die M-Object-Klasse *ProduktModell*.

Alle M-Object-Klassen enthalten auch das Attribut definedAtObject, das der Lesbarkeit halber in den Abbildungen vernachlässigt wurde. Im Fall der M-Object-Klassen *ProduktKatalog*, *ProduktKategorie*, *ProduktModell* und *ProduktPhysischesObjekt* verweist das Attribut defindedAtObject auf das M-Object *Produkt*. Das Attribut definesClass des M-Objects *Produkt* referenziert auf die vier zuvor erwähnten M-Object-Klassen.

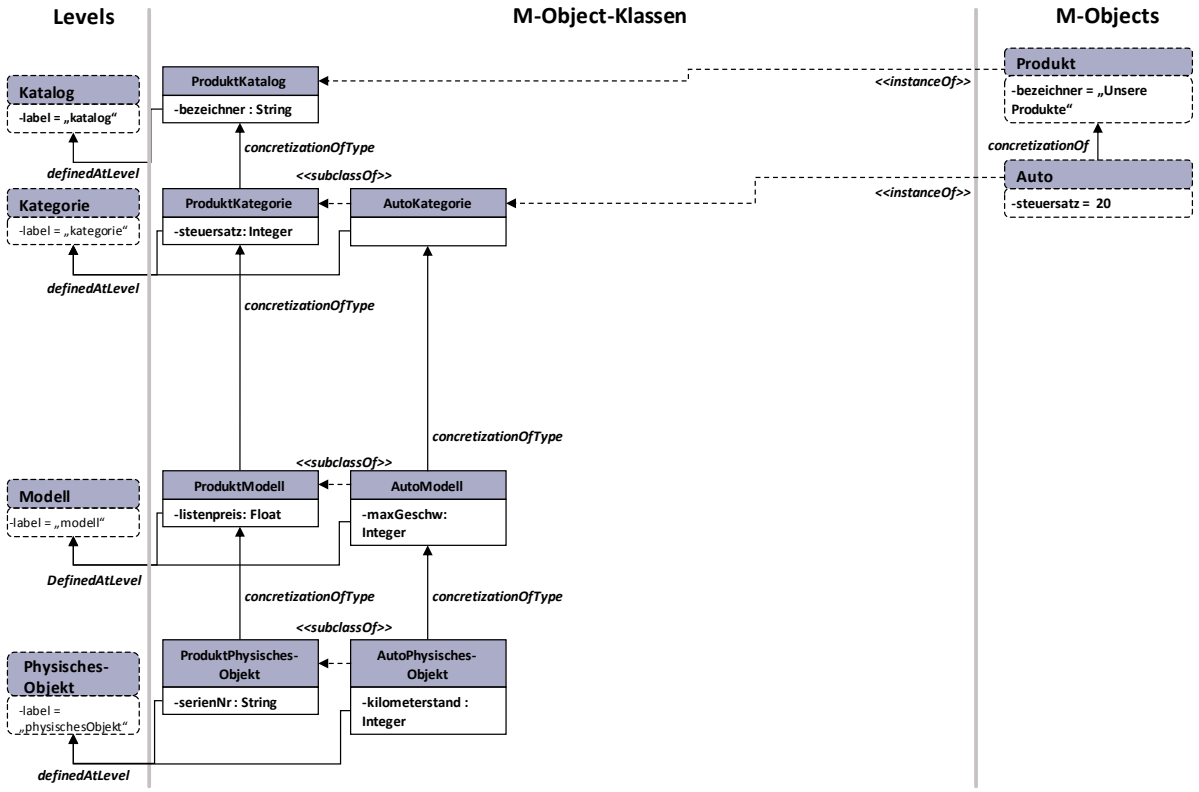

Abbildung 10: Darstellung der Umsetzung von M-Objects in Protégé (2)

Das M-Object *Auto* in Abbildung 10 ist eine Instanz der Klasse *AutoKategorie* und auch eine Konkretisierung von dem M-Object *Produkt*, weshalb es auch auf der nächst höheren Abstraktionsebene *Kategorie* angesiedelt ist. Diese Konkretisierungsbeziehung zwischen den M-Objects *Produkt* und *Auto* wird durch das Attribut `concretizationOf` des M-Objects *Auto* ausgedrückt. Die Einführung des M-Objects *Auto* hat zur Folge, dass die M-Object-Klassen *AutoKategorie*, *AutoModell* und *AutoPhysischesObjekt* erstellt werden. Diese sind untereinander wiederrum mittels des Attributes `concretizationOfType` verknüpft.

Das Attribut `defindAtObject` der neu erstellten M-Object-Klassen verweist auf das M-Object *Auto*. Das Attribut `definesClass` des M-Objects *Auto* verweist im umgekehrten Sinne auf die M-Object-Klassen *AutoKategorie*, *AutoModell* und *AutoPhysischesObjekt* **.** Zudem werden bei den neu erstellten M-Object-Klassen auf den verschiedenen Abstraktionsebenen neue Attribute wie *maxGeschw* und *kilometerstand* eingeführt.

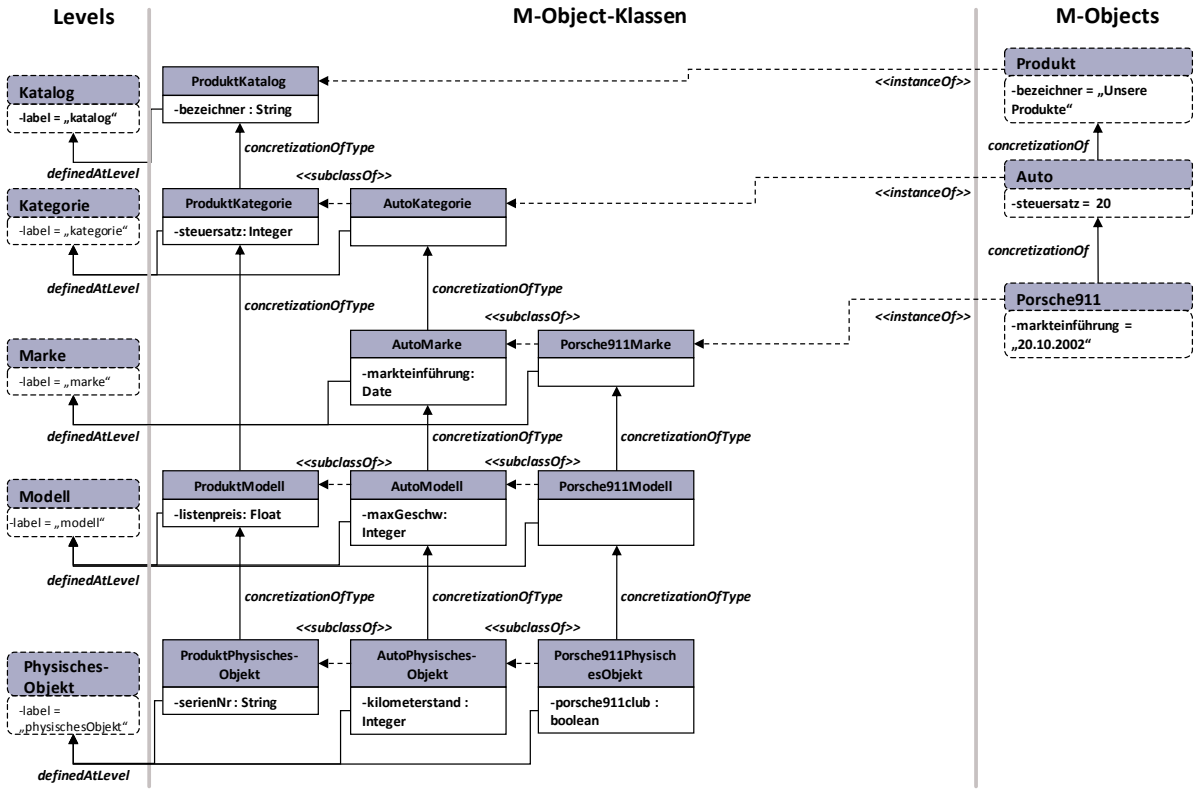

Abbildung 11: Darstellung der Umsetzung von M-Objects in Protégé (3)

In Abbildung 11 wird ein neuer Level *Marke* in die Abstraktionshierarchie eingeführt und damit verbunden auch ein neues M-Object *Porsche911*. Somit müssen aber auch die Konkretisierungsbeziehungen verändert werden. Die M-Object-Klasse *AutoModell* verweist nicht länger auf die M-Object-Klasse *AutoKategorie* sondern auf die neue M-Object-Klasse *AutoMarke*, die jetzt auf die M-Object-Klasse *AutoKategorie* verweist. Weiters werden neue M-Object-Klassen *Porsche911Marke*, *Porsche911Modell* und *Porsche911PhysichesObjekt* erstellt, wobei die M-Object-Klasse *Porsche911Modell* das Attribut *porsche911club* beinhaltet.

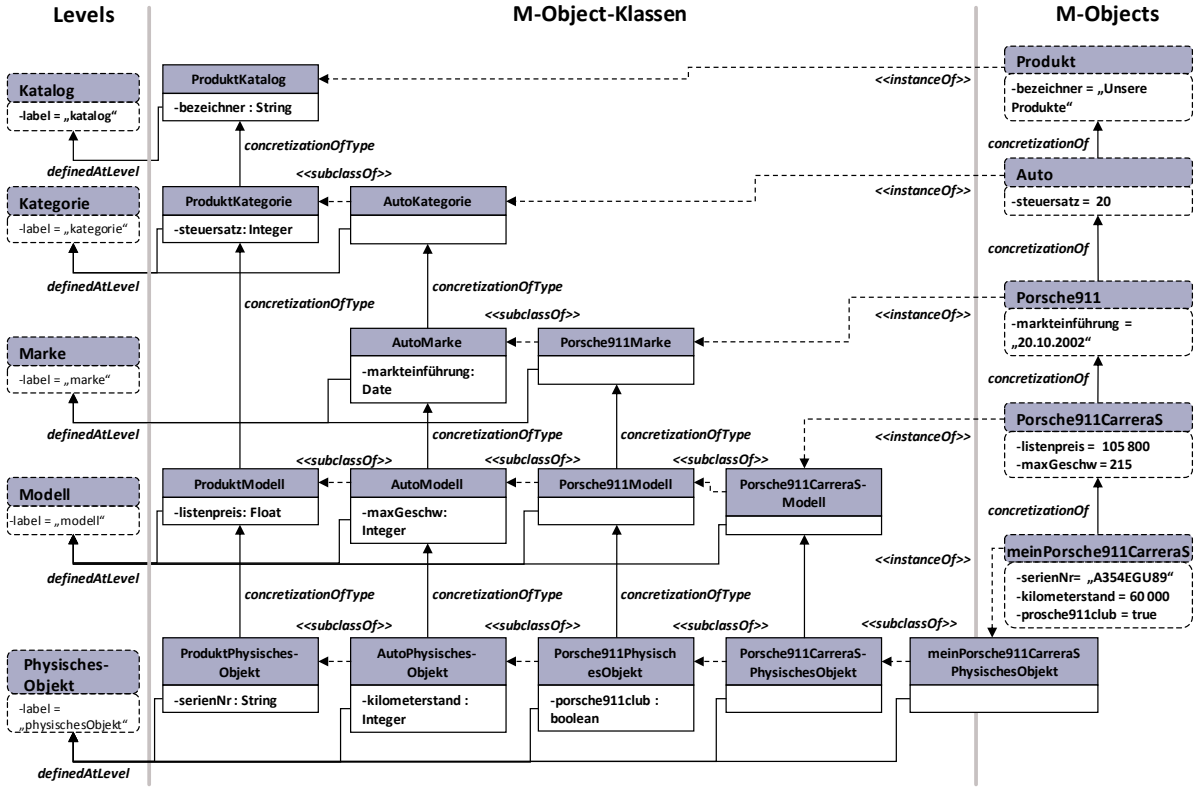

Abbildung 12: Darstellung der Umsetzung von M-Objects in Protégé (4)

In weiterer Folge werden die M-Objects *Porsche911CarreraS* und *meinPorsche911CarreraS* erstellt, wobei das M-Object *Porsche911CarreraS* eine Konkretisierung des M-Objects *Porsche911* und das M-Object *meinPorsche911CarreraS* wiederum das M-Object *Porsche911CarreraS* konkretisiert. Im Zuge der Erstellung dieser neuen M-Objects werden die M-Object-Klassen *Porsche911CarreraSModell*, *Porsche911CarreraSPhysichesObjekt* und *meinPorsche911CarreraSPhysichesObjekt* generiert, deren Abhängigkeiten untereinander in Abbildung 12 ersichtlich sind.

4.3 Strukturelles Mapping der M-Relationships

Abbildung 13 zeigt wie die M-Objects in Protégé realisiert wurden. Der Lesbarkeit halber wurde die Metaklasse `MObjectClass` und deren Attribute in der nachfolgenden Darstellung nicht berücksichtigt.

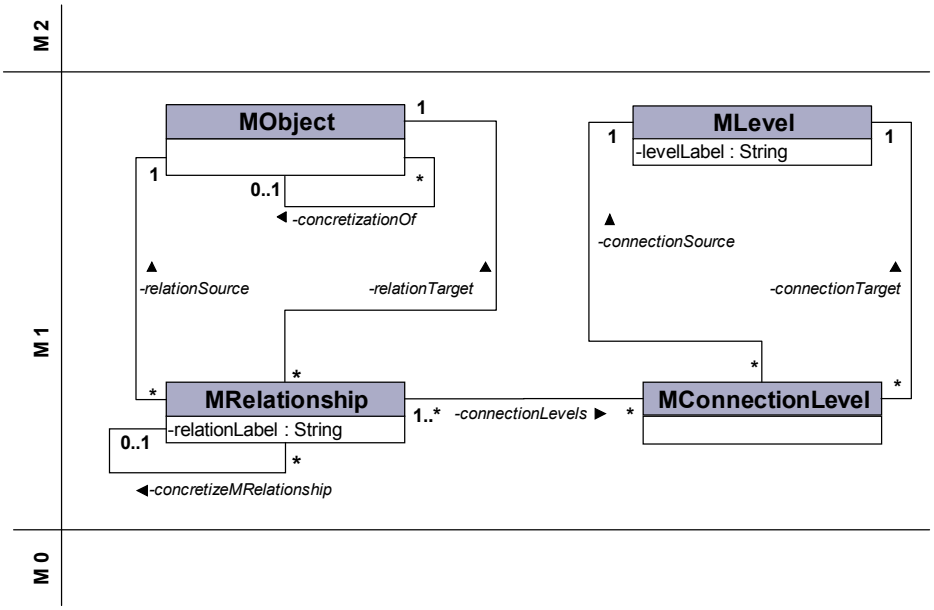

Abbildung 13: Metamodel M-Relationship

Kernstück der Multi-Level Relationships ist die Klasse `MRelationship` und die Klasse `MConnectionLevel`. Sie werden nachfolgend noch genauer beschrieben.

4.3.1 Klasse: MRelationship

Die Klasse `MRelationship` enthält fünf Attribute `relationSource`, `relationTarget`, `concretizeMRelationship`, `connectionLevels` und `relationLabel`. Instanzen dieser Klasse werden als M-Relationships bezeichnet.

Der Name eines M-Relationships setzt sich aus dem Wert des Attributes `relationLabel` und den Namen der beiden M-Objects zusammen, die miteinander in Beziehung stehen. Ein Beispiel hierfür wäre das M-Relationship *Produkt-produziertVon-Konzern*.

Attribut: `relationSource` und `relationTarget`

Das Attribut `relationSource` sowie das Attribut `relationTarget` verweisen beide auf M-Objects, die in einer Beziehung zueinander stehen. Der Wertebereich von `relationSource` und `relationTarget` ist somit die Menge aller M-Objects, wobei jedes M-Relationship mittels des Attributes `relationSource` bzw. `relationTarget`

auf genau ein M-Object verweist. Andererseits kann jedoch ein M-Object auf keine oder auch auf mehrere M-Relationships verweisen. Der Wert von `relationSource` wird als RelationSource, der Wert von `relationTarget` wird als RelationTarget bezeichnet.
Die Werte von RelationSource und RelationTarget des M-Relationships *Produkt-produziertVon-Konzern* wären somit die M-Objects *Produkt* und *Konzern*

Attribut: `concretizeMRelationship`

Das Attribut `concretizeMRelationship` definiert die Konkretisierung eines M-Relationships durch ein anderes M-Relationship. Der Wertebereich von `concretizeMRelationship` ist somit die Menge aller M-Relationships. Ein M-Relationship kann keine oder genau eine Konkretisierung eines anderen M-Relationship sein. Andererseits kann ein M-Relationship von keinem oder auch von mehreren M-Relationships konkretisiert werden.
Das Attribut `concretizeMRelationship` stellt somit die Konkretisierungsbeziehungen zwischen M-Relationships dar. Bezogen auf das im nächsten Kapitel beschriebene Beispiel ist der Wert des Attributes `concretizeMRelationship` des M-Relationships *Auto-produziertVon-Autohersteller* eine Referenz auf das M-Relationship *Produkt-produziertVon-Konzern*.

Attribut: `connectionLevels`

Das Attribut `connectionLevels` besitzt als Wertebereich die Menge aller ConnectionLevels. Eine M-Relationship kann als Werte für das Attribut `connectionLevels` keine oder mehrere Instanzen der Klasse `MConnectionLevels` aufweisen. Ein ConnectionLevel wird von mindestens einem oder auch mehreren M-Relationships in Anspruch genommen.
Ein M-Relationship kann mehrere ConnectionLevels besitzen. In diesem Zusammenhang verweist das im nächsten Kapitel dargestellte M-Relationship *Produkt-produziertVon-Konzern* auf die ConnectionLevels *Kategorie-Industriesektor*, *Modell-Unternehmen* und *PhysischesObjekt-Fabrik.*

Attribut: `relationLabel`

Das Attribut `relationLabel` definiert einen Bezeichner für ein M-Relationship. Der Wertebereich von `relationLabel` beschränkt sich auf String. Der endgültige Name des M-Relationships setzt sich zusammen aus den Werten der Attribute `relationSource`, `relationTarget` und `relationLabel`.

Das Attribut `relationLabel` des M-Relationships *Produkt-produziertVon-Konzern* hat den Wert *„produziertVon“*.

4.3.2 Klasse: MConnectionLevel

Die Klasse `MConnectionLevel` enthält zwei Attribute `connectionSource` und `connectionTarget`. Der Wertebereich dieser Attribute enthält alle Instanzen der Klasse `MLevel`. Instanzen der Klasse `MConnectionLevel` werden als ConnectionLevels bezeichnet.

Der Name eines ConnectionLevels setzt sich zusammen aus der Bezeichnung der Level, die miteinander in Verbindung gebracht werden, also dem Wert des Attributes `connectionSource` und `connectionTarget`. Ein Beispiel hierfür wäre das ConnectionLevel *Kategorie-Industriesektor,* das die Level *Kategorie* und *Industriesektor* miteinander in Verbindung setzt.

Attribut: `connectionSoure` und `connectionTarget`

Die Attribute `connectionSoure` und `connectionTarget` beinhalten jeweils einen Level, die untereinander in Verbindung stehen. Der Wertebereich von `connectionSource` und `connectionTarget` ist die Menge aller Levels, wobei jeder ConnectionLevel mittels des Attributes `connectionSource` bzw. `connectionTarget` genau auf einen Level verweist. Eine Instanz der Klasse `MLevel` kann jedoch von keinem oder auch von mehreren ConnectionLevels verwendet werden. Werte für das Attribut `connectionSource` bzw. `connectionTarget` eines konkreten ConnectionLevels sind bspw. die Levels *Kategorie* und *Industriesektor*.

4.3.3 Beispiel Mapping M-Relationships

Nachfolgendes Beispiel zeigt die Umsetzung der M-Relationships in Protégé. Der Lesbarkeit halber wurden für die Darstellung nicht relevante Attribute, bspw. die Attribute der M-Objects (`definesClass`, `concretizationOf`) vernachlässigt.

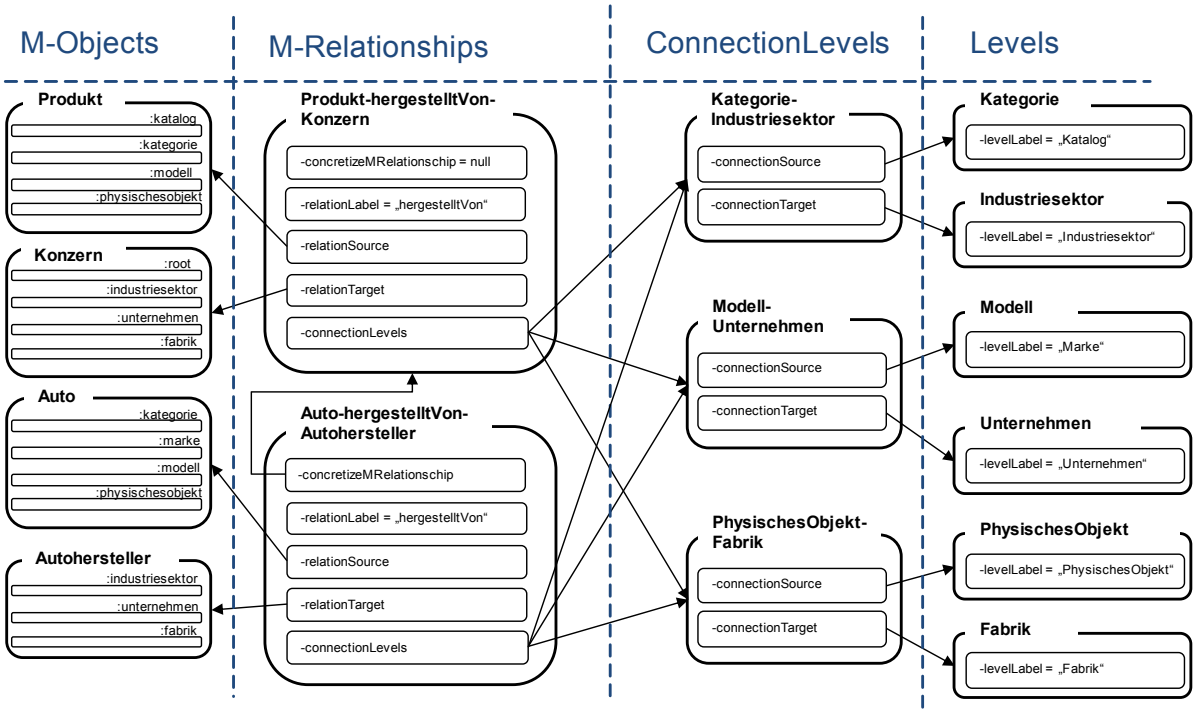

Abbildung 14: Darstellung von M-Relationships in Protégé

Abbildung 14 zeigt zwei M-Relationships nämlich *Produkt-produziertVon-Konzern* und *Auto-produziertVon-Autohersteller.* Der Name setzt sich zusammen aus dem Attribut `relationLabel` des M-Relationships, der vom Benutzer vergeben wird, und den beiden M-Objects, die es miteinander verlinkt. Dabei sind die Attribute `relationSource` und `relationTarget` ausschlaggebend, die beide auf ein M-Object referenzieren und somit Namensgeber des M-Relationships sind. Betrachtet man, wie im Beispiel angeführt, das M-Relationship *Auto-produziertVon-Autohersteller*, erkennt man, dass das Attribut `relationSource` auf das M-Object *Produkt* und das Attribut `connectionTarget` auf das M-Object *Konzern* verweist. Daraus ergibt sich dann die Bezeichnung des M-Relationships *Produkt-produziertVon-Konzern*.

Das M-Relationship *Auto-produziertVon-Autohersteller* ist eine Konkretisierung des M-Relationships *Produkt-produziertVon-Konzern*. Diese Beziehung wird durch das Attribut `concretizeMRelationship` dargestellt, dessen Wert auf das übergeordnete M-Relationship verweist. Dabei erbt das konkrete M-Relationship alle Beziehungen zwischen den Abstraktionsebenen, mit Ausnahme jener, wo diese Abstraktionsebenen in den Source- oder Target M-Objects nicht vorkommen. Das bedeutet, dass das M-Relationship *Auto-produziertVon-Autohersteller* alle Beziehungen zwischen den Abstraktionsebenen *Kategorie*, *Industriesektor*, *Modell, Unternehmen, PhysischesObjekt* und *Fabrik* erbt, da die M-Objects *Auto* und *Autohersteller* diese Abstraktionsebenen aufweisen.

Ein M-Relationship kann, wie schon erwähnt, zwei M-Objects über mehrere Abstraktionsebenen hinweg in Verbindung setzten. Dies geschieht über das Attribut `connectionLevels`, das mehrere Werte annehmen kann und so auf verschiedene Verknüpfungen von Abstraktionseben verweist. Das M-Relationship *Auto-produziertVon-Autohersteller* referenziert bspw. über das Attribut `connectionLevels` die ConnectionLevels *Kategorie-Industriesektor*, *Modell-Unternehmen* und *PhysischesObjekt-Fabrik*.

Ein ConnectionLevel besteht aus zwei Attributen, die beide auf einen Level verweisen. Es sind dies die Attribute `connectionSource` und `connectionTarget`, deren Werte auch gleichzeitig als Namensgeber für den ConnectionLevel fungieren. Der ConnectionLevel *Kategorie-Industriesektor* bspw. erhält seinen Namen durch die Referenzierung von `connectionSource` und `connectionTarget` auf die Level *Kategorie* und *Industriesektor*.

5. Operationales Mapping

Aufbauend auf das strukturelle Mapping in dem das Konzept zur Darstellung von M-Objects und M-Relationships in Protègè beschrieben wurde, werden in diesem Abschnitt die verschiedenen Operationen beschreiben, die Veränderungen der M-Objects und M-Relationships nach sich ziehen.

5.1 Operationen auf M-Objects

Operationen wie das Anlegen, Löschen, Verändern, etc. von M-Objects haben Auswirkungen auf die interne Struktur des Modells. Vor allem die Klassenstruktur, die Ausprägungen der einzelnen Attribute, usw. unterliegen aufgrund der Operationen ständig Veränderungen. Im Folgenden werden die einzeln Operation aufgelistet, die für die Bearbeitung von M-Objects zulässig sind.

- Erstellen von M-Objects
- Löschen von M-Objects
- Umbenennen von M-Objects
- Verschieben von M-Objects
- Erstellen eines Levels
- Löschens eines Levels
- Umbenennen eines Levels
- Erstellen, Ändern und Löschen von Attributen

Alle diese oben dargestellten Operationen haben Einfluss auf das erstellte Modell. In den folgenden Kapiteln werden diese Operationen genauer beschrieben. Um die Darstellung zu erleichtern wurden in den verwendeten Grafiken die Attribute `definedAtObject` und `definesClass` nicht berücksichtigt, da ansonsten die Lesbarkeit darunter gelitten hätte.

5.1.1 Erstellen von M-Objects

Wird ein neues M-Object eingeführt, das keine Konkretisierung eines anderen M-Objects ist, so muss im Zuge der Erstellung auch eine Abstraktionsebene also ein Level angelegt werden, der dann den Top-Level dieses M-Objects darstellt.

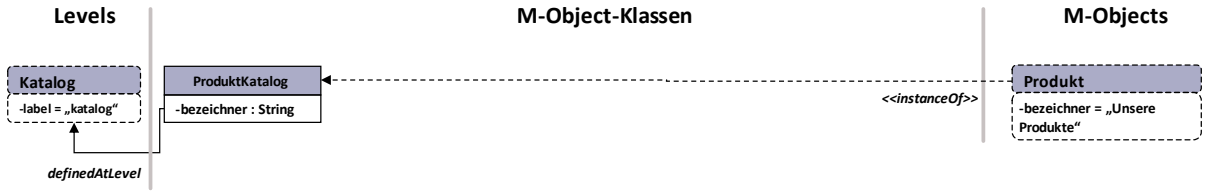

Abbildung 15: Erstellen von M-Objects (1)

Abbildung 15 zeigt das M-Object *Produkt*, die M-Object-Klasse *ProduktKatalog* und das Level *Katalog*. Mit der Erstellung des M-Objects *Produkt* musste auch zeitgleich der Level *Katalog* erstellt werden, da jedes M-Object eine Abstraktionsebene braucht. Resultierend aus dem M-Object und dem Level ergibt sich dann die M-Object-Klasse *ProduktKatalog*. Dieser M-Object-Klasse wurde ein Attribut *bezeichner* hinzugefügt, dessen Ausprägung das M-Object *Produkt* beschreibt.

Nach dem Erstellen des M-Objects *Produkt*, ergeben sich neue Werte für die Attribute der einzelnen Konstrukte, die in Tabelle 1 genauer beschrieben werden.

Konstrukt	Name	Attribute und Werte
M-Object	*Produkt*	-definesClass = <class>*ProduktKatalog* -concretizationOf = null
		-bezeichner = „Unsere Produkte“
M-Object-Klasse	*ProduktKatalog*	-concretizationOfType = null -definedAtObject = <instance>*Produkt* -definedAtLevel = <instance>*Katalog*
		-bezeichner : String
Level	*Katalog*	-levelLabel = <String> *Katalog*

Tabelle 1: Attributwerte beim Anlegen von M-Objects (1)

Wird ein M-Object auf einem existierendem Level erstellt, was automatisch heißt, dass es sich um eine Konkretisierung eines anderen M-Objects handelt, so müssen, abgesehen vom M-Object, auch M-Object-Klassen für alle Level angelegt werden, die vom übergeordnetem M-Object geerbt werden.

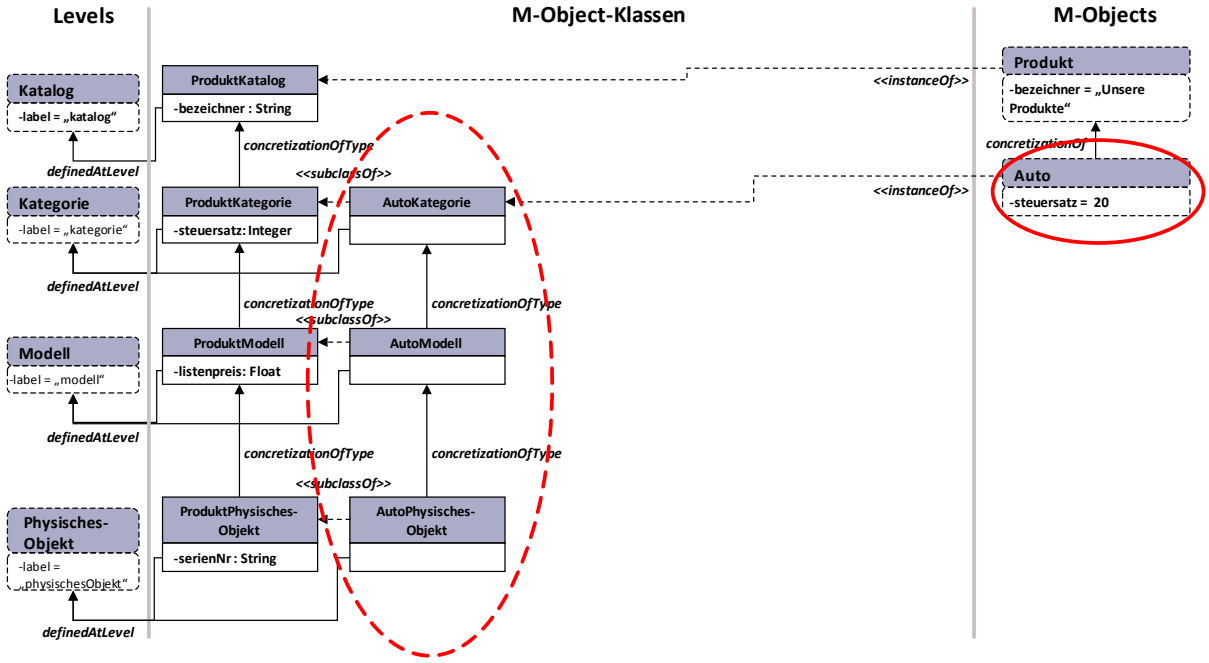

Abbildung 16: Erstellen von M-Objects (2)

Abbildung 16 zeigt, welche Veränderungen die Erstellung eines M-Objects *Auto* auf dem Level *Kategorie* mit sich bringt. Das M-Object *Auto* ist dabei eine Konkretisierung des M-Objects *Produkt*, das die Level *Kategorie*, *Modell* und *PhysischesObjekt* an das M-Object *Auto* vererbt. Dies hat zur Folge, dass für jeden Level eine eigene M-Object-Klasse eingeführt werden muss. Dies sind die M-Object-Klassen *AutoKategorie*, *AutoModell* und *AutoPhysischesObjekt*.

Nach dem Erstellen des M-Objects *Auto*, ergeben sich neue Werte für die Attribute der einzelnen Konstrukte, die in Tabelle 2 genauer beschrieben werden.

Konstrukt	Name	Attribute und Werte
M-Object	*Auto*	-definesClass = <class>*AutoKategorie* = <class>*AutoModell* = <class>*AutoPhysischesObjekt* -concretizationOf = <instance>*Produkt*
		-steuersatz = <Integer>20
M-Object-Klasse	*AutoKategorie*	-concretizationOfType = null -definedAtObject = <instance>*Auto* -definedAtLevel = <instance>*Kategorie*
M-Object-Klasse	*AutoModell*	-concretizationOfType = <class>*AutoKategorie* -definedAtObject = <instance>*Auto* -definedAtLevel = <instance>*Modell*
M-Object-Klasse	*AutoPhysischesObjekt*	-concretizationOfType = <class>*AutoModell* -definedAtObject = <instance>*Auto* -definedAtLevel = <instance>*PhysischesObjekt*

Tabelle 2: Attributwerte beim Anlegen von M-Objects (2)

5.1.2 Löschen von M-Objects

Das Löschen eines M-Objects hat Auswirkungen auf seine Konkretisierungen sowie auf die Abstraktionsebenen, die bei diesem M-Object angelegt wurden. Diese werden ebenfalls nicht mehr benötigt und somit auch gelöscht. Auch alle damit verbundenen M-Object-Klassen haben somit ihren Zweck erfüllt und müssen entfernt werden.

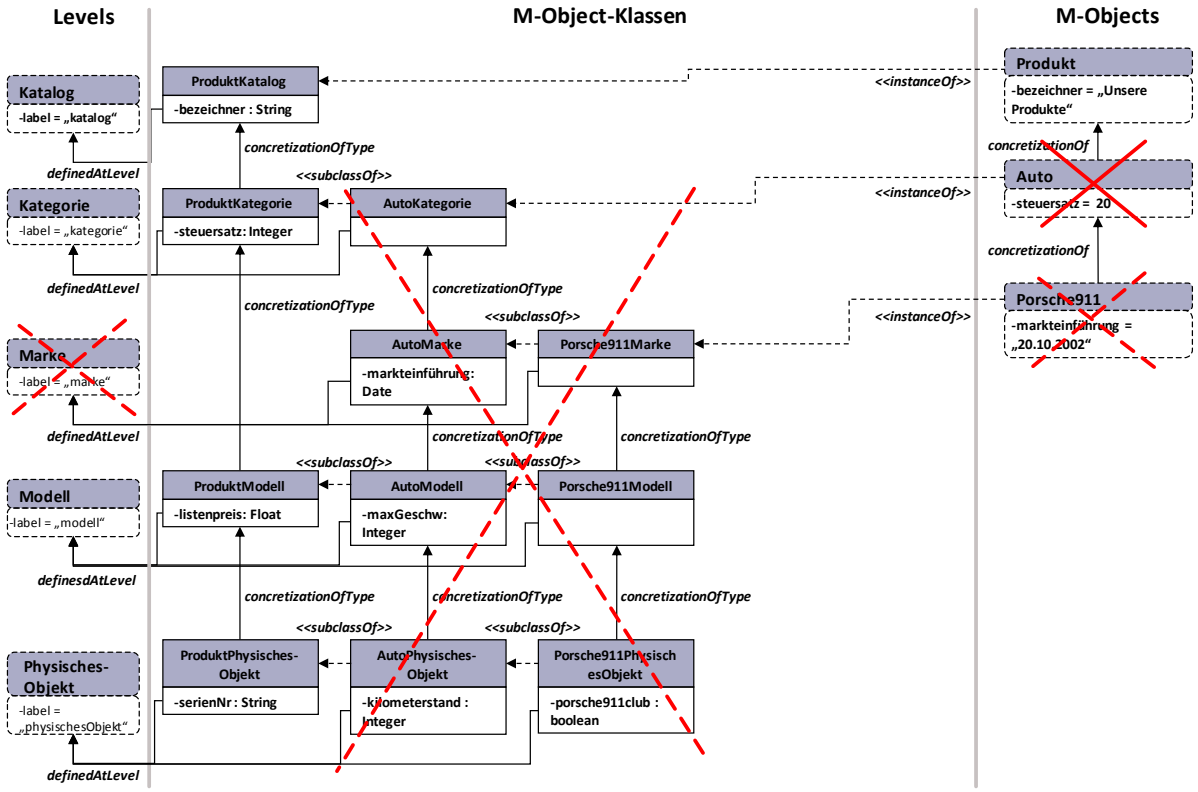

Abbildung 17: Löschen von M-Objects

In Abbildung 17 wird das M-Object *Auto* entfernt, wobei damit verbunden auch das konkretisierende M-Object *Porsche911* gelöscht wird. Zusätzlich wurde beim M-Object *Auto* ein neuer Level *Marke* eingeführt, der durch die Löschung des M-Objects *Auto* ebenfalls entfernt werden muss. Auch sämtliche Attribute, die auf den verschiedenen Abstraktionsebenen von *Auto* und *Porsche911* erstellt wurden, werden nicht mehr benötigt.

Durch das Entfernen des M-Objects *Auto* werden die Tabelle 3 dargestellten Elemente gelöscht.

Konstrukt	Name
M-Objects	*Auto, Porsche911*
M-Object-Klassen	*AutoKategorie, AutoMarke, AutoModell, AutoPhysischesObjekt, Porsche911Marke, Porsche911Modell, Porsche911PhysischesObjekt*
Attribute	*markteinführung, maxGeschw, kilometerstand, porsche911Club*

Tabelle 3: Löschen von M-Objects

5.1.3 Umbenennen von M-Objects

Grundsätzlich ist das Ändern von Namen innerhalb eines Modells keine große Herausforderung. Da allerdings der Name eines M-Objects auch Namensgeber für zahlreiche M-Object-Klassen und M-Relationships ist, müssen im Zuge der Änderung des Namens eines M-Objects auch die Namen dieser Konstrukte abgeändert werden.

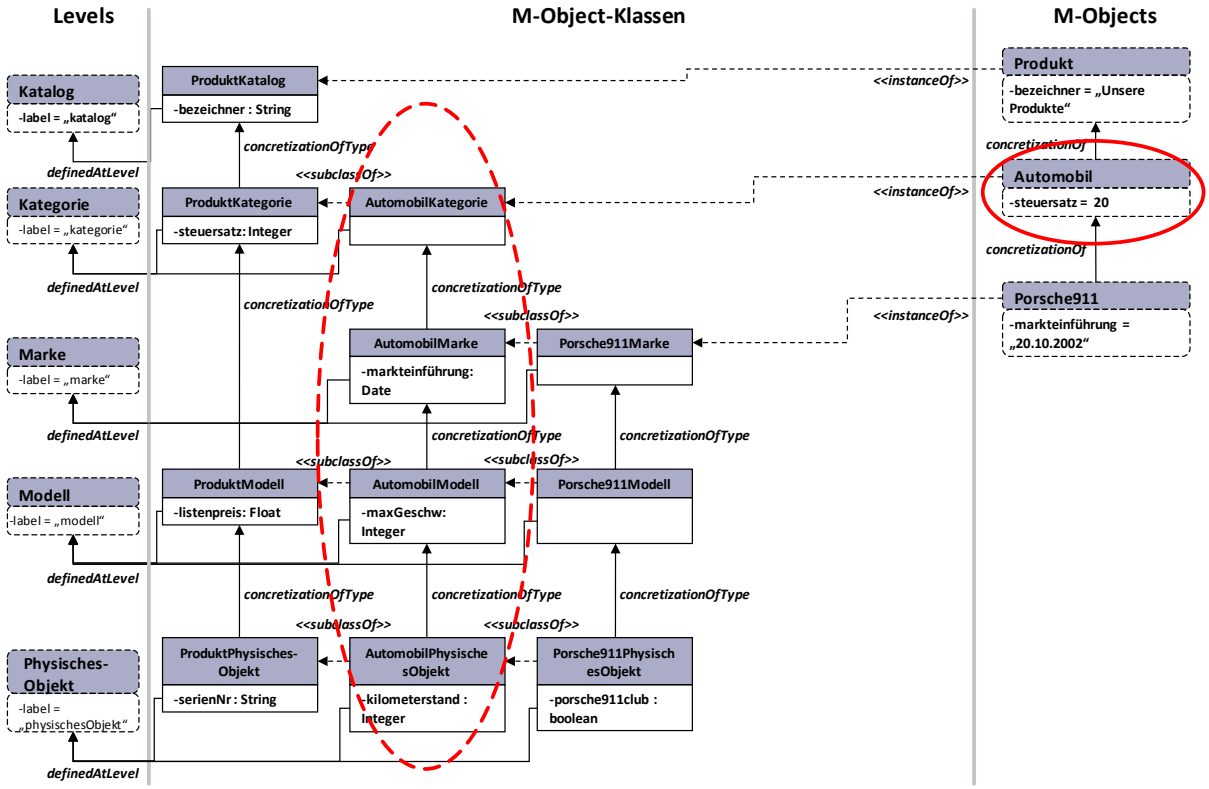

Abbildung 18: Umbenennen von M-Objects

In Abbildung 18 wurde das M-Object *Auto* in *Automobil* umbenannt. Dies hat Auswirkungen auf die M-Object-Klassen *AutoKategorie*, *AutoMarke*, *AutoModell* und *AutoPhysischesObjekt*. *AutoKategorie* wird abgeändert in *AutomobilKategorie*, *AutoMarke* in *AutomobilMarke*, usw. Wie erwähnt zieht die Namensänderung eines M-Objects auch Veränderungen der M-Relationships auf sich, mit denen das konkrete M-Object in Verbindung steht. Das M-Relationship *Auto-produziertVon-Autohersteller*, das allerdings in Abbildung 18 grafisch nicht veranschaulicht ist, muss ebenfalls umbenannt werden in *Automobil-produziertVon-Autohersteller*.

5.1.4 Verändern der Konkretisierungshierarchie eines M-Objects

Damit ist gemeint, dass ein M-Object aus einer Teilhierarchie in eine andere Teilhierarchie verschoben wird, wobei somit das M-Object die Konkretisierung eines anderen M-Objects wird. Dies ist nur dann möglich bzw. wird zugelassen, wenn das neue Parent M-Object dieselben Abstraktionsebenen aufweist wie das ursprüngliche, übergeordnete M-Object. Im Zuge dieser Änderungen müssen die Subklassenbeziehungen der mit dem M-Objects verbundenen M-Object-Klassen geändert werden.

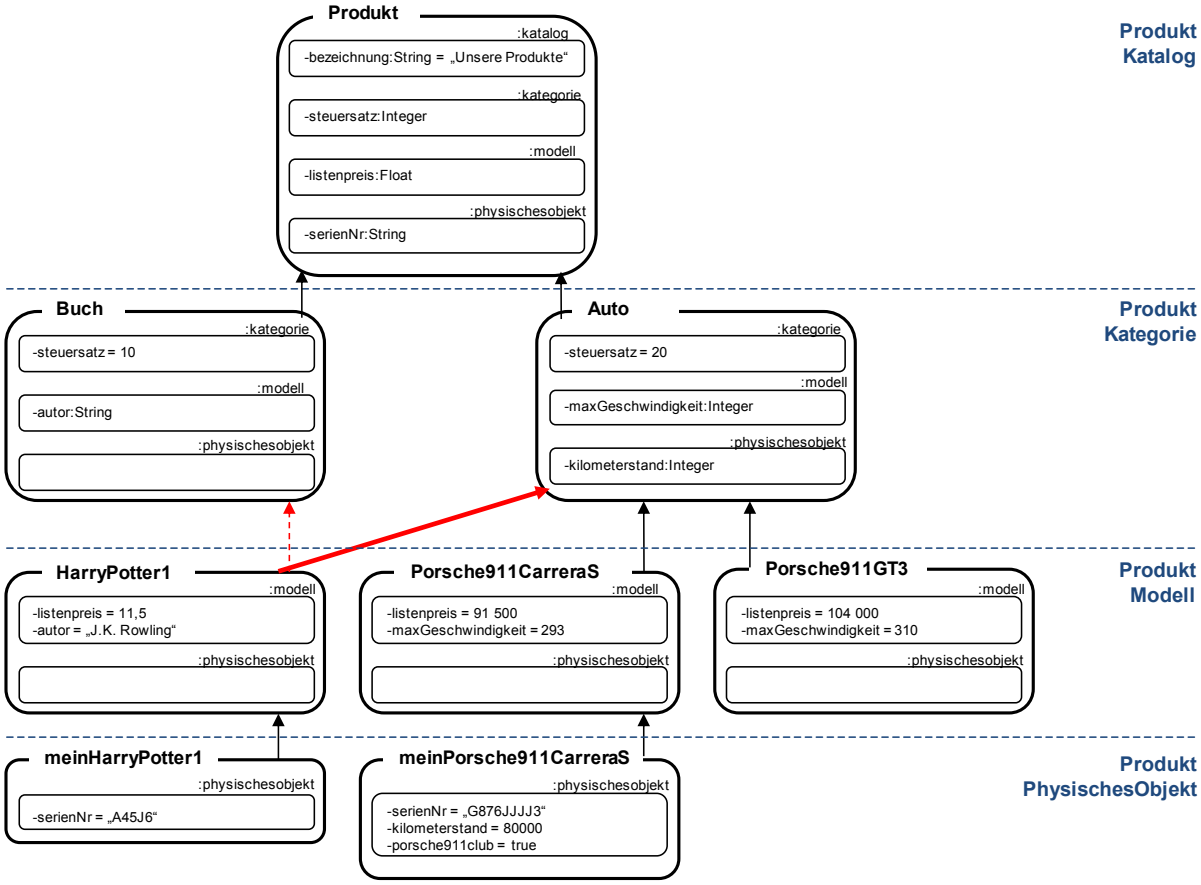

Abbildung 19: Verändern der Konkretisierungshierarchie eines M-Objects

Das Beispiel in Abbildung 19 stellt unter anderem die beiden M-Objects *Buch* und *Auto* dar. Beide M-Objects besitzen dieselbe Struktur bzw. dieselben Abstraktionsebenen *Kategorie*, *Modell* und *PhysischesObjekt*. Somit ist es möglich das M-Object *HarryPotter1* inklusive der konkretisierenden M-Objects des M-Objects *Buch* zu lösen und dem M-Object *Auto* zuzuweisen. *HarryPotter1* ist dann eine Konkretisierung von *Auto*, was zwar möglich wäre

aber natürlich nicht viel Sinn ergibt, da das M-Object *HarryPotter1* inklusive seiner konkretisierenden M-Objects keinen Bezug zur Autoindustrie aufweist. Die M-Object-Klassen *HarryPotter1Modell* und *HarryPotter1PhysischesObjekt* wären nicht länger Subklassen der M-Object-Klassen *BuchModell* und *BuchPhysischesObjekt* sondern von *AutoModell* und *AutoPhysischesObjekt*. Weiters würden auch alle Attribute, die beim M-Object *Buch* für die untergeordneten Abstraktionsebenen angelegt wurden (bspw. *autor*), nicht mehr länger in der Abstraktionshierarchie des M-Objects *HarryPotter1* aufscheinen. Stattdessen werden von dem neuen übergeordneten M-Object *Auto* die Attribute der untergeordneten Abstraktionsebenen (*maxGeschwindigkeit*, *kilometerstand*) geerbt. *HarryPotter1* würde somit auf der Ebene *modell* das Attribut *maxGeschwindigkeit* ausprägen.

Das gewählte Beispiel ist in diesem Zusammenhang nicht sonderlich sinnvoll, doch zeigt es, dass es in einem kleinen Rahmen möglich ist, die Konkretisierungen von M-Objects zu verändern. Ein besser Anwendungsfall, der sich allerdings grafisch schwer hätte veranschaulichen lassen, wäre gewesen, wenn der Online-Store neue Automodelle in sein Produktsortiment aufgenommen hätte. Wären diese Automodelle bereits in einem Modell integriert und besäßen dieselben Abstraktionshierarchien, könnten sie einfach ins neue Modell überführt werden.

5.1.5 Erstellen eines Levels

Beim Anlegen einer neuen Abstraktionsebene, eines Levels, gilt es zwischen drei verschiedenen Möglichkeiten zu unterschieden:

- Anlegen eines neuen Levels ohne bestehendes M-Object.
- Anlegen eines neuen Levels innerhalb einer Abstraktionsebene eines bestehenden M-Objects, das durch kein anderes M-Objects konkretisiert wird.
- Anlegen eines neuen Levels innerhalb einer Abstraktionsebene eines bestehenden M-Objects, das bereits durch andere M-Objects konkretisiert wird.

Wird ein neuer Level angelegt und es existiert kein M-Object so muss gleichzeitig im Zuge des Erstellens des neuen Levels ein M-Object angelegt werden, da ein Level der in keiner Verbindung zu einem M-Object steht, nicht existieren darf.

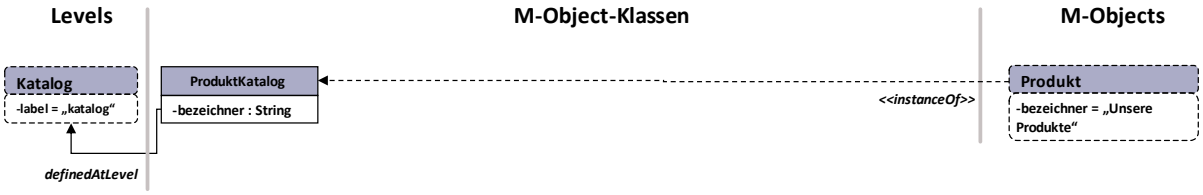

Abbildung 20: Erstellen eines Levels (1)

Abbildung 20 zeigt wie der neue Level *Katalog* erstellt wurde. Nachdem allerdings noch kein M-Object existiert, muss zeitgleich vom Benutzer ein neues M-Object *Produkt* angelegt werden. Die daraus resultierende M-Object-Klasse *ProduktKatalog* beschreibt das M-Object *Produkt* unter anderem durch das im Nachhinein zusätzlich angelegte Attribut *bezeichner*.

Nach dem Erstellen des Levels *Katalog* und des M-Objects *Produkt* ergeben sich neue Werte für die Attribute der einzelnen Konstrukte, die in Tabelle 4 genauer erläutert werden.

Konstrukt	Name	Attribute und Werte
M-Object	*Produkt*	-`definesClass` = <class>*ProduktKatalog* -`concretizationOf` = null
		-`bezeichner` = „Unsere Produkte“
M-Object-Klasse	*ProduktKatalog*	-`concretizationOfType` = null -`definedAtObject` = <instance>*Produkt* -`definedAtLevel` = <instance>*Katalog*
		-`bezeichner` : String
Level	*Katalog*	-`levelLabel` = <String> *Katalog*

Tabelle 4: Attributwerte beim Anlegen eines Levels (1)

Wird ein neuer Level innerhalb einer Abstraktionshierarchie eines M-Objects angelegt und wird dieses M-Object noch durch kein weiteres M-Objects konkretisiert, so ändert sich lediglich die Abstraktionshierarchie. Das bedeutet, dass der Wert des Attributes

`concretizationOfType` der M-Object-Klassen verändert werden muss, um die neue Abstraktionshierarchie darstellen zu können.

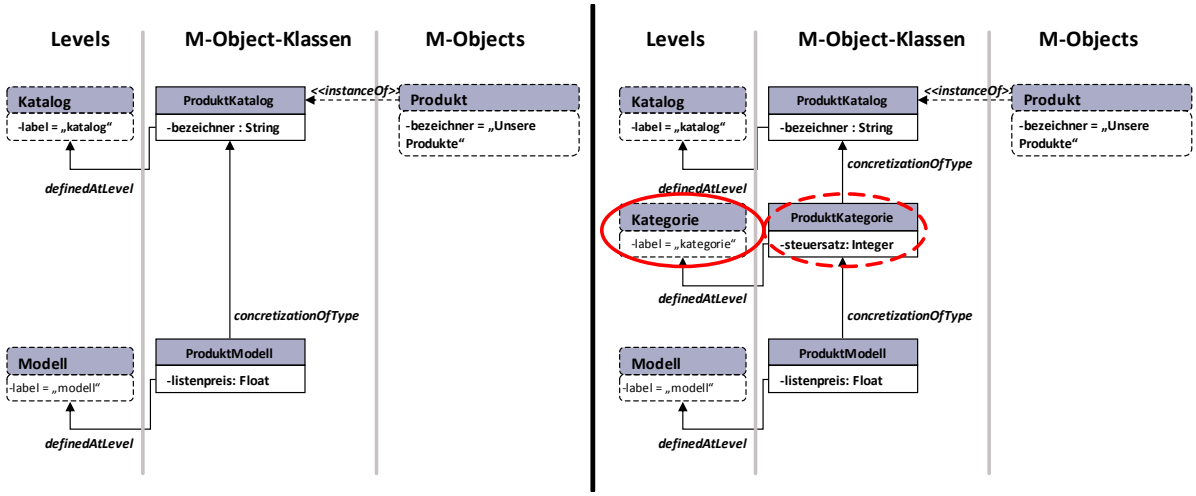

Abbildung 21: Erstellen eines Levels (2)

Die Abbildung 21 (linke Seite) zeigt das M-Object *Produkt* und die beiden Abstraktionsebenen *Katalog* und *Modell*. Für das M-Object *Produkt* wurden diese beiden Levels angelegt und die daraus resultierenden Klassen sind *ProduktKatalog* und *ProduktModell*, wobei die Konkretisierungsbeziehung durch das Attribut `concretizationOfType` gegeben ist. Die M-Object-Klasse *ProduktModell* verweist mittels dieses Attributes auf die M-Object-Klasse *ProduktKatalog*.

Im rechten Teil der Abbildung 21 wird nun ein neuer Level *Kategorie* als Konkretisierung des Levels *Katalog* eingeführt. Es entsteht somit eine neue M-Object-Klasse *ProduktKategorie*, die mittels des Attributes `concretizationOfType` auf die M-Object-Klasse *ProduktKatalog* verweist. Die M-Object-Klasse *ProduktModell* hingegen verweist nicht länger auf die M-Object-Klasse *ProduktKatalog* sondern auf die M-Object-Klasse *ProduktKategorie*.

Nach dem Erstellen des Levels *Kategorie* ergeben sich neue Werte für die Attribute der einzelnen Konstrukte, die in Tabelle 5 dargestellt werden.

Konstrukt	Name	Attribute und Werte
M-Object	*Produkt*	-definesClass = <class>*ProduktKatalog* = <class>*ProduktKategorie* = <class>*ProduktModell* -concretizationOf = <instance>*Produkt*
		-bezeichner = <String> „Unsere Produkte“
M-Object-Klasse	*ProduktKatalog*	-concretizationOfType = null -definedAtObject = <instance>*Produkt* -definedAtLevel = <instance>*Katalog*
		-bezeichner : String
M-Object-Klasse	*ProduktKategorie*	-concretizationOfType = <class>*ProduktKatalog* -definedAtObject = <instance>*Produkt* -definedAtLevel = <instance>*Kategorie*
		-steuersatz : Integer
M-Object-Klasse	*ProduktModell*	-concretizationOfType = <class>*ProduktKategorie* -definedAtObject = <instance>*Produkt* -definedAtLevel = <instance>*Modell*
		-listenpreis : Float

Tabelle 5: Attributwerte beim Anlegen eines Levels (2)

Wird ein neuer Level innerhalb einer Abstraktionsebene eines bestehenden M-Objects, das bereits durch andere M-Objects konkretisiert wird, angelegt so hat dies wesentlich komplexere Auswirkungen auf das bestehende Modell. Für jedes M-Object, das auf einer Ebene eingeführt wurde, die direkt über dem neu eingefügten Level liegt, und ein oder mehrerer Konkretisierungen besitzt, muss ein neues M-Object erstellt werden. Die M-Objects, die eine Konkretisierungsebene unterhalb des neu erstellten M-Objects liegen, konkretisierten somit dieses und nicht mehr ihr ursprüngliches Parent M-Object. Das neu erstellte M-Object ist nun eine direkte Konkretisierung des ursprünglichen Parent M-Objects. Weiters werden für jedes so neu erstellte M-Object auch neue M-Object-Klassen benötigt.

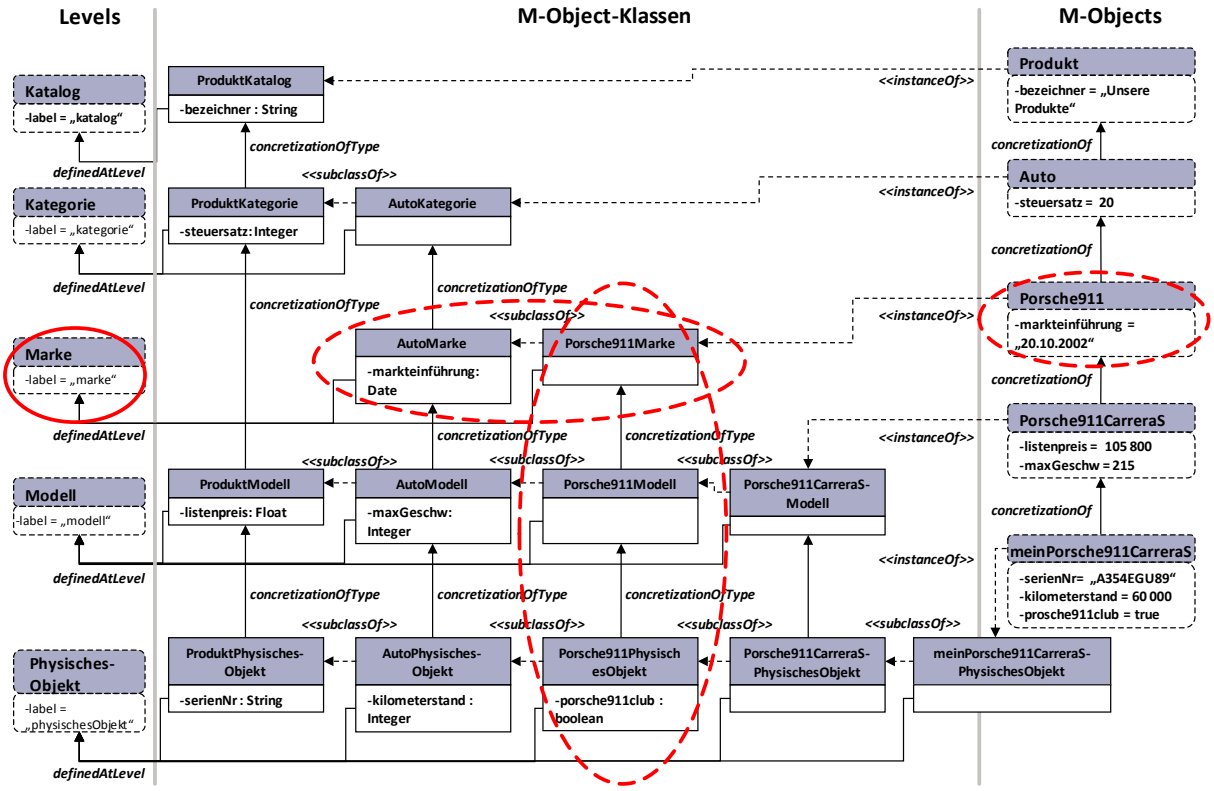

Abbildung 22: Erstellen eines Levels (3)

Das Beispiel in Abbildung 22 zeigt ein Modell mit den M-Objects *Produkt*, *Auto*, *Porsche911CarreraS* und *meinPorsche911CarreraS*, die jeweils aufeinandergefolgt direkte Konkretisierungen darstellen. Wird nun beim M-Object *Auto* ein neuer Level *Marke* eingeführt, so muss auch ein neues M-Object erstellt werden, da sonst die Konkretisierungshierarchie der M-Objects zerstört würde. Dieses neue, automatisch angelegte M-Object wird mit einem „default“-Namen erstellt und ist in diesem Beispiel bereits vom Benutzer in *Porsche911* umbenannt worden. Aus dem neuen M-Object *Porsche911* und dem neuen Level *Marke* ergeben sich dann drei neue M-Object-Klassen. Es sind dies *Porsche911Marke*, *Porsche911Modell* und *Porsche911PhysischesObjekt*. Weiters müssen noch sämtliche Konkretisierungsbeziehungen und Subklassenbeziehungen neu gestaltet werden. Die M-Object-Klasse *AutoModell* bspw. ist nicht länger eine direkte Konkretisierung von der M-Object-Klasse *AutoKategorie* sondern von der M-Object-Klasse *AutoMarke*. Auch das M-Objekt *Porsche911CarreraS* ist keine direkte Konkretisierung von dem M-Object *Auto* mehr. Das M-Object *Auto* wird nunmehr von *Porsche911* konkretisiert.

Für das neue M-Object *Porsche911* werden für jede Abstraktionsebene M-Object-Klassen benötigt. Es sind dies, wie vorher bereits erwähnt, die Klassen *Porsche911Marke*, *Porsche911Modell* und *Porsche911PhysischesObjekt*. Diese sind Subklassen der M-Object-Klassen *AutoMarke*, *AutoModell* und *AutoPhysischesObjekt*, wobei deren ursprüngliche Subklassen *Porsche911CarreraSModell* und *Porsche911CarreraSPhysischesObjekt* nun Subklassen der neu erstellten Klassen darstellen. Da der neue Level *Marke* beim M-Object *Auto* eingeführt wurde, wird auch hierfür eine neue M-Object-Klasse *AutoMarke* benötigt und in die Hierarchie integriert.

Nach dem Erstellen des Levels *Marke* ergeben sich neue Werte für die Attribute der einzelnen Konstrukte, die in Tabelle 6 dargestellt werden.

Konstrukt	**Name**	**Attribute und Werte**
M-Object	*Porsche911*	-definesClass = <class>*Porsche911Marke* = <class>*Porsche911Modell* = <class>*Porsche911PhysischesObjekt* -concretizationOf = <instance>*Auto*
		-markteinführung = <String> 20.10.2002
M-Object-Klasse	*AutoMarke*	-concretizationOfType = <class> *AutoKategorie* -definedAtObject = <instance>*Auto* -definedAtLevel = <instance>*Marke*
		-markteinführung : String
M-Object-Klasse	*Porsche911Marke*	-concretizationOfType = null -definedAtObject = <instance>*Porsche911* -definedAtLevel = <instance>*Marke*
M-Object-Klasse	*Porsche911Modell*	-concretizationOfType = <class>Porsche911Marke -definedAtObject = <instance>*Porsche911* -definedAtLevel = <instance>*Modell*
M-Object-Klasse	*Porsche911PhysischesObject*	-concretizationOfType = <class>*Porsche911Modell* -definedAtObject = <instance>*Porsche911* -definedAtLevel = <instance>*PhysischesObjekt*
		-porsche911club : Boolean

Tabelle 6: Attributwerte beim Anlegen eines Levels (3)

5.1.6 Löschen eines Levels

Wird ein Level gelöscht, so werden auch die M-Objects gelöscht, die diesen Level als Top-Level besitzen. Somit verändert sich die Konkretisierungsbeziehung der M-Objects, die diesen Level in der Abstraktionshierarchie aufweisen. Weiters werden sämtliche M-Object-Klassen gelöscht, die auf diesen Level referenzieren und damit verbunden auch die Attribute, die sie spezifizieren. Dadurch verändern sich die Konkretisierungsbeziehungen und die Subklassenbeziehungen der betroffenen M-Object-Klassen. Ein Level ist immer verbunden mit dem M-Object bei dem er zuerst definiert wurde und kann somit auch nur auf der Ebene dieses M-Objects gelöscht werden.

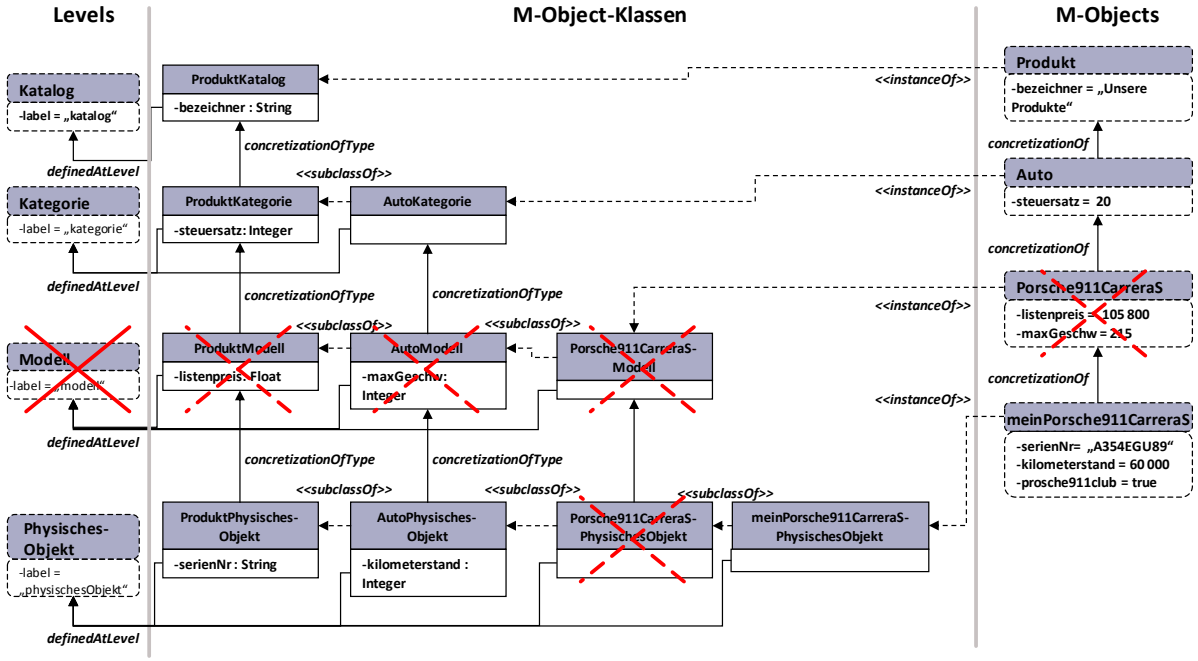

Abbildung 23: Löschen eines Levels

Abbildung 23 zeigt die Auswirkungen auf die Modellstruktur, wenn das Level *Modell* gelöscht wird. Das Löschen kann aber nur auf Produktebene erfolgen, da der Level auch beim M-Object *Produkt* angelegt wurde. Es wird das M-Object *Porsche911CarreraS* entfernt, da das gelöschte Level *Modell* den Top-Level des M-Objects bildet. In weiterer Folge werden auch die M-Object-Klassen *ProduktModell*, *AutoModell* und *Porsche911CarreraSModell* nicht mehr benötigt. Die M-Object-Klasse *Porsche911CarreraSPhysischesObjekt* wird ebenfalls entfernt, da sie durch den Wegfall des Levels *Modell* keine gültige Konkretisierung mehr darstellt.

Auch die in den nicht mehr benötigten M-Object-Klassen angelegten Attribute *listenpreis* und *maxGeschw* finden im Zuge der Löschung dieser Klassen keine Verwendung mehr und werden ebenfalls entfernt. Das M-Object *meinPorsche911CarreraS* ist damit nun die neue direkte Konkretisierung des M-Objects *Auto* und die M-Object-Klasse *meinPorsche911CarreraSPhysischesObjekt* ist eine direkte Subklasse der M-Object-Klasse *AutoPhysischesObjekt*. Weiters müssen die Konkretisierungsbeziehungen der beiden M-Object-Klassen *ProduktPhysischesObjekt* und *AutoPhysischesObjekt* geändert werden. Diese verweisen jetzt mittels des Attributes `concretizationOfType` auf die M-Object-Klassen *ProduktKategorie* bzw. *AutoKategorie*.

Durch das Entfernen des Levels *Modell* werden die in Tabelle 7 dargestellten Elemente gelöscht.

Konstrukt	**Name**
M-Objects	*Porsche911CarreraS*
M-Object-Klassen	*ProduktModell, AutoModell, Porsche911CarreraSModell, Porsche911CarreraSPhysischesObjekt*
Attribute	*listenpreis, maxGeschw, kilometerstand*

Tabelle 7: Löschen eines Levels

5.1.7 Umbenennen eines Levels

Wird ein Level umbenannt, hat das, wie auch beim Umbenennen eines M-Objects, Auswirkungen auf die verschiedenen M-Object-Klassen, da sich der Name einer M-Object-Klasse aus dem Namen des M-Objects und des Levels ableiten lässt. Somit müssen bei jeder Namensänderung eines Levels auch die Namen der betroffenen M-Object-Klassen verändert werden. In weiterer Folge hat die Namensänderung eines Levels auch Auswirkungen auf die ConnectionLevels, deren Bezeichnung sich zum Teil ebenfalls aus dem Namen der Levels zusammensetzt, womit auch diese angepasst werden müssen.

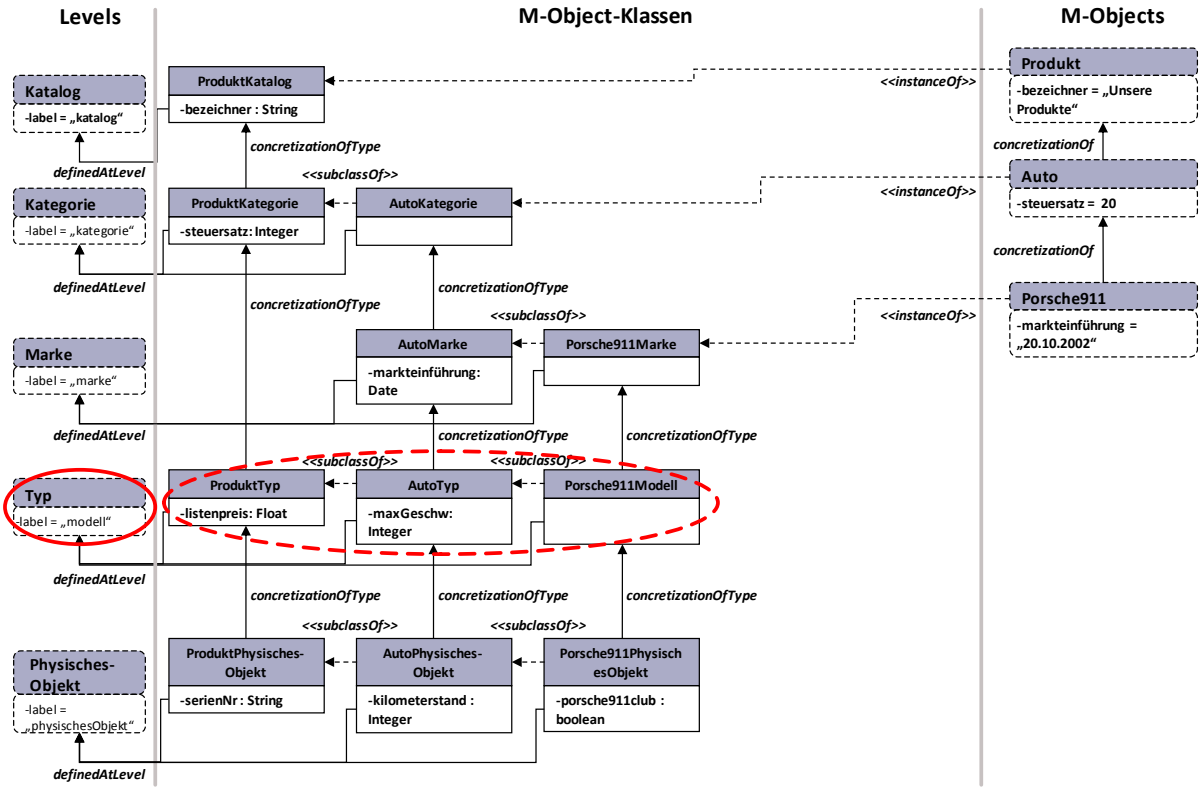

Abbildung 24: Umbenennen eines Levels

In Abbildung 24 wurde der Level *Marke* in *Typ* umbenannt. Aus diesem Grund mussten auch die Namen der M-Object-Klassen *ProduktModell*, *AutoModell* und *Porsche911Modell* in *ProduktTyp*, *AutoTyp* und *Porsche911Typ* geändert werden. Des Weiteren müssen auch ConnectionLevels wie bspw. *Modell-Unternehmen* in *Typ-Unternehmen* umbenannt werden.

5.1.8 Erstellen, Ändern und Löschen eines Attributes

Attribute beschreiben M-Objects und können für jedes M-Object auf jeder Abstraktionsebene also für jede M-Object-Klasse erstellt werden. Ein Wert kann erst bei einem konkreten M-Object vergeben werden. Allerdings kann beim Anlegen eines Attributes auf einer Abstraktionsebene ein Default-Wert gesetzt werden, der den Wert eines Attributes ersetzt, falls diesem keiner zugewiesen wurde.

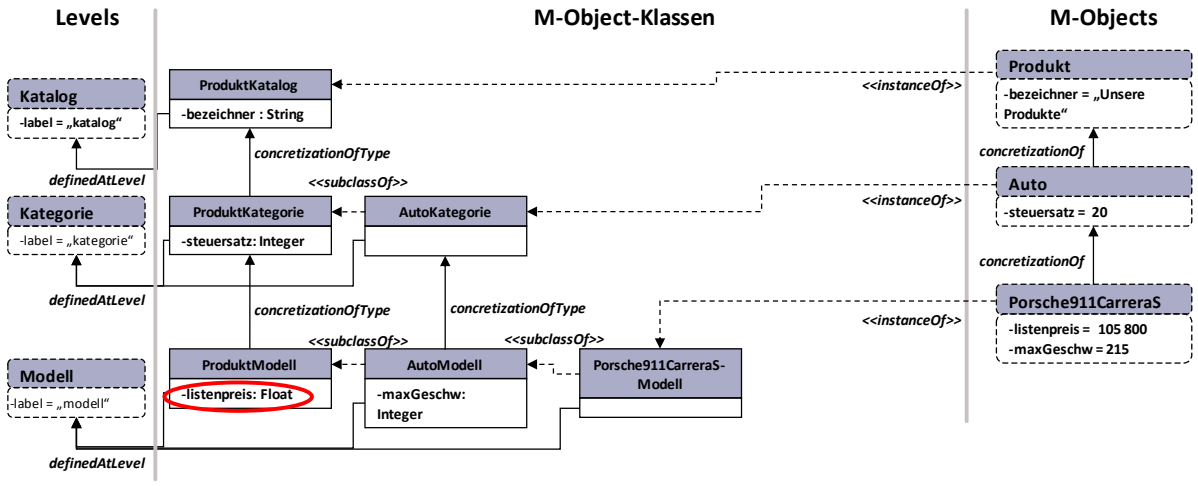

Abbildung 25: Erstellen, Ändern und Löschen von Attributen

Bei der Erstellung des Attributes muss die Bezeichnung und der Wertebereich (Boolean, Float, Integer oder String) angegeben werden. Optional dazu kann auch noch eine Beschreibung zu dem Attribut hinzugefügt und ein Default-Wert gesetzt werden. Diese Attribute können bei jedem M-Object auf jeder Abstraktionsebene eingefügt werden und werden durch die M-Object-Klassen weitervererbt. Betrachtet man in Abbildung 25 das Attribut *listenpreis* so lässt sich erkennen, dass es auf der Ebene *Modell* beim M-Object *Produkt* angelegt wurde. Ersichtlich ist dies in der M-Object-Klasse *ProduktModell*. Die Ausprägung dieses Attributes ist allerdings erst bei dem konkreten M-Object *Porsche911CarreraS* zu sehen, dessen Top-Level das Level *Modell* ist. In diesem Fall ist das Attribut *listenpreis* mit dem Wert „105 800“ belegt.

Die Änderung eines Attributes oder besser gesagt die Änderung des Namens des Attributes oder dessen Wertebereich hat Auswirkungen auf deren Ausprägungen. Wird bspw. der Wertebereich des Attributes *listenpreis* von *Float* auf *Integer* geändert, so wird der bestehende Wert des Attributes für das M-Object gelöscht, da die Typenverträglichkeit zwangsläufig nicht mehr gegeben ist. Ändert sich der Name, die Dokumentation oder der Default-Wert des Attributes werden beide Änderungen die Hierarchie abwärts weitervererbt. Ein Attribut kann nur bei dem M-Object auf der Abstraktionsebene verändert werden, wo es auch erstellt wurde.

Das Löschen eines Attributes hat insofern Auswirkungen auf das Modell, dass wenn das Attribut gelöscht wurde, es auch in keiner Subhierarchie mehr zu finden ist. Wie beim Ändern eines Attributes, kann ein Attribut nur bei dem M-Object auf der Abstraktionsebene gelöscht werden, wo es erstellt wurde.

5.2 Operationen auf M-Relationships

Um M-Relationships erstellen, ändern und löschen zu können bedarf es eine Reihe an Operationen. Solche Operationen verändern die Struktur bereits bestehender M-Relationships nachhaltig und unterliegen gewissen Restriktionen. Im Nachfolgenden werden die einzeln Operation aufgelistet, die für die Bearbeitung von M-Relationships zulässig sind.

- Erstellen von M-Relationships
- Löschen von M-Relationships
- Umbenennen von M-Relationships
- Erstellen, ändern und löschen eines ConnectionLevels

Diese Operationen verändern das Erscheinungsbild der M-Relationships und werden in den nachfolgenden Kapiteln genauer beschrieben. Zum besseren Verständnis dienen dabei Grafiken, die aufbauend auf das durchgängige Beispiel, die Auswirkungen der Operationen auf M-Relationships näher erläutern.

5.2.1 Erstellen von M-Relationships

Wird ein neues M-Relationship erstellt, das noch keine Konkretisierung eines anderen M-Relationships darstellt, so müssen bei der Erstellung durch den Benutzer zwei M-Objects ausgewählt werden, die miteinander in Beziehung stehen sollen. Zusätzlich wird ein Bezeichner benötigt, aus dessen Wert sich gemeinsam mit den Namen der beiden M-Objects ein eindeutiger Name für das neue M-Relationship ableiten lässt. Der Bezeichner wird im Attribut `relationLabel` der M-Relationship festgehalten, die beiden M-Objects bilden die Werte für die Attribute `relationSource` und `relationTarget`. Die

ConnectionLevels werden mittels des Wertes des Attributes `connectionLevels` referenziert.

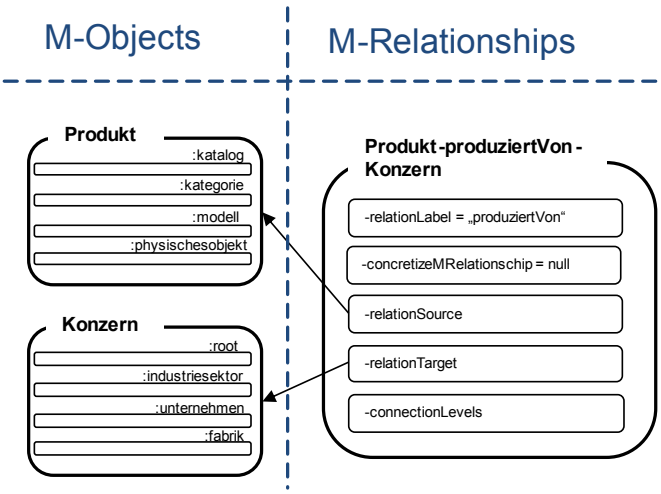

Abbildung 26: Erstellen von M-Relationships (1)

Das Beispiel in Abbildung 26 zeigt das neue M-Relationship *Produkt-produziertVon-Konzern*. Der Name des M-Relationships setzt sich zusammen aus den Namen der M-Objects *Produkt* und *Konzern* in Verbindung mit dem vom Benutzer gewählten Wert für das Attribut `relationLabel` „produziertVon“. Das Attribut `relationSource` referenziert auf das M-Object *Produkt* und das Attribut `relationTarget` auf das M-Object *Konzern*. ConnectionsLevels wurden noch keine angelegt, weshalb das Attribut `connectionLevel` keine Referenz auf einen ConnectionLevel besitzt. Auch das Attribut `concretizeMRelationship` referenziert auf keinen Wert, da das M-Relationship keine Konkretisierung eines anderen M-Relationships darstellt.

Wird ein M-Relationship erstellt, das eine Konkretisierung eines weiteren M-Relationships darstellt, so erbt das neue M-Relationship den Wert des Attributes `relationLabel` und die ConnectionLevels, sofern die M-Objects, die das neue M-Relationship miteinander verbindet, auch die Abstraktionsebenen beinhalten, die mittels dem ConnectionLevel miteinander verknüpft sind. Als Source-M-Object und Target-M-Object sind außerdem nur jene M-Objects erlaubt, die eine direkte oder indirekte Konkretisierung der M-Objects des übergeordneten M-Relationships darstellen. Mittels dem Wert des Attributes

`concretizeMRelationship` wird die Konkretisierungsbeziehung zum übergeordnetem M-Relationship veranschaulicht.

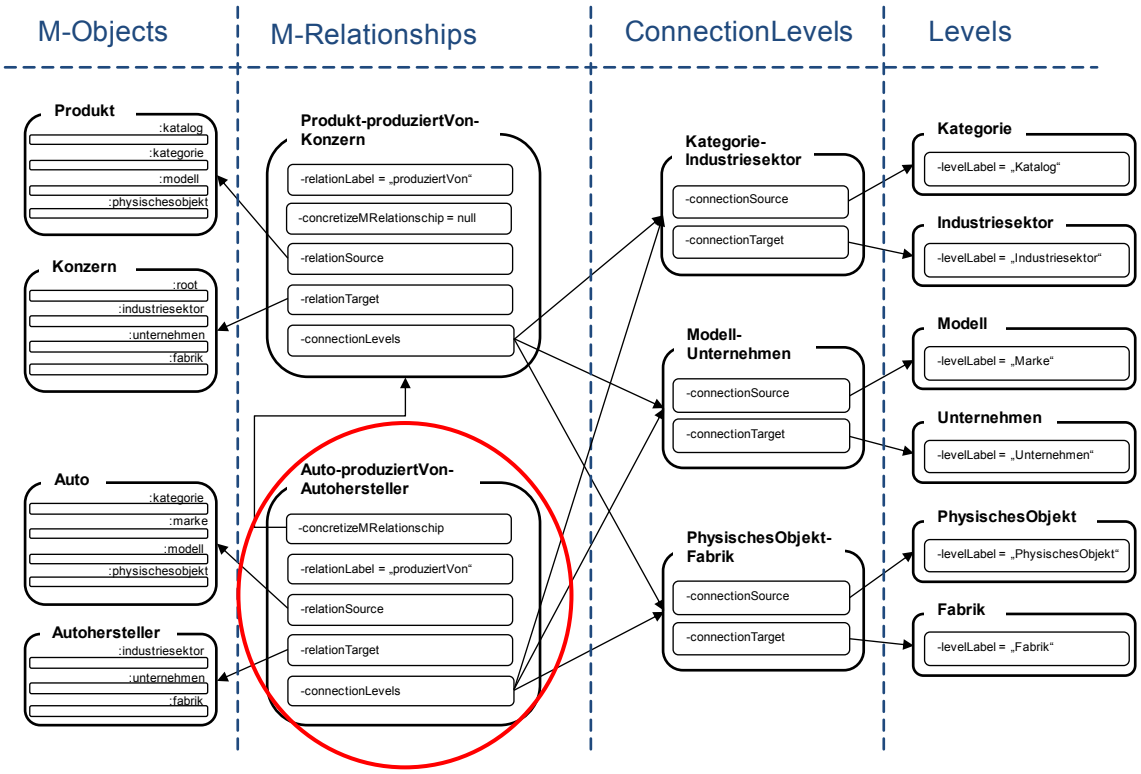

Abbildung 27: Erstellen von M-Relationships (2)

Abbildung 27 zeigt ein Beispiel indem ein neues M-Relationship *Auto-produziertVon-Autohersteller* erstellt wurde. Dieses M-Relationship stellt eine Konkretisierung des M-Relationships *Produkt-produziertVon-Konzern* dar, wobei die Konkretisierungsbeziehung durch eine Referenz des Attributes `concretizeMRelationship` auf das übergeordnete M-Relationship ersichtlich ist. Der Name des neuen M-Relationships setzt sich aus dem M-Object *Auto*, dem M-Object *Autohersteller* und dem Wert des Attributes `relationLabel`, der von dem übergeordneten M-Relationship vererbt wurde, zusammen. Weiters sind das Source-M-Object *Auto* und das Target-M-Object *Autohersteller* direkte bzw. indirekte Konkretisierungen von den M-Objects *Produkt* und *Konzern*, was allerdings in Abbildung 27 nicht ersichtlich ist.

Das neue M-Relationship verweist mittels einer Referenz des Attributes `connectionLevels` auf die ConnectionLevels *Kategorie-Industriesektor*, *Modell-Unternehmen* und *PhysischesObjekt-Fabrik*. Diese ConnectionLevels wurden vom übergeordneten M-Relationship geerbt, was zulässig war, weil sowohl das Source-M-Object *Auto*, die Level *Kategorie*, *Modell* und *PhysischesObjekt*, als auch das Target-M-Object *Autohersteller*, die Level *Industriesektor*, *Unternehmen* und *Fabrik*, beinhalten.

5.2.2 Löschen eines M-Relationships

Wird ein M-Relationship nicht mehr benötigt und somit gelöscht, hat dies Auswirkungen auf die konkretisierenden M-Relationships, die ebenfalls entfernt werden müssen. Auch die ConnectionLevels des aktuellen und des untergeordneten M-Relationships, die bei diesem angelegt wurden, verlieren somit ihre Existenzberechtigung.

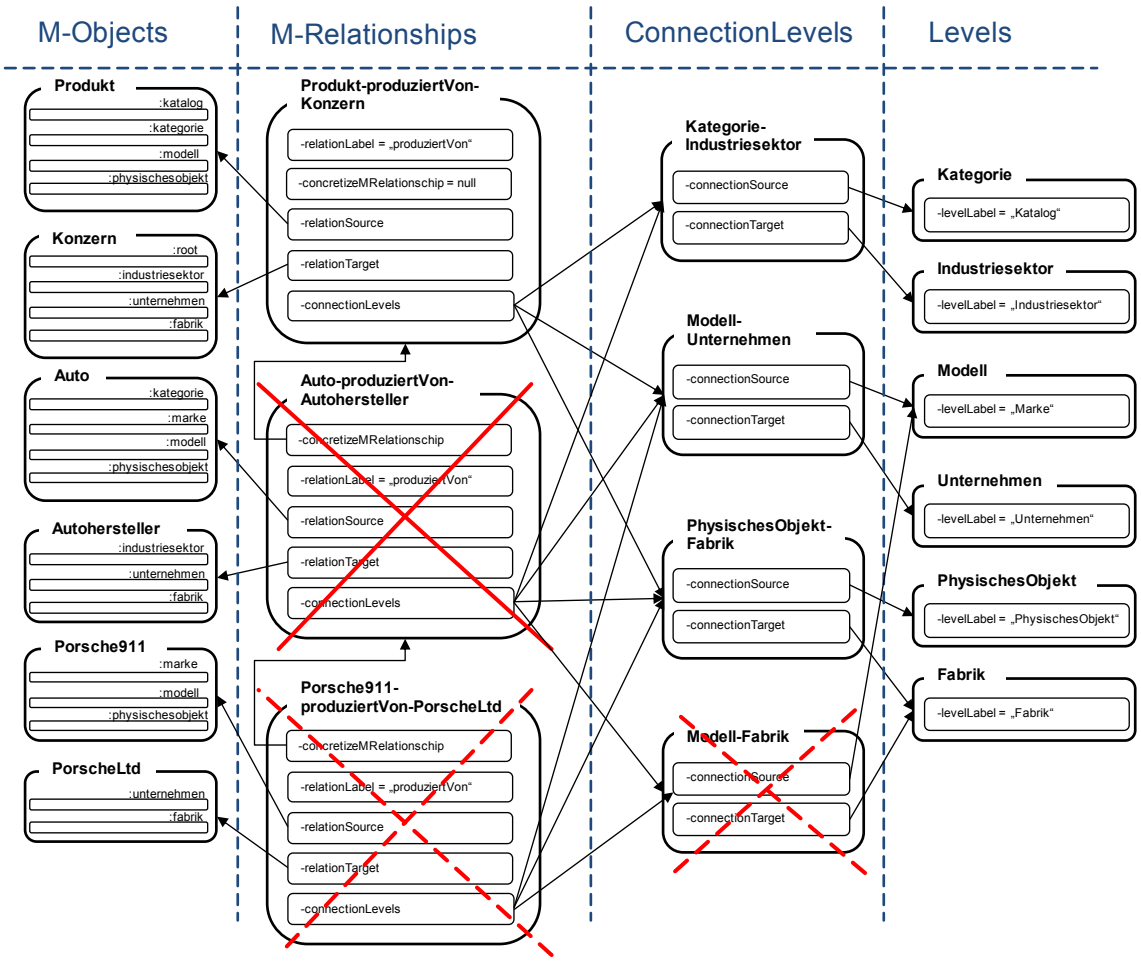

Abbildung 28: Löschen eines M-Relationships

Abbildung 28 zeigt ein Beispiel mit drei M-Relationships nämlich *Produkt-produziertVon-Konzern*, *Auto-produziertVon-Autohersteller* und *Porsche911-produziertVon-PorscheLtd*, die in dieser Reihenfolge direkte Konkretisierungen voneinander darstellen. Wird das M-Relationship *Auto-produziertVon-Autohersteller* gelöscht, so werden auch die direkten und indirekten Konkretisierungen dieses M-Relationships nicht mehr benötigt. In diesem Fall wäre das das M-Relationship *Porsche911-produziertVon-PorscheLdt*. Weiters wurde beim M-Relationship *Auto-produziertVon-Autohersteller* ein neuer ConnectionLevel *Modell-Fabrik* eingeführt, der im Zuge der Löschung des M-Relationships ebenfalls entfernt werden muss.

5.2.3 Umbenennen eines M-Relationships

Der Name eines M-Relationships setzt sich, wie in Kapitel 4.3 beschrieben, aus dem Namen der M-Objects, die es verbindet, und dem Wert des Attributes `relationLabel` zusammen. Beim Umbenennen eines M-Relationships wird lediglich der Wert des Attributes `relationLabel` verändert. Der Benutzer hat somit nur die Möglichkeit einen neuen Bezeichner für das M-Relationship zu vergeben, wobei die durch das M-Relationship verbundenen M-Objects unberührt bleiben. Würden diese geändert, entstünde ein komplett neues M-Relationship, was nicht Sinn der Sache wäre.

Allerdings hat auch die schlichte Änderung des Bezeichners Auswirkungen auf die konkretisierenden M-Relationships. Da diese das Attribut `relationLabel` erben und es selbst für die Namensgebung verwenden, müssen bei einer Änderung des Bezeichners eines übergeordneten M-Relationships auch die Namen der M-Relationships verändert werden, die eine Konkretisierung darstellen.

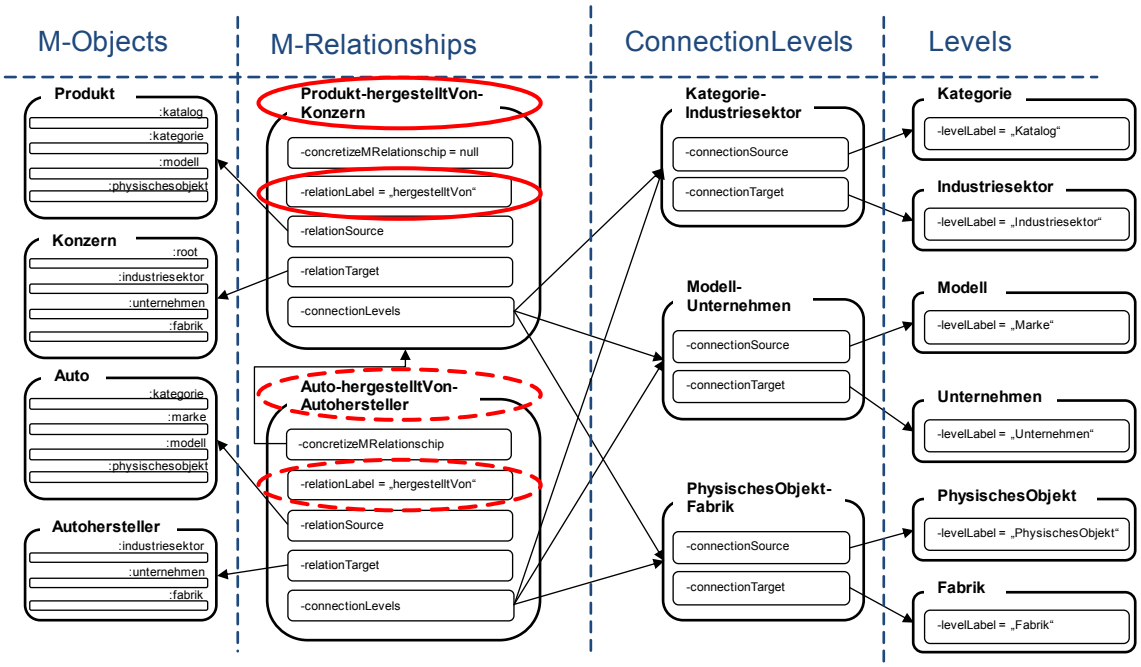

Abbildung 29: Umbenennen von M-Relationships

In Abbildung 29 wurde das M-Relationship *Produkt-produziertVon-Konzern* in *Produkt-hergestelltVon-Konzern* umbenannt. Dabei wurde allerdings nur der Wert des Attributes `relationLabel` neu gesetzt, was zur Namensänderung führte. Verbunden damit wurde auch das konkretisierende M-Relationship *Auto-produziertVon-Autohersteller* in *Auto-hergestelltVon-Autohersteller* umbenannt. Der Wert des Attributes `relationLabel` beider M-Relationships wurde geändert auf „hergestelltVon“.

5.2.4 Erstellen, ändern und löschen eines ConnectionLevels

Beim Erstellen oder Ändern eines ConnectionLevels wird aus den Namen der Abstraktionsebenen, die in Beziehung gesetzt werden, der Name des ConnectionLevels generiert. Die zulässigen Levels für diese Beziehung werden in erster Linie über die M-Objects bestimmt, auf die das M-Relationship referenziert, bei dem der ConnectionLevel angelegt wird. Weiters wird diese Auswahl durch die bereits existierenden ConnectionLevels des M-Relationships beschränkt.

Als Beispiel hierfür wäre der ConnectionLevel *Kategorie-Industriesektor* zu nennen, der beim M-Relationship *Produkt-produziertVon-Konzern* angelegt wurde. Dieses M-Relationship

referenziert auf die beiden M-Objects *Produkt* und *Konzern*, wobei diese die Abstraktionsebenen *Kategorie* und *Industriesektor* aufweisen. Auch existiert kein ConnectionLevel in der Konkretisierungshierarchie des M-Relationships *Produkt-produziertVon-Konzern,* der diese beiden Levels in Beziehung setzt. Somit ist eine Verbindung dieser beiden Abstraktionsebenen möglich.

Das Löschen von ConnectionLevels ist nur möglich auf der Ebene des M-Relationships, bei dem es angelegt wurde. Auswirkungen hat das Löschen eines ConnectionLevels auch auf die konkretisierenden M-Relationships, die den ConnectionLevel geerbt haben, wobei auch bei diesen der ConnectionLevel entfernt wird.

Teil C – Implementierung

6	Implementierung in Protégé-Frames	73
7	Benutzerhandbuch	96

6. Implementierung in Protégé-Frames

In den nachfolgenden Kapiteln wird veranschaulicht, wie bei der Implementierung von M-Objects und M-Relationships in Protégé vorgegangen wurde. Es wird gezeigt, welche Möglichkeiten Protégé zur Erweiterung bietet und welche Entwicklungswerkzeuge verwendet wurden. Weiters wird die tatsächliche Implementierung beschrieben, also der Aufbau des Plug-Ins und schlussendlich wie M-Objects und M-Relationships in Protégé dargestellt werden.

6.1 Vorbereitungen für die Implementierung

Der erste Schritt war der Download des Source Codes von Protégé-Frames 3.5, sowie eine Suche nach einer Möglichkeit Protégé zu erweitern. Dafür kam ein Plug-In in Frage, das im Grunde genommen eine Erweiterung eines Programms darstellt. Protégé bietet dem Entwickler verschiedene Plug-In Schnittstellen zur Erweiterung:

- create project plugin
- project plugin
- back-end plugin
- export plugin
- tab widget plugin
- slot widget plugin

All diese Plug-Ins sind auf der Protégé Homepage unter [Stan15a] sehr gut beschrieben und mit Beispielen hinterlegt, was das Verstehen dieser Plug-Ins wesentlich erleichtert. Aus diesen verschiedenen Möglichkeiten Protégé zu erweitern, fiel die Wahl auf das Tab-Widget-Plug-In, das für die Realisierung von M-Objects und M-Relationships am geeignetsten schien.

Der zweite Schritt war die Einbettung eines Tab-Widget-Beispiels in die verwendete Entwicklungsumgebung Eclipse, die im Anhang noch genauer erläutert wird. Dafür mussten unter anderem zum Klassenpfad die JAR-Dateien „protege.jar“, „looks.jar“ und „unicode_panel.jar“ hinzugefügt werden, sowie das Verzeichnis „meta-inf“ mit der

„manifest-Datei“ in das Ausgabe Verzeichnis kopiert werden. Für genauere Informationen zu diesem Thema wird auf [Stan15a] verwiesen.

Dritter Schritt der Vorbereitung für die Implementierung war die Einbettung des neuen Plug-Ins in Protégé. Das Plug-In war zwar an dieser Stelle bereits lauffähig, wurde aber in Protégé nicht angezeigt. Hierfür musste in Protégé unter dem Menüpunkt „Project/Configure“ das neue Tab-Widget-Plug-In „MObjectMRelationshipTab“ ausgewählt werden, sodass es in Protégé zur Anzeige hinzugefügt und dargestellt werden konnte.

Der vierte Schritt war die Erstellung der benötigten Elemente für das Mapping. Es mussten die Metaklasse `MObjectClass`, die Klasse `MObject`, die Klasse `MRelationship`, sowie die Klassen `MLevel` und `MConnectionLevel` samt ihrer Attribute erstellt werden. Dies geschah anfänglich direkt in Protégé und wurde später in das Plug-In integriert, wobei im Plug-In überprüft wird, ob die Klassen bereits vorhanden sind. Ist dies nicht der Fall werden sie beim Start des Plug-Ins automatisch angelegt.

Fünfter und letzter Schritt war die Überlegung der Darstellung von M-Objects und M-Relationships, also wie das Plug-In aufgebaut werden muss, um eine möglichst übersichtliche, benutzerfreundliche Anzeige und Handhabung der verschiedenen Elemente zu gewährleisten. Dabei fiel die Entscheidung, dass M-Objects und M-Relationships jeweils in einem Browser als Baumstruktur dargestellt werden und deren Abstraktionsebenen und Attribute jeweils in einem Editor in Tabellenformat aufgelistet werden.

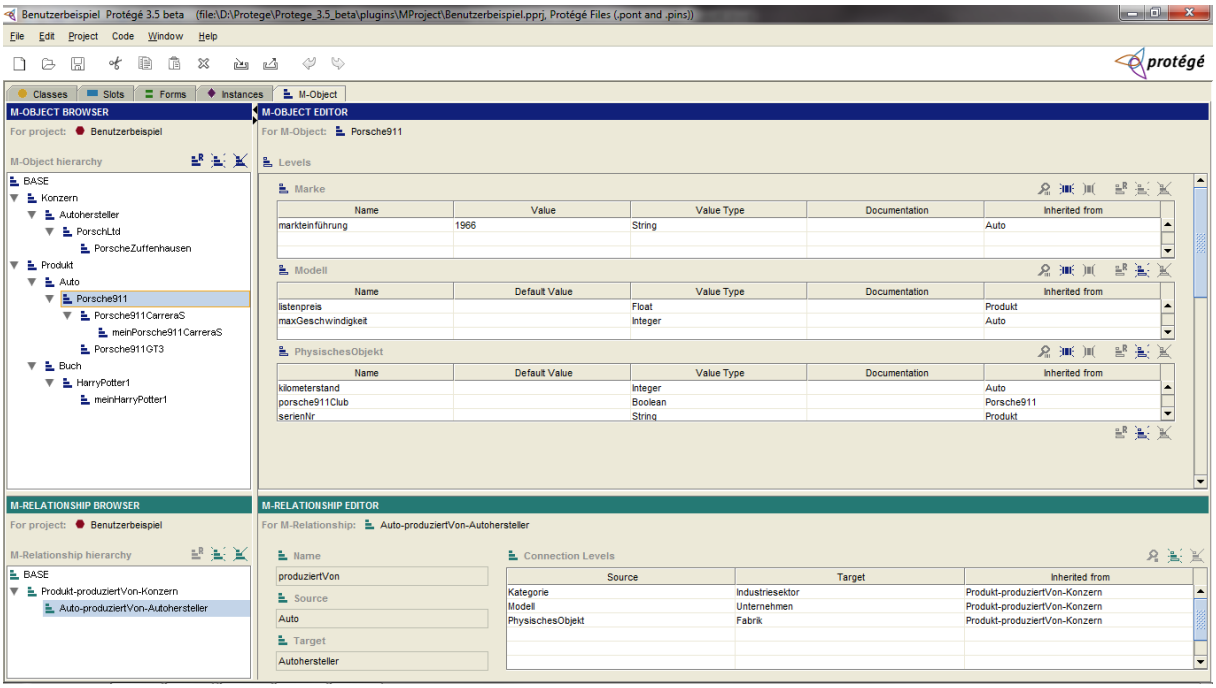

Abbildung 30: Darstellung von M-Objects und M-Relationships in Protégé

Abbildung 30 zeigt die Darstellung von M-Objects und M-Relationships in Protégé. Links oben im Bild zu sehen ist der M-Object Browser, indem die M-Objects linear nach der Konkretisierungshierarchie in einer Baumstruktur angezeigt werden. Im rechten, oberen Teil der Abbildung ist der M-Object Editor zu sehen, indem die einzelnen Abstraktionsebenen des ausgewählten M-Objects inklusive der Attribute dargestellt werden. Der M-Relationship Browser und der M-Relationship Editor sind in der unteren Hälfte der Abbildung zu finden. Wie auch bei den M-Objects zuvor sind auch die sich konkretisierenden M-Relationships in einer Baumstruktur dargestellt. Der M-Relationship Editor zeigt die Attribute des ausgewählten M-Relationships, sowie die eigenen und die geerbten ConnectionLevels.

6.2 M-Object Browser

Der M-Object Browser stellt das Kernelement für die Verwaltung und die Navigation zwischen den einzelnen M-Objects dar. Die M-Objects sind dort, wie bereits erwähnt, in einer Baumstruktur entsprechend der Konkretisierungshierarchie aufgelistet, wobei das konkreteste M-Object ganz unten in der Struktur zu finden ist.

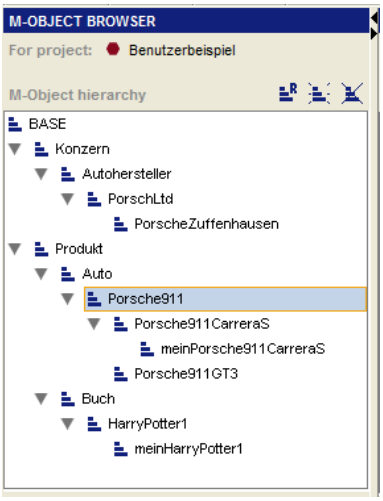

Abbildung 31: M-Object Browser

Der M-Object Browser, dargestellt in Abbildung 31, enthält nach der Bezeichnung „For Project“ den Namen des aktuellen Projektes, das gerade geöffnet ist. Im unteren Teil nach der Bezeichnung „M-Object hierarchy“ werden die M-Objects in einem Baum aufgelistet, deren Wurzelknoten immer der Knoten „BASE“ darstellt. Die Navigation durch die Struktur der M-Objects erfolgt mittels Bedienung durch die Maus oder den Pfeiltasten der Tastatur, wobei die einzelnen Äste der Baumstruktur erweitert oder zusammengeklappt werden können. Operationen auf die M-Objects können über ein Kontextmenü durch Linksklick mit der Maus auf ein ausgewähltes M-Object oder durch Klick auf das entsprechende Symbol in der Toolbar durchgeführt werden. Dabei enthält die Toolbar bzw. das Kontextmenü folgende Operationen, die auf M-Objects durchgeführt werden können:

- Anlegen eines neuen M-Objects (Create M-Object)
- Löschen eines M-Objects (Delete M-Object)
- Umbenennen eines M-Objects (Rename M-Object)

Zusätzlich enthält das Kontextmenü noch den Menüpunkt zum Verschieben von M-Objects (Change parent) und weitere Operationen wie Erweitern (Expand) und Zusammenklappen (Collapse) der Baumstruktur.

Wie erwähnt, können Operationen auf M-Objects im M-Object Browser über das Kontextmenü oder über die Toolbar ausgewählt werden. Das Auswählen dieser Operationen

ist allerdings nur zulässig, wenn sich diese Operationen auf das aktuell ausgewählte M-Object auch ausführen lassen. Ist das nicht der Fall werden diese Operationen für den Benutzer sichtbar auf inaktiv gesetzt, sodass diese Operationen erst gar nicht ausgewählt werden können.

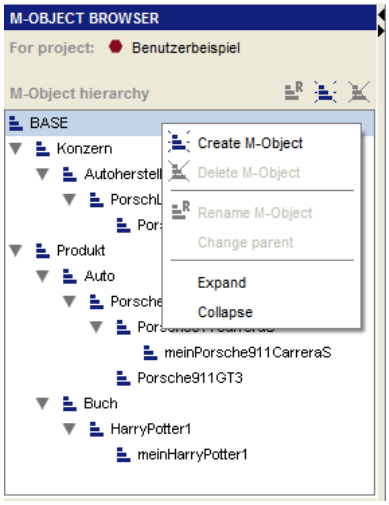

Abbildung 32: Kontextmenü im M-Object Browser

Abbildung 32 zeigt das Kontextmenü für das Objekt „BASE“, wobei einzelne Operationen auf inaktiv gesetzt wurden, da sie auf dieses Objekt nicht erlaubt sind. Somit kann der Benutzer diese Operationen nicht ausführen.

Die Erstellung eines neuen M-Objects durch Klick auf die entsprechende Operation hat zur Folge, dass sich je nach ausgewähltem M-Object ein Dialog öffnet, indem der Benutzer aufgefordert wird, einen Namen für das neue M-Object zu vergeben. Ist für das neue M-Object kein Level also keine Abstraktionsebene vorhanden, auf der es angelegt werden kann, muss auch ein Name für ein neues Level eingegeben werden. Dies ist bspw. der Fall, wenn eine komplett neue Hierarchie angelegt wird und noch kein Level existiert.
Der durch den Benutzer vergebene Name wird direkt bei der Eingabe überprüft, ob nicht bereits ein anderes Objekt diesen Namen trägt. Sollte das der Fall sein, wird der Benutzer darauf aufmerksam gemacht, indem der eingegebene Name sichtlich markiert wird. Die Erstellung eines M-Objects ist erst abgeschlossen, wenn der Benutzer einen systemweit eindeutigen Namen für das neue M-Object eingegeben hat.

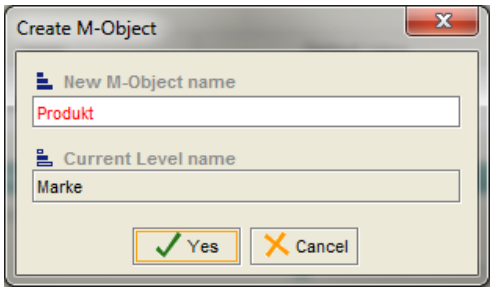

Abbildung 33: Dialog zum Erstellen von M-Objects

In Abbildung 33 wird der Dialog zum Erstellen eines neuen M-Objects gezeigt, wobei der Name „Produkt“ rot markiert ist, da ein M-Object *Produkt* bereits existiert und deshalb als Name für das neue M-Object unzulässig ist.

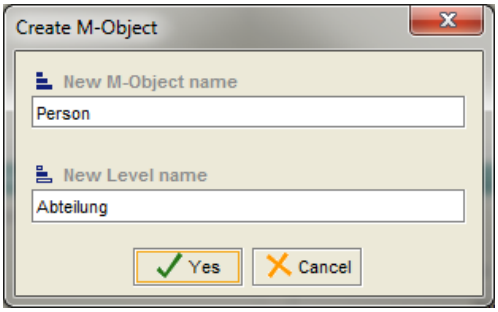

Abbildung 34: Dialog zum Erstellen eines M-Objects und eines Levels

Abbildung 34 stellt einen Dialog zur Erstellung eines neuen M-Objects dar, das noch kein anderes M-Object konkretisiert und auch keine Abstraktionsebenen besitzt. Somit muss im Zuge der Erstellung vom Benutzer auch ein Name für ein neues Level eingegeben werden, für das auch, wie schon bei den M-Objects, kontrolliert wird, ob der eingegebene Name systemweit eindeutig ist.

Beim Umbenennen von M-Objects durch Klick auf die entsprechende Operation erscheint ebenfalls ein Dialog, wo wiederum kontrolliert wird, ob der vom Benutzer eingegebene neue Name auch systemweit eindeutig ist. Ist dies nicht der Fall wird der eingegebene Name rot markiert und die Operation kann erst abgeschlossen werden, wenn der neue Name im System eindeutig ist.

Beim Löschen von M-Objects durch den Benutzer wird dieser darauf aufmerksam gemacht, dass das Löschen eines M-Objects Auswirkungen auf sämtliche Konkretisierungen und die dazugehörigen Klassen hat, die dann ebenfalls gelöscht werden.

Durch Klick auf die Operation Verschieben von M-Objects im Kontextmenü wird ein Dialog geöffnet, der als Auswahl alle möglichen M-Objects bietet, die für diese Operation zulässig sind. Es sind dies alle M-Objects, die dieselbe Abstraktionshierarchie aufweisen, wie das ursprüngliche übergeordnete M-Object.

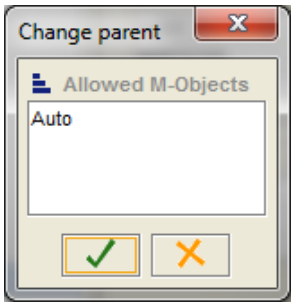

Abbildung 35: Dialog für die Auswahl von übergeordneten M-Objects

6.3 M-Object Editor

Der M-Object Editor dient zur Darstellung der Abstraktionshierarchie des ausgewählten M-Objects. Für jede Abstraktionsebene, des im M-Object Browser ausgewählten M-Objects, existiert eine Tabelle, indem die Attribute auf dieser Ebene aufgelistet sind.

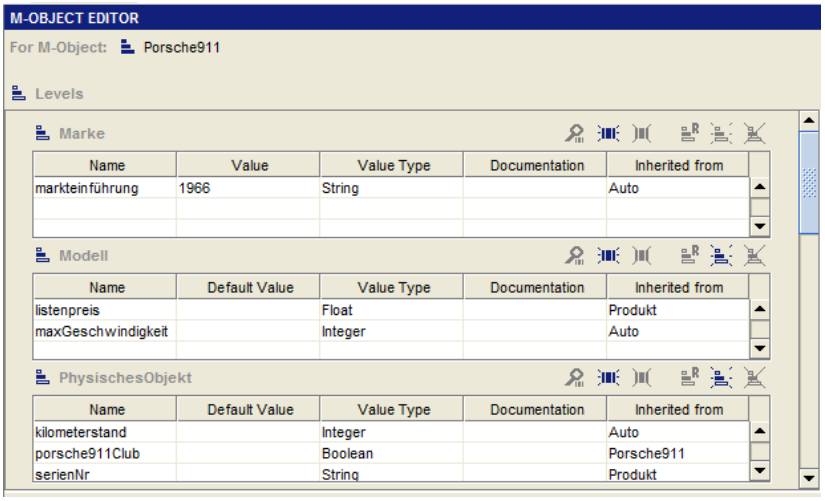

Abbildung 36: M-Object Editor

Abbildung 36 zeigt die Anzeige des M-Object Editors für das M-Object *Porsche911,* ersichtlich im oberen Teil der Grafik nach der Bezeichnung „For M-Object“. Nachfolgend werden die einzelnen Abstraktionsebenen, also die Levels, in Form von Tabellen beschrieben. Jede Ebene wird dabei in einer Tabelle dargestellt, deren Inhalt die angelegten Attribute sind. Im Beispiel existieren Tabellen für die Abstraktionsebenen *Marke*, *Modell* und *PhysischesObjekt* des M-Objects *Porsche911*. Dabei beinhalten die Tabellen folgende Werte:

- Name: Der Name des Attributes.
- Value bzw. Default Value: Der Wert des Attributes bzw. der Default Wert des Attributes.
- Value Type: Der Wertebereich des Attributes.
- Documentation: Eine zusätzliche Beschreibung des Attributes.
- Inherited from: Beinhaltet den Namen des M-Objects, bei dem das Attribut angelegt wurde.

Die Navigation durch die einzelnen Tabellen und den in ihnen enthaltenen Attributen erfolgt mittels Mausführung, wobei durch Doppelklick mit der linken Maustaste auf ein Attribut dessen Werte in einem eigenen Dialog angezeigt werden. Wird ein Doppelklick auf eine leere Zeile innerhalb einer Tabelle durchgeführt, erscheint ebenfalls ein Dialog zum Erstellen eines neuen Attributes. Generell lassen sich Operationen auf Attribute auch durch ein Kontextmenü oder durch Auswahl der gewünschten Operation in der Toolbar aufrufen. Das Kontextmenü wird aktiviert durch einen Linksklick auf ein ausgewähltes Attribut und die Toolbar mit den Operationen ist über der Tabelle, in dem das Attribut enthalten ist, positioniert. Es können folgende Operationen für die Attribute aufgerufen werden:

- Anlegen eines neuen Attributes (Create attribute)
- Löschen eines Attributes (Delete attribute)
- Anschauen bzw. Ändern eines Attributes (View attribute)

Weiters besteht im M-Object Editor die Möglichkeit neue Abstraktionsebenen für das im M-Object Browser ausgewählte M-Object einzuführen. Dafür steht ebenfalls eine Toolbar über der Tabelle des Levels zur Verfügung, wo die neue Ebene eingeführt werden soll. Die Toolbar enthält folgende Operationen für die Manipulation von Levels:

- Erstellen eines neuen Levels (Create new Level)
- Löschen eines Levels (Delete Level)
- Umbenennen eines Levels (Rename Level)

Über das Kontextmenü oder die Toolbar können Operationen auf die Attribute aufgerufen werden, wobei diese nur zugelassen werden, sofern sie auch auf das ausgewählte Attribut erlaubt sind. Ist das nicht der Fall wird sowohl im Kontextmenü als auch in der Toolbar die entsprechende Operation auf inaktiv gesetzt, sodass sie vom Benutzer nicht betätigt werden kann. Dies ist bspw. dann der Fall, wenn wie Abbildung 37 zeigt, ein Attribut gelöscht werden soll, das allerdings auf einem anderen M-Object angelegt wurde, als das derzeit selektierte M-Object darstellt. In diesem Fall ist die Operation für das Löschen dieses Attributes für den Benutzer nicht auswählbar.

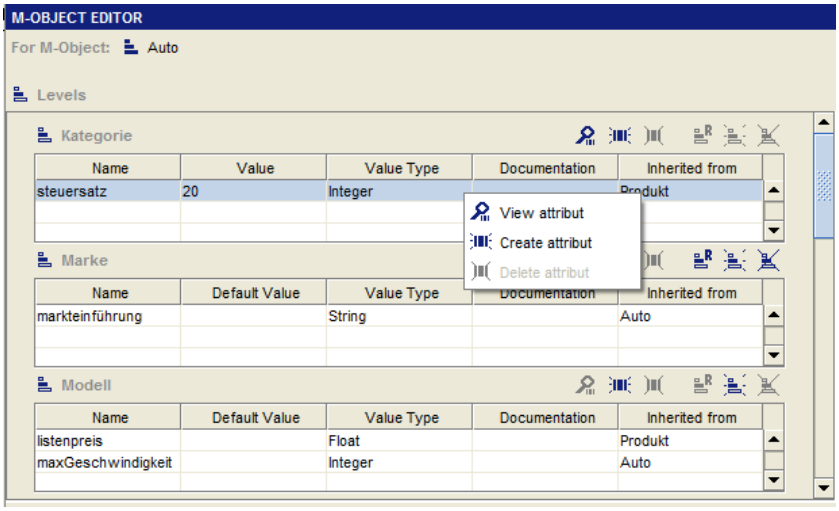

Abbildung 37: Kontextmenü für Attribute im M-Object Editor

Im konkreten Beispiel wurde das Attribut *steuersatz* auf der Abstraktionsebene *Kategorie* ausgewählt, dass wie in der Tabelle ersichtlich, beim M-Object *Produkt* erstellt wurde. Somit ist es nicht zulässig, dass auf dem Level *Kategorie* beim aktuell gewählten M-Object *Auto,* das Löschen dieses Attributes möglich ist.

Beim Erstellen eines neuen Attributes wird ein Dialog geöffnet, indem je nach der Abstraktionsebene auf der das Attribut erstellt werden soll, unterschiedliche Möglichkeiten

zur Eingabe für den Benutzer bereitstehen. Ist das neu zu erstellende Attribut auf dem Top-Level des ausgewählten M-Objects angesiedelt, so wird dem Benutzer gestattet, abgesehen von dem Namen des neuen Attributes, auch den Wertebereich sowie einen konkreten Wert und eine zusätzliche Beschreibung einzugeben. Wird hingegen ein neues Attribut für ein anderes, in der Abstraktionshierarchie weiter unten liegendes Level, erstellt, so beschränkt sich die Eingabe auf den Namen, den Wertebereich, den Default-Wert und die Beschreibung. Der konkrete Wert des Attributes kann in diesem Fall nicht eingegeben werden.

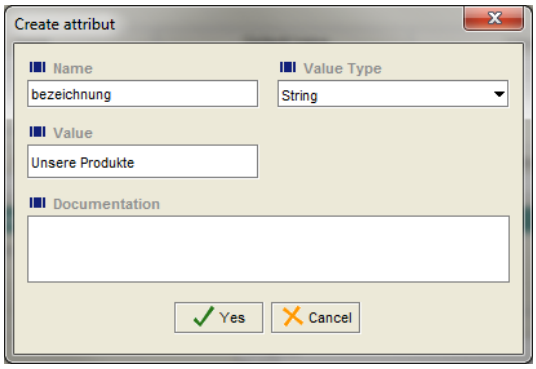

Abbildung 38: Dialog zur Eingabe eines neuen Attributes

In Abbildung 38 ist der Dialog zur Eingabe eines neuen Attributes auf dem Top-Level eines M-Objects zu sehen, wobei der Benutzer den Namen, den Wertebereich, den Wert und eine Beschreibung eingeben kann. Name und Wertebereich sind verpflichtend für die Erstellung des Attributes, wohingegen der Wert und die Beschreibung optional eingegeben werden können. Wird kein Wert eingeben so tritt der Default-Wert in Erscheinung, sofern einer gesetzt wurde.

Da es im Modell auf den verschiedenen Abstraktionsebenen der unzähligen Konkretisierungshierarchien der M-Objects vorkommen kann, dass ein Attribut die selbe Bezeichnung hat, muss sichergestellt sein, dass das neue Attribut systemweit eindeutig referenziert werden kann. Somit setzt sich der systemweit eindeutige Name aus dem Wert des Feldes Name und einer automatisch kreierten Zeichenkette zusammen, die systemweite Eindeutigkeit garantiert. Dieser eindeutige Name ist allerdings für den Benutzer nicht sichtbar und auch nicht von Bedeutung für ihn.

Das Ansehen bzw. Ändern von Attributen erfolgt über den selben Dialog, wie das Erstellen, wobei auch hier je nach ausgewählten M-Object und Abstraktionsebene, auf dem das Attribut ansässig ist, für den Benutzer die Eingabe der unterschiedlichen Werte erlaubt ist oder nicht. Sollte die Eingabe nicht gestattet sein, wird das entsprechende Feld auf inaktiv gesetzt, sodass es nicht vom Benutzer bearbeitet werden kann.

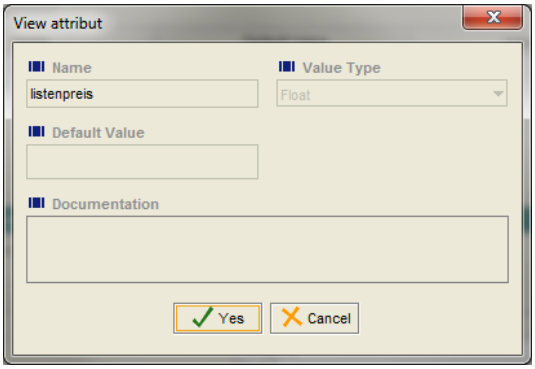

Abbildung 39: Dialog zum Ändern eines Attributes

Sämtliche Felder des in Abbildung 39 gezeigten Dialogs sind auf inaktiv gesetzt und somit für den Benutzer nicht zugänglich, da in diesem Fall das Attribut auf einer Abstraktionsebene eines M-Objects betrachtet wird, wo keine Rechte bestehen, es zu verändern.

Das Löschen eines Attributes ist nur zulässig, wenn das Attribut auch auf dem Level des ausgewählten M-Objects erstellt wurde. Andernfalls ist, wie bereits erwähnt, die Auswahl der Operation Löschen sowohl im Kontextmenü als auch im Toolbarbereich nicht möglich. Wird allerdings ein Attribut gelöscht so erscheint ein einfach gestrickter Dialog, indem der Benutzer nochmals gefragt wird, ob er das Attribut wirklich löschen will.

Operationen auf Level können nur über die Toolbar, die oberhalb jeder Tabelle der einzelnen Abstraktionsebenen angesiedelt ist, aufgerufen werden. Wiederum sind je nach existierendem Level und dem ausgewählten M-Object nur gewisse Operationen erlaubt und für den Benutzer zugänglich. Bspw. ist die Operation zum Löschen eines Levels nur aktiv, wenn er auch bei dem selektierten M-Object erstellt wurde.

Das Erstellen eines neuen Levels hat zur Folge, dass ein Dialog für die Eingabe von Daten erscheint, in dem der neue Name für das Level eingegeben werden kann. Dieser eingegebene Name wird in Verbindung mit einer systemweit eindeutigen Zeichenkette verknüpft und somit entsteht in weiterer Folge ein systemweit eindeutiger Name für den neuen Level. Dies ist notwendig, da in verschiedenen Konkretisierungshierarchien der M-Objects, Abstraktionsebenen mit gleichen Namen angelegt werden können. Allerdings wird beim Erstellen des neuen Levels sichergestellt, dass innerhalb der konkreten Abstraktionshierarchie kein Name für einen Level vergeben werden kann, der bereits in dieser Hierarchie existiert. Ist dies der Fall, wird dieser Umstand den Benutzer sichtbar gemacht und es ist nicht möglich den neuen Level anzulegen.

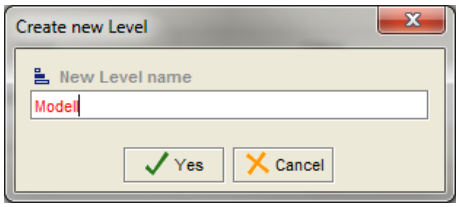

Abbildung 40: Dialog zum Erstellen eines neuen Levels

In Abbildung 40 wird der Dialog zum Erstellen eines neuen Levels beschrieben, wobei in diesem Beispiel versucht wird ein Level *Modell* einzugeben, das aber bereits in der Abstraktionshierarchie vorhanden ist. Aus diesem Grund ist der Name rot eingefärbt und der Benutzer kann den neuen Level mit diesem Namen nicht anlegen.

Wird ein neuer Level innerhalb einer Abstraktionshierarchie angelegt, wobei für die zukünftig darunter liegenden Ebenen M-Objects vorhanden sind, so müssen automatisch für diese Ebene M-Objects angelegt werden. Konkret hat eine solche Erstellung eines neuen Levels Auswirkungen auf den M-Object Browser, indem die neu erstellten M-Objects an der richtigen Position der Konkretisierungshierarchie eingefügt werden müssen.

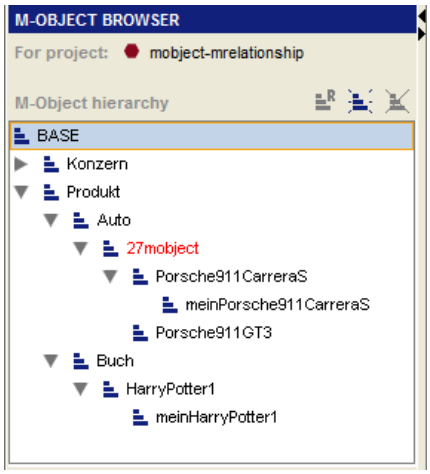

Abbildung 41: Auswirkung des Erstellens eines neuen Levels auf den M-Object Browser

Im Beispiel, das in Abbildung 41 gezeigt wird, wurde beim M-Object *Auto* zwischen den Abstraktionsebenen *Kategorie* und *Modell* ein neuer Level *Marke* eingeführt. Dies hatte zur Folge, dass ein neues M-Object mit dem Top-Level *Marke* erstellt werden musste. In diesem Beispiel war es nur ein M-Object, das automatisch erstellt wurde und für das ein eindeutiger Name generiert werden musste (*27mobject*). Der Name des automatisch erstellten M-Objects ist durch eine rote Farbkennzeichnung gekennzeichnet, sodass der Benutzer diese neuen M-Objects leichter erkennt und gegebenenfalls umbenennen kann. Wird das M-Object umbenannt verschwindet die Kennzeichnung.

Für das Löschen eines Levels wird wiederum ein Dialog aufgerufen, der dem Benutzer dahingehend informiert, dass mit dem Löschen einer bestimmten Abstraktionsebene auch M-Objects verloren gehen, die auf diesem Level angelegt wurden.

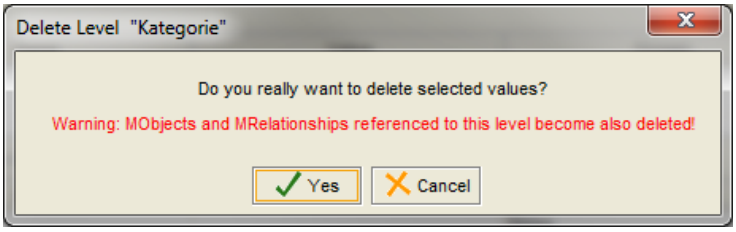

Abbildung 42: Dialog zum Löschen eines Levels

Weitere Auswirkungen des Löschens eines Levels zeichnen sich auch im M-Object Browser ab, sofern für diese Abstraktionsebene M-Objects existieren. In diesem Fall müssen auch die

M-Objects dieser Ebene gelöscht werden und es entsteht eine neue Konkretisierungshierarchie. Die M-Objects, die auf der Abstraktionsebene unterhalb des gelöschten Levels liegen, konkretisieren nun direkt das M-Object, das zuvor von dem M-Object konkretisiert wurde, das durch die Löschung der Abstraktionsebene verloren ging. Dies hat natürlich auch Auswirkungen auf den M-Object Browser, der somit die neue Konkretisierungshierarchie darstellen muss.

Das Umbenennen eines Levels erfolgt in einem ähnlichen Dialog wie das Erstellen eines Levels, wobei an dieser Stelle vom Benutzer nur der Name des Levels geändert wird. Allerdings wird auch, wie schon bereits beim Erstellen, genau überprüft, ob der neue Name des Levels in der Konkretisierungshierarchie des M-Objects und dessen übergeordneten M-Objects nicht auftaucht.

6.4 M-Relationship Browser

Die Darstellung der M-Relationships und die Navigation zwischen den einzelnen M-Relationships wird durch den M-Relationship Editor ermöglicht. Wie bereits beschrieben, sind die M-Relationships in einer Baumstruktur entsprechend ihrer Konkretisierungshierarchie dargestellt.

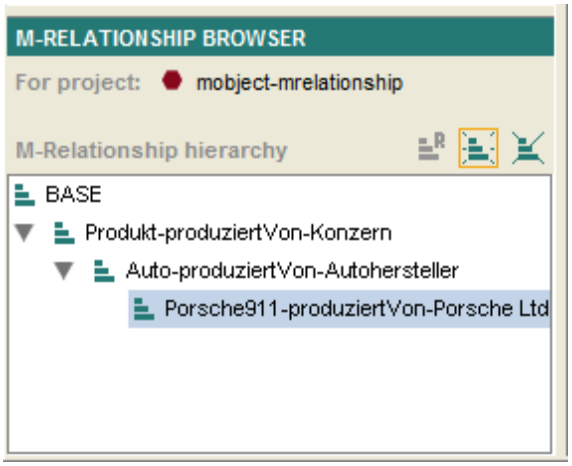

Abbildung 43: M-Relationship Browser

Abbildung 43 zeigt den M-Relationship Browser, der als oberstes Element nach dem Bezeichner „For project“ das aktuell geöffnete Projekt anzeigt. Direkt darunter befinden sich die M-Relationships in einer geordneten Baumstruktur, beginnend vom abstraktesten bis hin zum konkretesten M-Relationship. Der Wurzelknoten dieser Struktur ist „BASE“. Weiters steht eine Toolbar für Operationen auf die M-Relationships zur Verfügung, die rechts oberhalb der Baumstruktur angeheftet ist. Die Navigation durch den Baum erfolgt, wie auch schon im M-Object Browser, mittels Maus oder Pfeiltasten, wobei die Baumstruktur damit erweitert oder zusammengeklappt werden kann. Operationen auf die M-Relationships können durch Auswahl im Kontextmenü, das durch Rechtsklick mit der Maus auf ein M-Relationship erscheint, oder durch Betätigung der entsprechenden Operation in der Toolbar aufgerufen werden. Je nachdem welche Operationen für das ausgewählte M-Relationship erlaubt sind, wird die Auswahl für den Benutzer eingeschränkt, indem die nicht zulässigen Operationen auf inaktiv gesetzt werden. Solche Operationen, die über das Kontextmenü oder die Toolbar aufgerufen werden können, sind:

- Anlegen eines neuen M-Relationships (Create M-Relationship)
- Löschen eines M-Relationships (Delete M-Relationship)
- Umbenennen eines M-Relationships (Rename M-Relationship)

Weiters beinhaltet das Kontextmenü noch zwei zusätzliche Menüpunkte zum Erweitern(Expand) und Zusammenklappen(Collapse) der Baumstruktur, in der die M-Relationships dargestellt sind.

Operationen im M-Relationship Browser können über das Kontextmenü des ausgewählten M-Relationships oder über die Toolbar getätigt werden. Eine Operation kann aber nur gewählt werden wenn, wie bereits beschreiben, diese auch zulässig und somit aktiv ist.

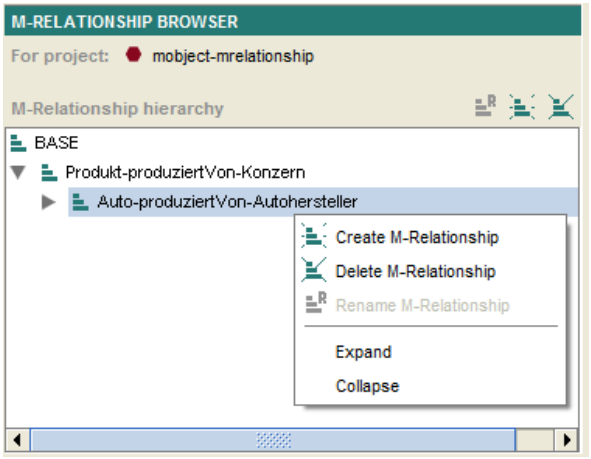

Abbildung 44: Kontextmenü im M-Relationship Browser

Im Kontextmenü des M-Relationship Browsers in Abbildung 44 ist für das gewählte M-Relationship *Auto-produziertVon-Autohersteller* das Umbenennen nicht erlaubt, da die Beziehung *produziertVon* von einem übergeordnetem M-Relationship geerbt wurde und diese auch nur bei diesem M-Relationship geändert werden kann.

Die Erstellung eines neuen M-Relationships hat zur Folge, dass sich ein entsprechender Dialog zur Eingabe der Daten für das neue M-Relationship öffnet. Bei der Eingabe der Daten durch den Benutzer muss allerdings unterschieden werden, ob es sich bei dem zu erstellenden M-Relationship um ein konkretisierendes M-Relationship handelt oder um eines, das noch keine Konkretisierung eines anderen M-Relationships darstellt. Konkretisiert das neue M-Relationship bereits ein weiteres, so sind die Bezeichnung sowie die möglichen M-Objects, die das M-Relationship miteinander verbinden kann, fest vorgegeben. Die Auswahl der M-Objects, die in Verbindung gebracht werden können, beschränkt sich dabei auf die M-Objects des übergeordneten M-Relationships und auf die Konkretisierungen dieser M-Objects.

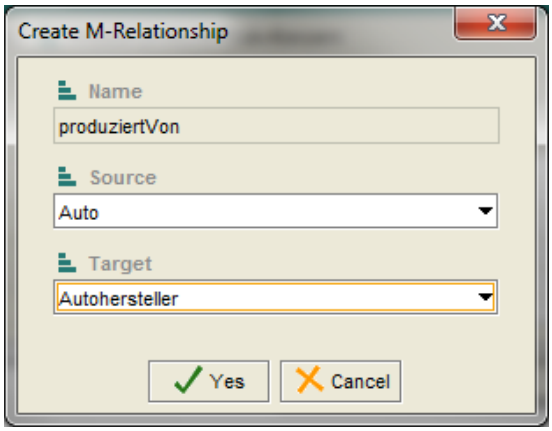

Abbildung 45: Dialog zum Erstellen eines M-Relationships

Abbildung 45 zeigt das Erstellen des M-Relationships *Auto-produziertVon-Autohersteller,* wobei der Bezeichner des M-Relationships *produziertVon* bereits fest vorgegeben ist, da er vom übergeordneten M-Relationship *Produkt-produziertVon-Konzern* geerbt wurde und deshalb vom Benutzer nicht mehr geändert werden kann. Die Auswahl des Source M-Objects beschränkt sich dabei auf das M-Object *Produkt* inklusive aller direkten oder indirekten Konkretisierungen (bspw. *Auto*, *Buch*, *Porsche911*, usw.). Analog dazu können als Target M-Object das M-Object *Konzern* und dessen Konkretisierungen gewählt werden (bspw. *Autohersteller*, *PorscheLtd*, *PorscheZuffenhausen*).

Der systemweit eindeutige Name des neuen M-Relationships setzt sich zusammen aus dem Source M-Object, dem Bezeichner der Verbindung und dem Target M-Object. Es wird auch bei der Erstellung sichergestellt, dass das gleiche M-Relationship, also ein M-Relationship mit dem gleichen Bezeichner, dem gleichen Source M-Object und dem gleichen Target M-Object nicht vom Benutzer ausgewählt werden kann.

Beim Löschen des M-Relationships wird der Benutzer durch einen Dialog darauf aufmerksam gemacht, dass mit dem Löschen auch verbunden, sämtliche konkretisierenden M-Relationships ebenfalls entfernt werden. Weiters sind auch die ConnectionLevels, die beim M-Relationship und dessen Konkretisierungen angelegt wurden, ebenfalls von der Löschung betroffen.

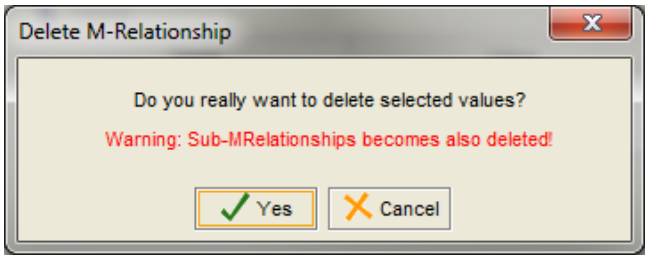

Abbildung 46: Dialog zum Löschen eines M-Relationships

Beim Umbenennen eines M-Relationships kann in einem Dialog lediglich nur der Bezeichner also im Beispiel *produziertVon* geändert werden, wobei dies auch nur bei dem M-Relationship geschehen kann, wo er eingeführt wurde. Wiederum wird bei der neuen Eingabe auf die systemweite Eindeutigkeit Rücksicht genommen. Im Zuge der Änderung des Bezeichners muss der Name aller konkretisierenden M-Relationships ebenfalls geändert werden.

6.5 M-Relationship Editor

Der M-Relationship Editor dient im Wesentlichen zur Darstellung der Attribute des M-Relationships und dessen ConnectionLevels, die dieses M-Relationship geerbt hat oder bei dem diese angelegt wurden.

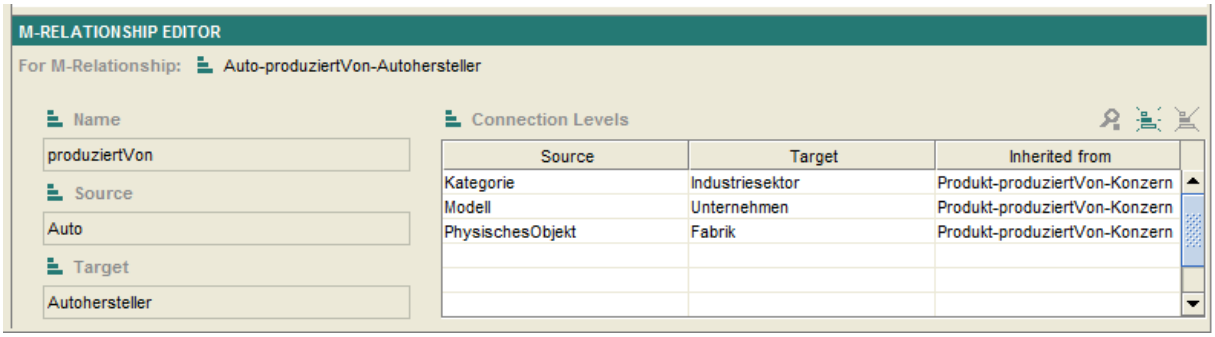

Abbildung 47: M-Relationship Editor

In Abbildung 47 wird der M-Relationship Editor dargestellt, wobei ganz oben nach dem Bezeichner „For M-Relationship“ der Name des aktuell ausgewählten M-Relationships zu sehen ist (*Auto-produziertVon-Autohersteller*). Darunter positioniert sind der Bezeichner, das

Source M-Object und das Target M-Object. Im linken Teil werden die einzelnen ConnectionLevels, die das ausgewählte M-Relationship beinhaltet, in Form einer Tabelle dargestellt. Die Spalten der Tabelle enthalten folgende Werte:

- Source: Beinhaltet als Wert den Source Level.
- Target: Beinhaltet als Wert den Target Level.
- Inherited from: Beinhaltet den Namen des M-Relationships, bei dem das ConnectionLevel angelegt wurde.

Die Navigation im M-Relationship Editor erfolgt mittels Mausführung, wobei durch Doppelklick mit der linken Maustaste auf ein ConnectionLevel dessen Werte in einem eigenen Dialog angezeigt werden. Wird ein Doppelklick auf eine leere Zeile innerhalb der Tabelle, indem die ConnectionLevels aufgelistet sind, durchgeführt, erscheint ein Dialog zum Erstellen eines neuen ConnectionLevels. Wie schon im M-Object Editor, lassen sich auch hier Operationen auf ConnectionLevels durch ein Kontextmenü oder durch Auswahl der gewünschten Operation in der Toolbar aufrufen. Mittels Linksklick auf einen ConnectionLevel wird das Kontextmenü geöffnet, wobei dieselben Operationen auch über eine Toolbar oberhalb der Tabelle aufgerufen werden können. Es können folgende Operationen für einen ConnectionLevel aufgerufen werden:

- Anlegen eines neuen ConnectionLevels (Create ConnectionLevel)
- Löschen eines ConnectionLevels (Delete ConnectionLevel)
- Anschauen bzw. Ändern eines ConnectionLevels (View ConnectionLevel)

Operationen auf ConnectionLevels können im M-Relationship Editor wiederum über das Kontextmenü oder die Toolbar aufgerufen werden, wobei dies nur möglich ist, sofern sie auch auf dem bestehenden ConnectionLevel erlaubt sind. Ist das nicht der Fall wird sowohl im Kontextmenü als auch in der Toolbar die entsprechende Operation auf inaktiv gesetzt.

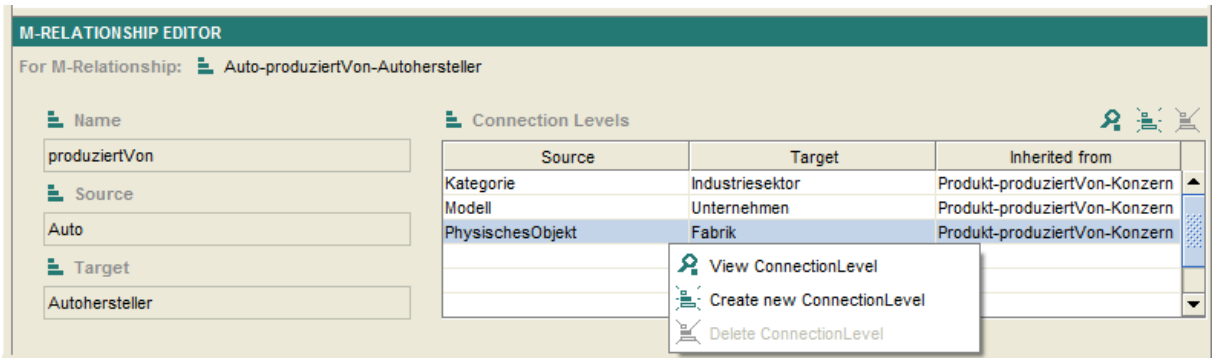

Abbildung 48: Kontextmenü für ConnectionLevels im M-Relationship Editor

Im vorliegenden Beispiel, das in Abbildung 48 dargestellt ist, kann das Löschen des selektieren ConnectionLevels nicht ausgewählt werden, da das ConnectionLevel von einem übergeordnetem M-Relationship geerbt wurde und deshalb auch nur bei diesem M-Relationship gelöscht werden kann.

Das Anlegen eines neuen ConnectionLevels erfolgt mittels eines Dialoges, der dem Benutzer die möglichen Levels für den Source-Level und den Target-Level vorgibt. Der systemweit eindeutige Name ergibt sich wiederum aus einer Zeichenkette und den beiden Level, die miteinander in Beziehung gebracht werden. Welche Level für die Erstellung zur Auswahl stehen ist abhängig von den vorhandenen M-Relationships, den beteiligten M-Objects und deren Abstraktionsebenen und den bereits bestehenden ConnectionLevels.

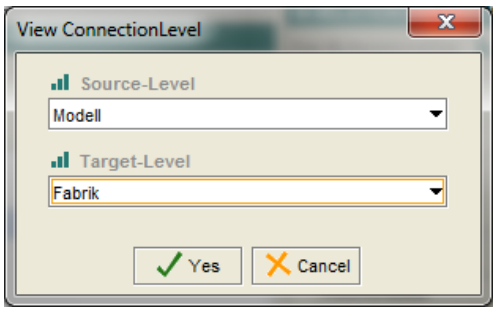

Abbildung 49: Dialog zum Erstellen eines neuen ConnectionLevels

In Abbildung 49 wird ein neuer ConnectionLevel mit dem Source-Level *Modell* und dem Target-Level *Fabrik* erstellt. Dies ist zulässig, da erstens noch kein ConnectionLevel in der

Konkretisierungshierarchie des aktuellen M-Relationships angelegt wurde, das den Level *Modell* und den Level *Fabrik* in Verbindung setzt. Zweitens beinhalten die M-Objects, die durch das M-Relationship miteinander verbunden sind, sowohl das Level *Modell* als auch das Level *Fabrik*. Schlussendlich kreuzen drittends die neu miteinander verbundenen Ebenen keine andere Verbindung, die durch ein weiteres ConnectionLevel in der Hierarchie repräsentiert wird. All diese Restriktionen werden berücksichtigt und dem Benutzer nur jene Levels als Source- bzw. Target-Levels zur Verfügung gestellt, die auch wirklich erlaubt sind.

Das Löschen eines ConnectionLevels eines M-Relationships hat zur Folge, dass auch die konkretisierenden M-Relationships, diesen ConnectionLevel, falls er vererbt wurde, nicht mehr besitzen. Ein ConnectionLevel kann nur bei dem M-Relationship gelöscht werden, wo er auch angelegt wurde. Durch einen kleinen Dialog wird der Benutzer noch einmal explizit gefragt, ob er den ConnectionLevel wirklich löschen will.

Das Ansehen bzw. Ändern von ConnectionLevels erfolgt über denselben Dialog wie das Erstellen, wobei ändern und löschen nur dann zulässig ist, wenn das ConnectionLevel auch bei dem ausgewählten M-Relationship angelegt wurde. Andernfalls wird nur ein Dialog angezeigt indem der Benutzer zwar den Inhalt lesen, ihn aber nicht verändern kann. Kann ein ConnectionLevel geändert werden, so werden auch hier wie beim Erstellen eines neuen ConnectionLevels, die möglichen Werte für das Source-Level und das Target-Level vorgegeben.

6.6 Restriktionen der Implementierung

Aufgrund der Eigenheiten von Protégé und des M-Object- bzw. M-Relationship-Ansatzes tauchten bei der Implementierung des Plug-Ins Probleme bzw. Spezialfälle auf, die nur teilweise gelöst werden konnten. Diese werden im Nachfolgenden genauer erläutert.

Die „Undo“ und „Redo“ Funktion in Protégé, die Aktionen rückgängig machen oder wiederherstellen kann, wurde für das Plug-In auf inaktiv gesetzt. Somit ist es dem Benutzer nicht möglich, beim Arbeiten mit dem Plug-In, diese Befehle aufzurufen. Der Grund für das

Deaktivieren dieser Funktionen ist, dass in den standardmäßigen Formularen von Protégé immer eine Einzelaktion (bspw. das Anlegen einer neuen Klasse) stattfindet. Da aber Operationen im Plug-In eine Vielzahl zeitgleicher Einzelaktionen hervorrufen (bspw. werden beim Anlegen eines neuen M-Objects viele neue Klassen für die verschiedenen Abstraktionsebenen angelegt), können diese Funktionen in dieser Form nicht verwendet werden. Ein Wiederherstellen einer einzelnen Aktion würde die Integrität des Plug-Ins bzw. der erstellten Ontologie zerstören. Weiters kann ein Benutzer auch, indem er auf ein anderes standardmäßiges Formular in Protégé, bspw. den „CLASS BROWSER“ wechselt, die Integrität insofern verletzen, indem er dort Klassen löscht oder sie verändert.

Für das Erstellen eines neuen M-Relationships wird für das Source M-Object und für das Target M-Object eine Auswahl an M-Objects getroffen, die dem Benutzer zur Verfügung gestellt wird. Dabei ist die Auswahl des Target M-Objects abhängig davon, welches Source M-Object gewählt wurde. Konkretisiert das zu erstellende M-Relationship kein anderes M-Relationship, so ist die Auswahl des Source M-Objects die Menge aller M-Objects. Als Target M-Objects sind in diesem Fall alle M-Objects erlaubt, die in der gesamten über- und untergeordneten Konkretisierungshierarchie des Source M-Objects nicht vorkommen. Ist das neu zu erstellende M-Relationship allerdings eine Konkretisierung eines anderen M-Relationships, dann können als Source M-Objects alle M-Objects gewählt werden, die eine direkte oder indirekte Konkretisierung des Source M-Objects des übergeordneten M-Relationship sind, sowie das Source M-Object des übergeordneten M-Relationships selbst. Die Auswahl des Target M-Objects beschränkt sich dann auf die M-Objects, die das Target M-Object des übergeordneten M-Relationships direkt oder indirekt konkretisieren. Weiters wird diese Auswahl des Target M-Objects noch um jene Target M-Objects verringert, die bereits in einem M-Relationship mit dem gleichen Source M-Object und dem gleichen Bezeichner vorhanden sind, wie die neu zu erstellende M-Relationship.
Die so erstellte Vorauswahl für Source- und Target M-Objects soll dem Benutzer beim Anlegen eines neuen M-Relationships behilflich sein. Das Erstellen einer sinnvollen Verbindung von M-Objects durch ein M-Relationship ist allerdings Aufgabe des Benutzers.

Beim Anlegen von ConnectionLevels werden dem Benutzer, wie bereits erwähnt, nur die zulässigen Levels angeboten, die miteinander in Beziehung gebracht werden können. Dazu werden sowohl das konkrete M-Relationship und die weiteren M-Relationships entlang der Konkretisierungshierarchie, sowie die mit ihnen verbundenen M-Objects analysiert. Zusätzlich werden noch die eigenen und die vererbten ConnectionLevels betrachtet, um dann schlussendlich dem Benutzer die mögliche Auswahl an Levels für die Erstellung eines neuen ConnectionLevels zur Verfügung zu stellen. Diese Auswahl ist allerdings nur ein grobes Sortiment, das dem Benutzer eine Hilfestellung sein soll. Für die richtige Verbindung der verschiedenen Levels untereinander ist der Benutzer selbst verantwortlich.

7. Benutzerhandbuch

Im Benutzerhandbuch wird anhand des durchgängigen Beispiels Schritt für Schritt die Entstehung eines Modells, im konkreten Fall eines Modells einer Produkthierarchie samt ihren Beziehungen, erklärt. Dabei soll es dem Benutzer eine schnelle Einführung in den Umgang mit dem Plug-In bieten, sodass er nach dem Durchführen der einzelnen Schritte des Benutzerhandbuchs in der Lage ist, eigene Modelle zu erstellen.

Das Benutzerhandbuch setzt direkt bei der Erstellung des Modells an, weshalb auch nicht näher auf die Einbettung bzw. Installation des Plug-Ins in Protégé eingegangen wird. Diese Tätigkeiten werden im Anhang dieser Arbeit genau erläutert.

Die Beschreibung der nachfolgenden Schritte ist so aufgebaut, dass zuerst die Ausgangssituation erläutert wird. Danach werden die Aktionen beschrieben, die vom Benutzer durchgeführt werden müssen. Zum Schluss wird eine Sollsituation bzw. das gewünschte Ergebnis dargestellt, mit dem der Benutzer das aktuell erstellte Modell vergleichen kann, um mögliche Abweichungen bzw. Fehler zu eruieren.

Für eine bessere Übersicht werden die einzelnen Schritte in Blöcke gegliedert, wobei zunächst auf das Erstellen von M-Objects, Level und Attribute näher eingegangen wird, was vor allem Auswirkungen auf den M-Object Browser und den M-Object Editor hat. Anschließend werden die Veränderungen des M-Relationship Browsers und des M-Relationship Editors genauer betrachtet, deren Darstellung sich hauptsächlich aufgrund der Erstellung von M-Relationships und ConnectionLevels ändert.

7.1 Erstellen von Objekten im M-Object Browser und M-Object Editor

Im Folgenden werden jene Schritte des durchgängigen Beispiels nach [NGS09] erklärt, die Auswirkungen auf die beiden Plug-In Elemente M-Object Browser und M-Object Editor nach sich ziehen.

Schritt 1: Erstellen eines M-Objects *Produkt* und den Levels *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt*

Ausgangssituation

Ein leeres Projekt in Protégé wurde erzeugt. Im M-Object Browser sowie im M-Relationship Browser ist nur der Wurzelknoten „BASE“ zu sehen. Der M-Object Editor und auch der M-Relationship Editor zeigen keine relevanten Daten an.

Benutzeraktionen

Für das Erstellen eines neuen M-Objects *Produkt* wird im M-Object Browser in der Toolbar die Operation zur Erstellung eines neuen M-Objects ausgewählt. Es erscheint ein Dialog, indem der Name für das neue M-Object *Produkt* und auch ein Name für den neuen Top-Level *Katalog* eingegeben werden muss. Nach erfolgreichem Abschluss des Dialoges wird im M-Object Editor durch Auswahl der Operation zum Erstellen eines neuen Levels auf der untersten Ebene des M-Object Editors ein neuer Dialog aufgerufen. Er erfordert die Eingabe eines Namens für den neuen Level *Kategorie.* Dieser Schritt wird für den Level *Modell* und den Level *PhysischesObjekt* wiederholt.

Ergebnis

Im M-Object Browser wird ein M-Object *Produkt* dargestellt. Im M-Object Editor wird das M-Object *Produkt*, durch die Abstraktionsebenen *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt* in Form von Tabellen beschrieben. (siehe Abbildung 50)

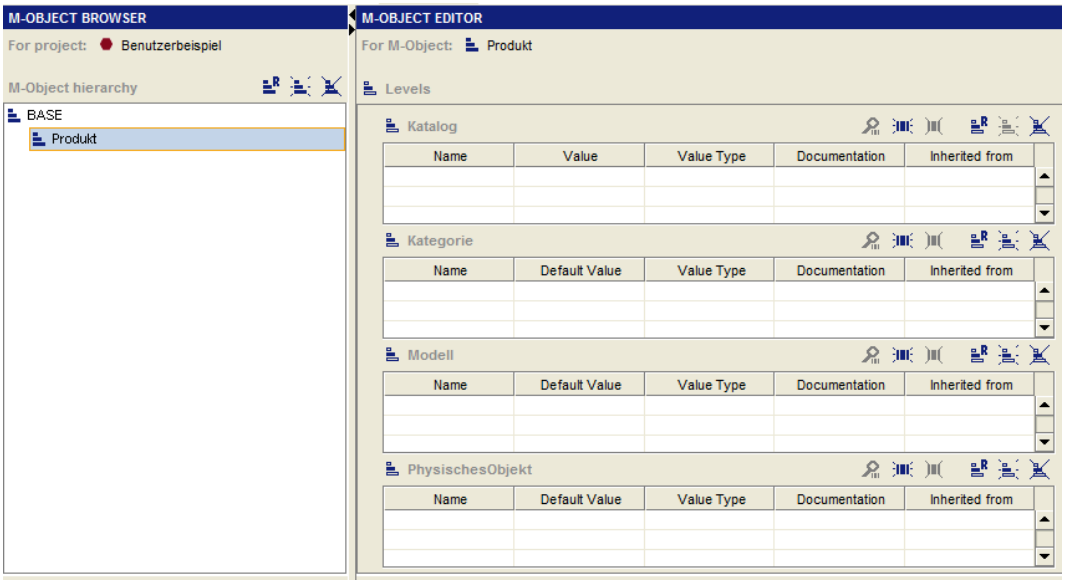

Abbildung 50: Schritt 1 – Erstellen von *Produkt*, *Katalog*, *Kategorie*, *Modell*, *PhysischesObjekt*

Schritt 2: Erstellen der Attribute *bezeichner*, *steuersatz*, *listenpreis* und *serienNr*

Ausgangssituation

Der M-Object Browser enthält ein M-Object *Produkt*, dessen Abstraktionsebenen *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt* im M-Object Editor angezeigt werden.

Benutzeraktionen

Durch Doppelklick der linken Maustaste auf ein leeres Feld in der Tabelle des Levels *Katalog* wird ein Dialog zum Erstellen eines neuen Attributes geöffnet, wobei im Feld Name *bezeichner*, als Wertebereich *String* und als Wert *„Unsere Produkte“* eingetragen werden müssen. Dieser Schritt wird für die weiteren Attribute wiederholt, allerdings auf einer anderen Ebene. Das Attribut *steuersatz* wird auf dem Level *Kategorie* mit dem Wertebereich *Integer*, das Attribut *listenpreis* auf dem Level *Modell* mit dem Wertebereich *Float* und das Attribut *serienNr* auf dem Level *PhysischesObjekt* mit dem Wertebereich *String* angelegt.

Ergebnis

Der M-Object Editor zeigt für das M-Object *Produkt* alle vier Abstraktionsebenen mit den neu eingeführten Attributen *bezeichner*, *steuersatz*, *listenpreis* und *serienNr*, wobei jedes dieser Attribute auf einem anderen Level angesiedelt ist. (siehe Abbildung 51)

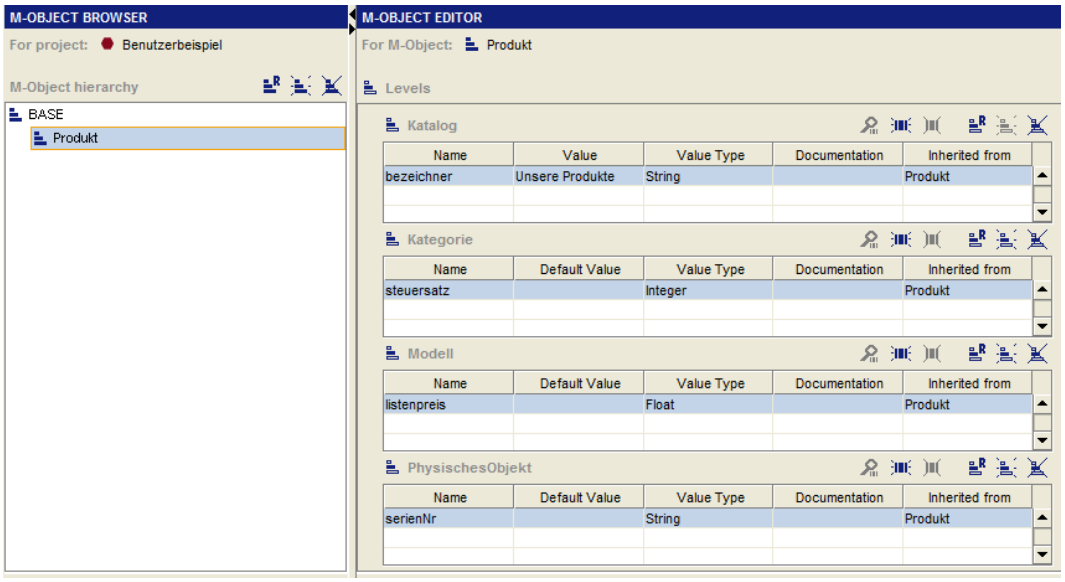

Abbildung 51: Schritt 2 – Erstellen von *bezeichner*, *steuersatz*, *listenpreis*, *serienNr*

Schritt 3: Erstellen des M-Objects *Auto* und der Attribute *maxGeschwindigkeit* und *kilometerstand*

Ausgangssituation

Der M-Object Browser enthält ein M-Object *Produkt*, das keine weiteren Konkretisierungen besitzt. Im M-Object Editor werden die Abstraktionsebenen *Katalog*, *Kategorie*, *Modell* und *PhysischesObjekt* inklusive der angelegten Attribute *bezeichner*, *steuersatz*, *listenpreis* und *serienNr* angezeigt.

Benutzeraktionen

Durch Rechtsklick auf das M-Object *Produkt* im M-Object Browser wird ein Kontextmenü sichtbar, indem die Operation zum Erstellen eines neuen M-Objects ausgewählt wird. Es erscheint ein Dialog zur Anlage eines neuen M-Objects auf der Hierarchieebene *Kategorie*, wobei im Feld Name *Auto* eingetragen werden muss. Nach erfolgreicher Beendigung des Dialoges erscheinen im M-Object Editor die geerbten Abstraktionsebenen des neu erstellten M-Objects *Auto*.

Durch Doppelklick der linken Maustaste auf ein leeres Feld in der Tabelle des Levels *Modell* wird ein Dialog zum Erstellen eines neuen Attributes geöffnet, wobei im Feld Name *maxGeschwindigkeit* und als Wertebereich *Integer* eingetragen werden muss. Für die Erstellung des Attributes *kilometerstand* wird dieser Schritt auf dem Level *PhysischesObjekt* wiederholt, wobei auch hier für den Wertebereich *Integer* gewählt wird.

Durch Doppelklick auf das Attribut *steuersatz* auf dem Level *Kategorie* erscheint ein Dialog indem ein konkreter Wert für dieses Attribut eingetragen werden kann. Der Wert dieses Attributes wird mit „20“ belegt.

Ergebnis

Im M-Object Browser stellt das neu erstellte M-Object *Auto* eine Konkretisierung des M-Objects *Produkt* dar, was dadurch ersichtlich ist, dass es in der Baumstruktur dem M-Object *Produkt* untergeordnet ist. Der M-Object Editor zeigt für das M-Object *Auto* die geerbten Abstraktionsebenen *Kategorie*, *Modell* und *PhysischesObjekt* inklusive der geerbten und neu erstellten Attribute *maxGeschwindigkeit* und *kilometerstand*. Weiters ist der Wert des Attributes *steuersatz* ersichtlich. (siehe Abbildung 52)

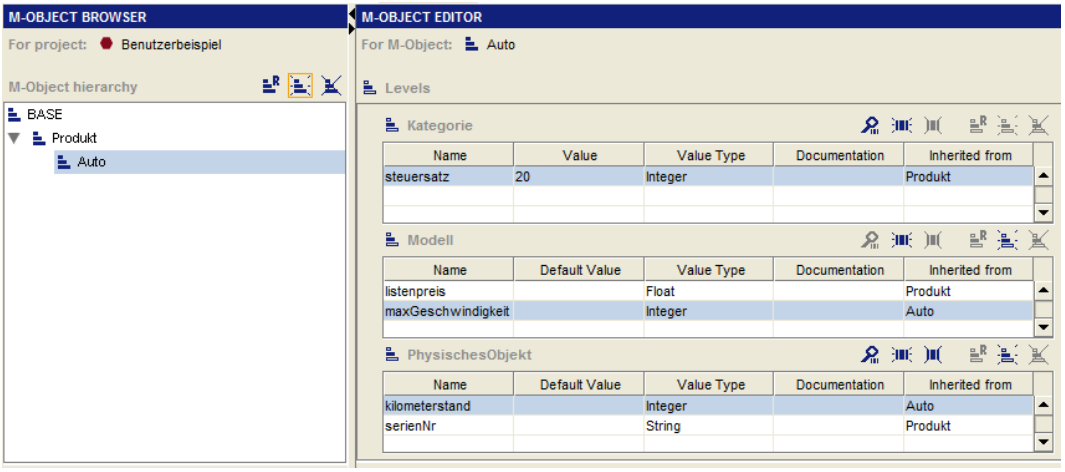

Abbildung 52: Schritt 3 – Erstellen von *Auto*, *maxGeschwindigkeit*, *kilometerstand*

Schritt 4: Erstellen des M-Objects *Buch* und des Attributes *autor*

Ausgangssituation

Der M-Object Browser enthält zwei M-Objects *Produkt* und *Auto*, wobei *Auto* eine Konkretisierung von *Produkt* darstellt. Im M-Object Editor werden die Abstraktionsebenen *Kategorie*, *Modell* und *PhysischesObjekt* inklusive der Attribute, für das in Schritt 3 angelegte M-Object *Auto*, angezeigt.

Benutzeraktionen

Wie bereits bei der Erstellung des M-Objects *Auto* wird durch Rechtsklick auf das M-Object *Produkt* im M-Object Browser ein Kontextmenü geöffnet und die Operation für das Erstellen

eines M-Objects ausgewählt. Ein Dialog für das Erstellen eines neuen M-Objects auf dem Level *Kategorie* erscheint, wobei im Feld Name *Buch* einzugeben ist. Nach erfolgreicher Beendigung des Dialoges erscheinen im M-Object Editor die geerbten Levels des M-Objects *Buch*. Es sind dies die Level *Kategorie*, *Modell* und *PhysischesObjekt*.

Durch Auswahl der Operation zum Erstellen eines neuen Attributes in der Toolbar des Levels *Modell* wird ein Dialog aufgerufen. Im Feld Name muss *autor* eingegeben und als Wertebereich *String* ausgewählt werden.

Durch Doppelklick auf das Attribut *steuersatz* auf dem Level *Kategorie* erscheint ein Dialog indem ein konkreter Wert für dieses Attribut eingetragen werden kann. Der Wert dieses Attributes wird mit „10“ belegt.

Ergebnis

Der M-Object Browser enthält nun das M-Object *Produkt* und zwei direkt untergeordnete M-Objects *Auto* und *Buch*. Der M-Object Editor zeigt für das M-Object *Buch* die geerbten Abstraktionsebenen *Kategorie*, *Modell* und *PhysischesObjekt*, sowie die geerbten Attribute und das neu erstellte Attribut *autor*. Weiters ist der Wert des Attributes *steuersatz* ersichtlich. (siehe Abbildung 53)

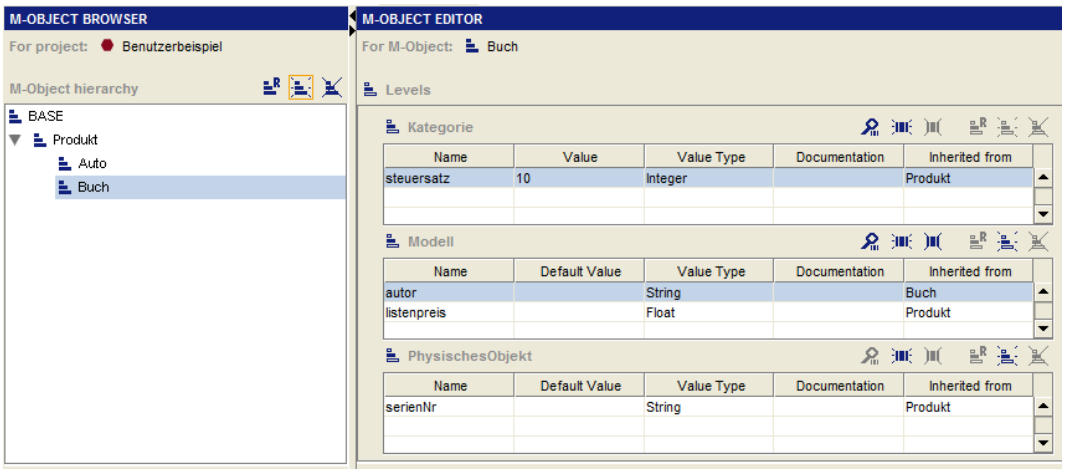

Abbildung 53: Schritt 4 – Erstellen von *Buch*, *autor*

Schritt 5: Erstellen des Levels *Marke* und des Attributes *markteinführung*

Ausgangssituation

Der M-Object Browser zeigt die M-Objects *Produkt*, *Auto* und *Buch*, wobei die beiden letzteren in der Baumstruktur unterhalb von *Produkt* zu finden sind. Im M-Object Editor werden die Abstraktionsebenen *Kategorie*, *Modell* und *PhysischesObjekt* inklusive der Attribute, für das in Schritt 4 angelegte M-Object *Buch*, angezeigt.

Benutzeraktionen

Im M-Object Browser muss mittels Maus das M-Object *Auto* ausgewählt werden. Die Abstraktionseben des M-Objects werden nun im M-Object Editor angezeigt. Oberhalb des Levels *Modell* wird im M-Object Editor durch Auswahl der Operation zum Erstellen eines neuen Levels ein Dialog aufgerufen. Dieser Dialog dient zur Erstellung eines neuen Levels, wobei im Feld Name *Marke* eingetragen werden muss. Nach erfolgreicher Beendigung des Dialoges erscheint im M-Object Editor der neue Level *Marke* zwischen den beiden Levels *Kategorie* und *Modell*.

Durch Doppelklick mit der linken Maustaste auf eine leere Zeile innerhalb der Tabelle des Levels *Marke* wird ein Dialog zum Anlegen eines neuen Attributes geöffnet. Im Feld Name dieses Dialogs muss *markteinführung* eingetragen werden und als Wertebereich wird *String* ausgewählt.

Ergebnis

Der M-Object Browser ist unverändert und enthält wie in Schritt 4 die M-Objects *Produkt*, *Auto* und *Buch*. Der M-Object Editor zeigt das M-Object *Auto* mit den Abstraktionsebenen *Kategorie*, *Marke*, *Modell* und *PhysischesObjekt*, wobei *Marke* neu eingeführt wurde. Neben den Attributen auf den verschiedenen Abstraktionsebenen wird in der Tabelle des Levels *Marke* das neu erstellte Attribut *markteinführung* veranschaulicht. (siehe Abbildung 54)

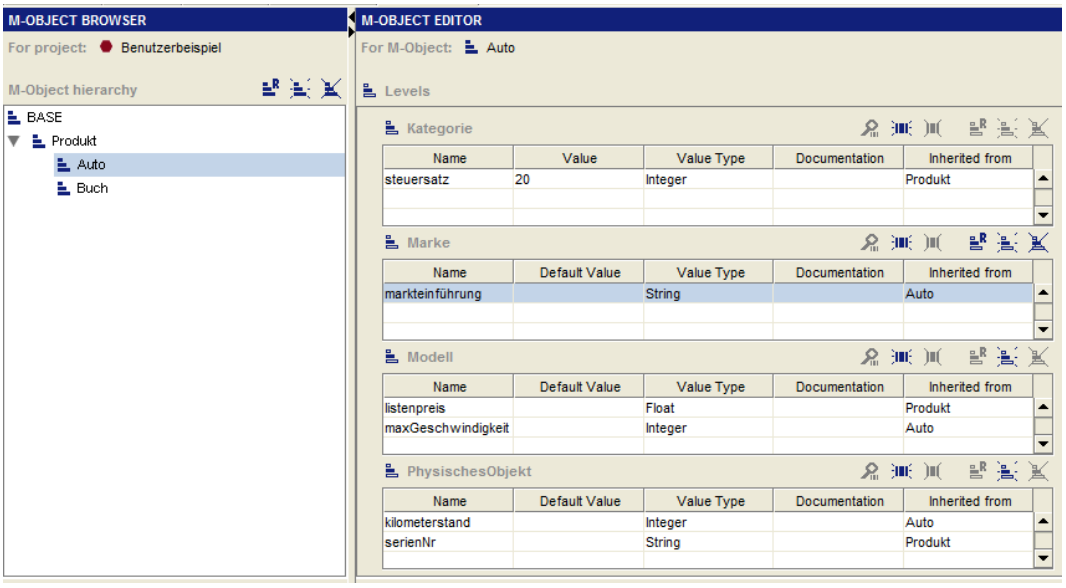

Abbildung 54: Schritt 5 – Erstellen von *Marke*, *markteinführung*

Schritt 6: Erstellen der M-Objects *Porsche911*, *HarryPotter1* und des Attributes *porsche911Club*

Ausgangssituation

Der M-Object Browser zeigt die M-Objects *Produkt*, *Auto* und *Buch*, wobei das M-Object *Auto* selektiert ist. Der M-Object Editor zeigt die Abstraktionsebenen inklusive der Attribute des M-Objects *Auto* an. Es sind dies die Ebenen *Kategorie*, *Marke*, *Modell* und *PhysischesObjekt*.

Benutzeraktionen

Durch Klick mittels der linken Maustaste im M-Object Browser in der Toolbar auf die Operation zum Erstellen eines neuen M-Objects wird der Dialog zum Erstellen eines neuen M-Objects aufgerufen. Es muss in das Feld Name *Porsche911* eingegeben werden. Nach positiver Beendigung des Dialoges erscheinen im M-Object Editor die geerbten Abstraktionsebenen des neuen M-Objekts.

Durch Doppelklick der linken Maustaste in der Tabelle des Levels *PhysischesObjekt* öffnet sich ein Dialog zum Erstellen eines neuen Attributes, indem für das Feld Name *porsche911Club* eingegeben und als Wertebereich *Boolean* ausgewählt werden muss. Im Anschluss daran wird das neu erstellte Attribut in der Tabelle der Ebene *PhysischesObjekt* angezeigt.

Ein weiterer Doppelklick der linken Maustaste auf das existierende Attribut *markteinführung* führt zum Öffnen eines Dialoges indem der Wert „1966“ eingetragen wird.

Für das Erstellen eines neuen M-Objects *HarryPotter1* wird die Vorgangsweise wie beim Erstellen für das M-Object *Porsche911* wiederholt, mit dem Unterschied, dass das M-Object *HarryPotter1* eine Konkretisierung von *Buch* darstellt und deshalb der Fokus bei der Erstellung im M-Object Browser auf dem M-Object *Buch* liegen muss. Weiters muss für die existierenden Attribute *autor* und *listenpreis* beim neuen M-Object *HarryPotter1* auf dem Level *Modell* ein Wert eingegeben werden. Dies geschieht, wie auch schon zuvor beim Attribut *markteinführung*, durch Doppelklick der linken Maustaste auf das jeweilige Attribut. Für das Attribut *autor* wird der Wert „J.K. Rowling“ und für das Attribut *listenpreis* der Wert „11,5“ eingegeben.

Ergebnis

Der M-Object Browser enthält fünf M-Objects, wobei die neu erstellten M-Objects *Posche911* und *HarryPotter1* jeweils in der Hierarchie unterhalb des M-Objects *Auto* bzw. *Buch* dargestellt werden. Der M-Object Editor stellt das zuletzt erstellte M-Object *HarryPotter1* mit den geerbten Abstraktionsebenen *Modell* und *PhysischesObjekt* inklusive der geerbten Attribute dar. Der Wert der Attribute *autor* und *listenpreis* ist auf dem Top-Level im M-Object Editor zu sehen. (siehe Abbildung 55)

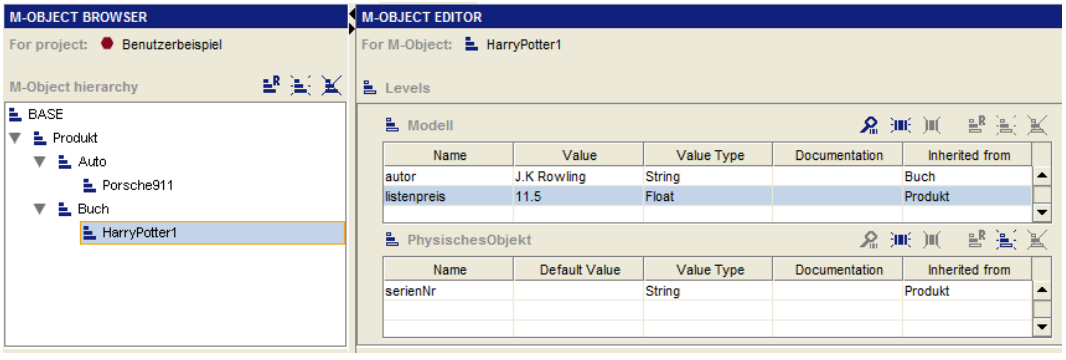

Abbildung 55: Schritt 6 – Erstellen von *Porsche911*, *HarryPotter1*, *porsche911Club*

Schritt 7: Erstellen der M-Objects *Porsche911CarreraS* und *Porsche911GT3*

Ausgangssituation

Im M-Object Browser werden die fünf existierenden M-Objects *Produkt*, *Auto*, *Buch*, *Porsche911* und *HarryPotter1* dargestellt, wobei letzteres selektiert ist. Der M-Object Editor zeigt die Abstraktionsebenen inklusive der Attribute des M-Objects *HarryPotter1* an. Es sind dies die Level *Modell* und *PhysischesObjekt*.

Benutzeraktionen

Für das Erstellen der beiden M-Objects *Porsche911CarreraS* und *Porsche911GT3* wird der Fokus im M-Object Browser auf das M-Object *Porsche911* gelegt. Im Anschluss wird über die Toolbar des M-Object Browsers die Operation für das Erstellen eines neuen M-Object aufgerufen und über den Dialog, der sich öffnet, nacheinander jeweils die beiden neuen M-Objects *Porsche911CarreraS* und *Porsche911GT3* angelegt.

Weiters müssen ebenfalls nacheinander die beiden neuen M-Objects selektiert werden und im M-Object Editor Werte für die Attribute *listenpreis* und *maxGeschwindigkeit* vergeben werden. Dies geschieht durch Doppelklick der linken Maustaste auf das entsprechende Attribut, wobei sich zur Eingabe ein Dialog öffnet. Es sind für das M-Object *Porsche911CarreraS* die Werte „91500“ und „293“ für die Attribute *listenpreis* und *maxGeschwindigkeit* einzutragen. Für das M-Object *Porsche911GT3* werden die Werte „104000“ und „310“ eingesetzt.

Ergebnis

Im M-Object Browser werden sieben M-Objects dargestellt, wobei die neu erstellten M-Objects *Posche911CarreraS* und *Prorsche911GT3* jeweils in der Hierarchie unterhalb des M-Objects *Porsche911* zu finden sind. Der M-Object Editor stellt das zuletzt erstellte M-Object *Porsche911GT3* mit den geerbten Abstraktionsebenen *Modell* und *PhysischesObjekt* inklusive der geerbten Attribute dar. Auch die Werte der Attribute *listenpreis* und *maxGeschwindigkeit* sind auf dem Top-Level *Modell* im M-Object Editor zu sehen. (siehe Abbildung 56)

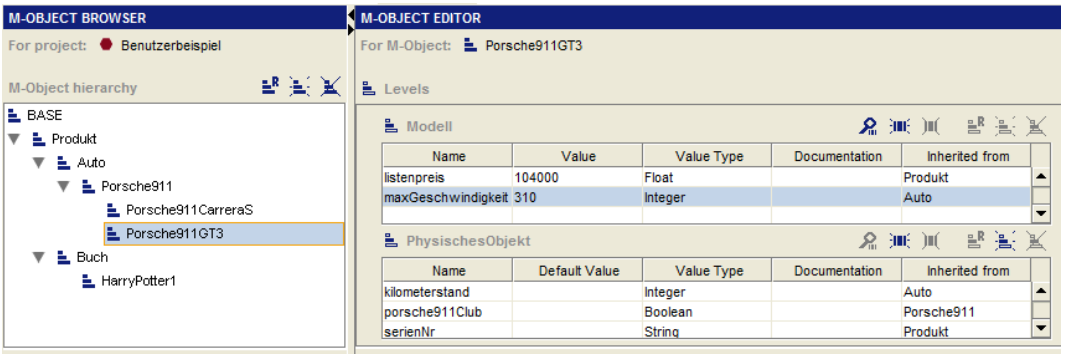

Abbildung 56: Schritt 7 – Erstellen von *Porsche911CarreraS*, *Porsche911GT3*

Schritt 8: Erstellen der M-Objects *meinPorsche911CarreraS* und *meinHarryPotter1*

Ausgangssituation

Der M-Object Browser enthält die M-Objects *Produkt*, *Auto*, *Buch*, *Porsche911, HarryPotter1*, *Porsche911CarreraS* und *Porsche911GT3*, wobei das M-Object *Porsche911GT3* selektiert ist. Im M-Object Editor werden die geerbten Levels des M-Objects *Porsche911GT3* inklusive deren Attribute auf den einzelnen Abstraktionsebenen *Modell* und *PhysischesObjekt* angezeigt.

Benutzeraktionen

Zum Erstellen des M-Objects *meinPorsche911CarreraS* wird das M-Object *Porsche911CarreraS* ausgewählt und im Kontextmenü, das durch Rechtsklick mit der Maus auf das selektierte M-Object geöffnet wird, die Operation zum Erstellen eines neuen M-Objects aufgerufen. Im folgenden Dialog wird als Name für das neue M-Object *meinPorsche911CarreraS* eingetragen. Nach erfolgreichem Abschluss des Dialogs erscheint das Top-Level *PhysischesObjekt* im M-Object Editor. Dort wird jeweils durch Doppelklick der linken Maustaste auf die Attribute *kilometerstand*, *serienNr* und *porsche911Club* ein Dialog zur Eingabe eines Wertes geöffnet. Für das Attribut *kilometerstand* muss der Wert „80000“, für das Attribut *serienNr* der Wert „G876JJJ3“ und für das Attribut *porsche911Club* der Wert „true“ eingegeben werden.

Das M-Object *meinHarryPotter1* wird als Konkretisierung des M-Objects *HarryPotter1* angelegt, weshalb der Fokus auf *HarryPotter1* gesetzt werden muss. Das Vorgehen zur Erstellung des neuen M-Objects *meinHarryPotter1* entspricht dem der Erstellung des vorigen M-Objects mein*Porsche911CarreraS*. Für den Wert des Attributes *serienNr* wird im M-Object

Editor auf dem Top-Level *PhysischesObjekt* des M-Objects *meinHarryPotter1* der Wert „A45J6“ eingegeben. Anschließend wird das M-Object *meinPorsche911CarreraS* im M-Object Browser selektiert.

Ergebnis

Der M-Object Browser enthält neun M-Objects, wobei die neu erstellten M-Objects *meinPosche911CarreraS* und mein*HarryPotter1* jeweils in der Hierarchie unterhalb des M-Objects *Porsche911CarreraS* bzw. *HarryPotter1* dargestellt werden. Der M-Object Editor stellt das M-Object *meinPorsche911CarreraS* mit dem Top-Level *PhysischesObjekt* und der Attribute *kilometerstand*, *porsche911Club* und *serienNr* dar. (siehe Abbildung 57)

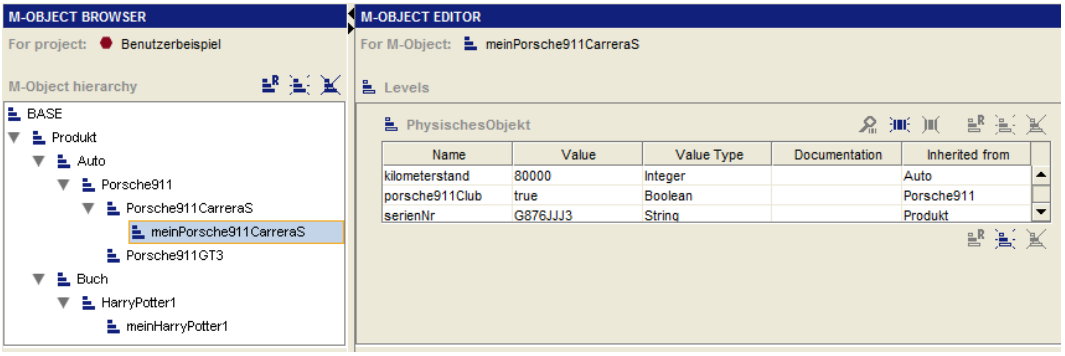

Abbildung 57: Schritt 8 – Erstellen von *meinPorsche911CarreraS*, *meinHarryPotter1*

Schritt 9: Erstellen eines M-Objects *Konzern* und den Levels *Root*, *Industriesektor*, *Unternehmen* und *Fabrik*

Ausgangssituation

Der M-Object Browser zeigt eine komplette Produkthierarchie nach [NGS09]. Der M-Object Editor stellt die Level des zuletzt erstellten, aktuell selektierten M-Objects *meinPorsche911CarreraS* dar.

Benutzeraktionen

Für das Anlegen eines neuen M-Objects *Konzern* muss im M-Object Browser der Wurzelknoten „BASE“ selektiert werden. Durch Klick mit der linken Maustaste in der Toolbar des M-Object Browsers wird die Operation zur Erstellung eines neuen M-Objects ausgewählt. Es erscheint ein Dialog, indem der Name für das neue M-Object *Konzern* und auch ein Name für den neuen Top-Level *Root* eingegeben werden muss. Nach erfolgreichem Abschluss des Dialoges wird, durch das Betätigen der Operation zur Erstellung eines neuen

Levels auf der untersten Ebene des M-Object Editors, ein neuer Dialog aufgerufen. Er erfordert die Eingabe eines Namens für den neuen Level *Industriesektor.* Dieser Schritt wird für den Level *Unternehmen* und den Level *Fabrik* wiederholt.

Ergebnis

Im M-Object Browser wird neben der einzelnen M-Objects in der Produkthierarchie das M-Object *Konzern* dargestellt. Im M-Object Editor wird das M-Object *Konzern*, durch die Abstraktionsebenen *Root*, *Industriesektor*, *Unternehmen* und *Fabrik* in Form von Tabellen beschrieben. (siehe Abbildung 58)

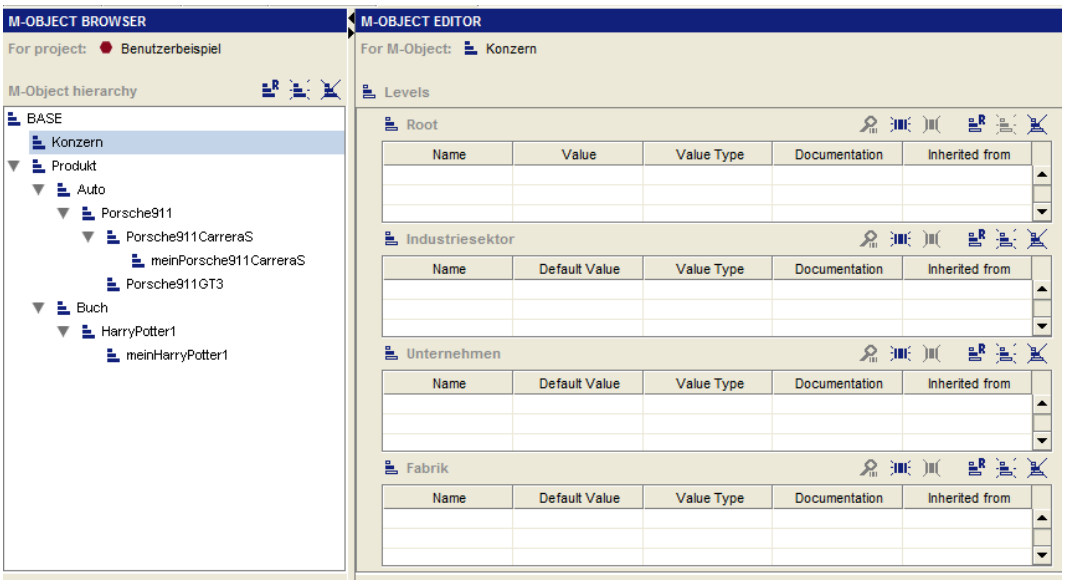

Abbildung 58: Schritt 9 – Erstellen von *Konzern*, *Root*, *Industriesektor*, *Unternehmen*, *Fabrik*

Schritt 10: Erstellen der M-Objects *Autohersteller*, *PorscheLtd* und *PorscheZuffenhausen*

Ausgangssituation

Der M-Object Browser zeigt neben der Produkthierarchie auch das M-Object *Konzern*, das aktuell selektiert ist. Der M-Object Editor stellt die Level *Root*, *Industriesektor*, *Unternehmen*, und *Fabrik* des M-Objects *Konzern* dar.

Benutzeraktionen

Durch Rechtklick auf das M-Object *Konzern* erscheint ein Kontextmenü, indem die Operation für das Erstellen eines neuen M-Objects ausgewählt werden muss. Es öffnet sich ein Dialog,

indem der Name für das neue M-Object *Autohersteller* einzugeben ist. Nach erfolgreicher Beendigung des Dialogs wird das neue M-Object erstellt.

Für die weiteren zu erstellenden M-Objects wird dieser Vorgang wiederholt, wobei als Ausgangsbasis für die Erstellung immer das vorhergehende M-Object herangezogen wird. Das bedeutet, dass das M-Object *PorscheLdt* als Konkretisierung des M-Objects *Autohersteller* und das M-Object *PorscheZuffenhausen* als Konkretisierung des M-Objects *PorscheLtd* angelegt werden müssen.

Ergebnis

Im M-Object Browser wird die Produkthierarchie und die M-Objects *Konzern, Autohersteller, PorscheLtd* und *PorscheZuffenhausen* dargestellt, wobei die vier genannten M-Objects der Reihe nach direkte Konkretisierungen des jeweils vorhergehenden M-Objects darstellen. Im M-Object Editor wird das zuletzt erstellte M-Object *PorscheZuffenhausen* durch das Top-Level *Fabrik* beschrieben. (siehe Abbildung 59)

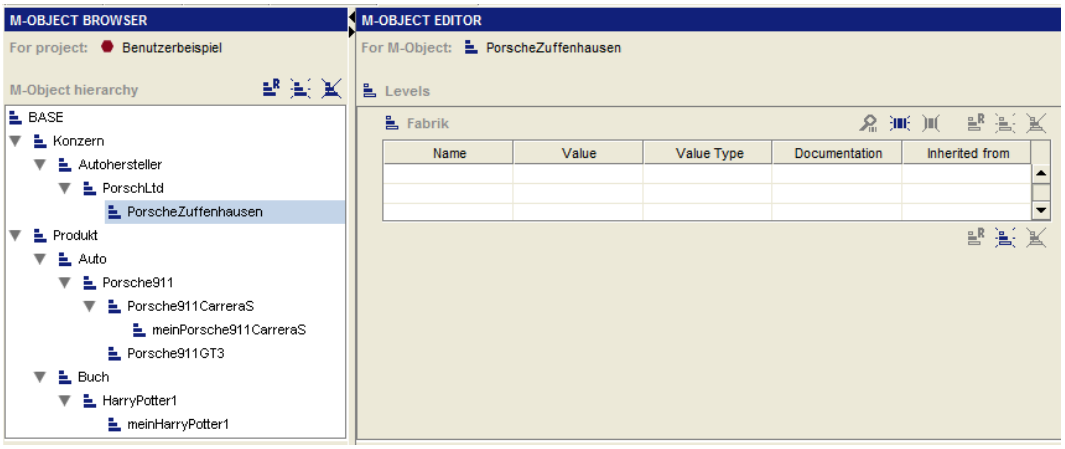

Abbildung 59: Erstellen von *Autohersteller*, *PorscheLtd*, *PorscheZuffenhausen*

7.2 Erstellen von Objekten im M-Relationship Browser und M-Relationship Editor

Die nachfolgen Schritte zur Erstellung des durchgängigen Beispiels beeinflussen den Inhalt der Plug-In Elemente M-Relationship Browser und M-Relationship Editor. Im Folgenden werden diese Schritte genauer erklärt.

Schritt 1: Erstellen des M-Relationships *Produkt-produziertVon-Konzern*

Ausgangssituation

Der M-Relationship Browser enthält nur den Wurzelknoten „BASE“. Der M-Relationship Editor zeigt keine relevanten Daten.

Benutzeraktionen

Nach der Selektion des Wurzelknotens „BASE“ muss über die Toolbar im M-Relationship Browser die Operation zum Erstellen eines neuen M-Relationships aufgerufen werden. In dem erscheinenden Dialog ist der Name „produziertVon“ einzugeben. Als Werte für das Source-M-Object und das Target-M-Object werden die M-Objects *Produkt* und *Konzern* ausgewählt. Nach erfolgreicher Beendigung des Dialoges wird das neue M-Relationship angelegt.

Ergebnis

Der M-Relationship Browser zeigt das neu erstellte M-Relationship *Produkt-produziertVon-Konzern*. Im M-Relationship Editor werden die Attribute des M-Relationships *Produkt-produziertVon-Konzern* angezeigt. (siehe Abbildung 59)

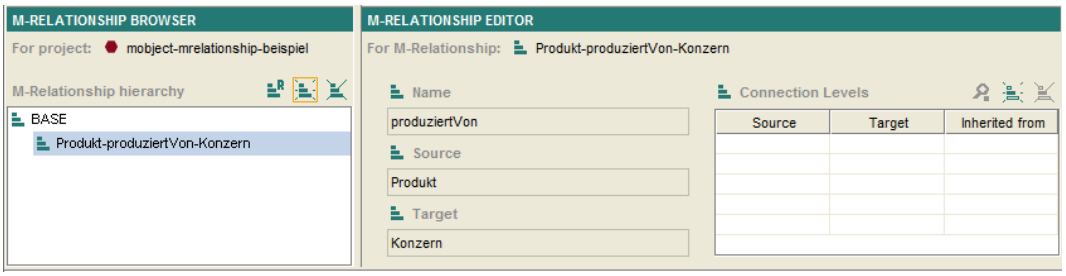

Abbildung 60: ***Erstellen von Produkt-produziertVon-Konzern***

Schritt 2: Erstellen der ConnectionLevel *Kategorie-Industriesektor*, *Modell-Unternehmen* und *PhysischesObjekt-Fabrik*

Ausgangssituation

Der M-Relationship Browser enthält das M-Relationship *Produkt-produziertVon-Konzern*. Der M-Relationship Editor zeigt die Attribute des M-Relationships *Produkt-produziertVon-Konzern*.

Benutzeraktionen

Im M-Relationship Editor öffnet sich durch Doppelklick mit der linken Maustaste auf die Tabelle der ConnectionLevels ein Dialog zur Eingabe eines neuen ConnectionLevels. Hier muss als Source-Level *Kategorie* und als Target-Level *Industriesektor* ausgewählt werden. Nach erfolgreicher Beendigung des Dialoges wird der neue ConnectionLevel *Kategorie-Industriesektor* erstellt. Diese Vorgehensweise wird für die ConnectionLevels *Modell-Unternehmen* und *PhysischesObjekt-Fabrik* wiederholt.

Ergebnis

Der M-Relationship Browser zeigt das M-Relationship *Produkt-produziertVon-Konzern*, das aktuell selektiert ist. Im M-Relationship Editor werden die Attribute des M-Relationships *Produkt-produziertVon-Konzern* und die neu angelegten ConnectionLevels *Kategorie-Industriesektor*, *Modell-Unternehmen* und *PhysischesObjekt-Fabrik* angezeigt. (siehe Abbildung 61Abbildung 59)

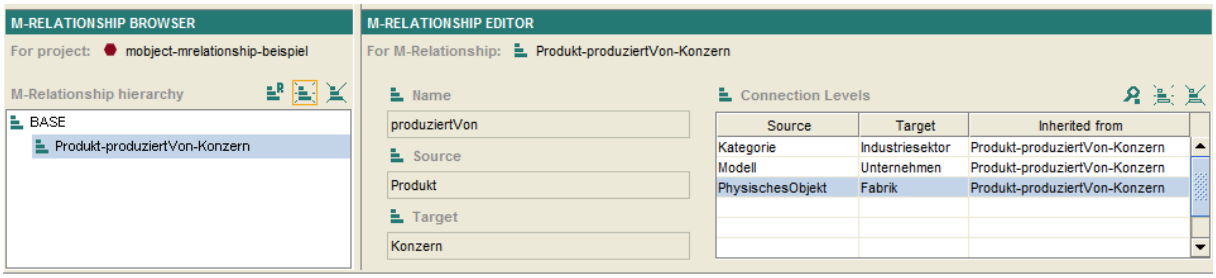

Abbildung 61: Erstellen von *Kategorie-Industriesektor*, *Modell-Unternehmen*, *PhysischesObjekt-Fabrik*

Schritt 3: Erstellen des M-Relationships *Auto-produziertVon-Autohersteller*

Ausgangssituation

Der M-Relationship Browser enthält das M-Relationship *Produkt-produziertVon-Konzern*, das selektiert ist. Der M-Relationship Editor zeigt die Attribute des M-Relationships *Produkt-produziertVon-Konzern* inklusive der ConnectionLevels *Kategorie-Industriesektor*, *Modell-Unternehmen* und *PhysischesObjekt-Fabrik*.

Benutzeraktionen

Im M-Relationship Browser wird durch Auswahl der Operation zum Erstellen eines neuen M-Relationships in der Toolbar ein Dialog geöffnet. Der Name „produziertVon“ ist vom übergeordneten M-Relationship bereits fest vorgegeben und kann nicht mehr geändert

werden. Als Souce-M-Object muss *Auto* und als Target-M-Object *Autohersteller* eingegeben werden. Nach erfolgreicher Beendigung des Dialoges wird das neue M-Relationship *Auto-produziertVon-Autohersteller* angelegt.

Ergebnis

Der M-Relationship Browser zeigt das M-Relationship *Produk-produziertVon-Konzern* und das untergeordnete M-Relationship *Auto-produziertVon-Autohersteller*. Im M-Relationship Editor werden die Attribute des M-Relationships *Auto-produziertVon-Autohersteller sowie* die geerbten ConnectionLevels angezeigt. (siehe Abbildung 62)

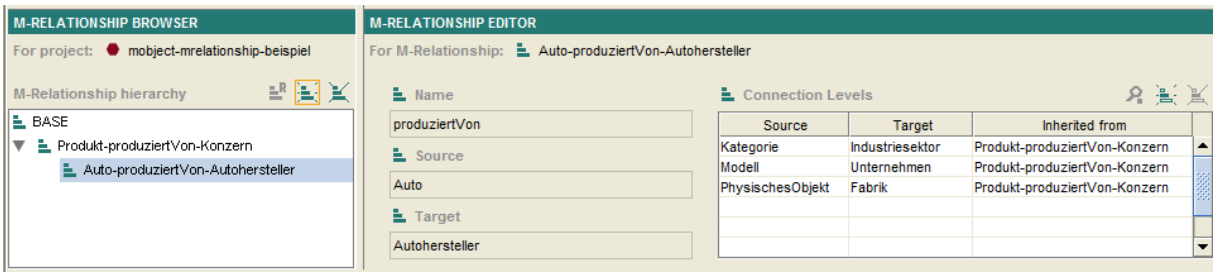

Abbildung 62: Erstellen von *Auto-produziertVon-Autohersteller*

7.3 Abschließende Bemerkungen

In den vorangegangen zwölf Schritten wurde das gesamte durchgängige Beispiel mittels des Plug-Ins in Protégé modelliert. Es wurde gezeigt, wie ein konkretes Beispiel, in diesem Fall ein Beispiel einer Produkthierarchie, modelliert werden kann. Durch das Durchführen der einzelnen Schritte hat der Benutzer sämtliche Plug-In Elemente und deren Funktionen kennengelernt und ist nun in der Lage ein eigenes Modell zu erstellen.

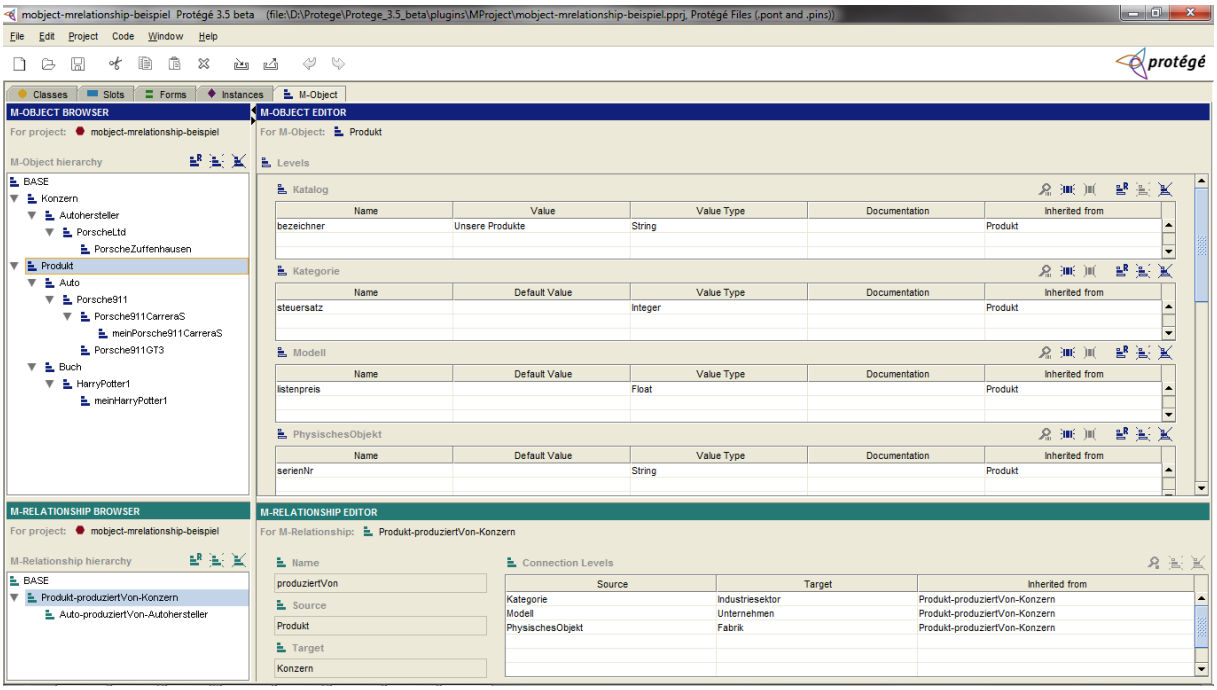

Abbildung 63: Modell einer Produkthierarchie in Protégé

Abbildung 63 zeigt das gesamte in vorhergehenden Kapitel erstellte durchgängige Beispiel mit seinen Kernelementen M-Object Bowser, M-Object Editor, M-Relationship Browser und M-Relationship Editor. Diese Grafik stellt das Endergebniss in seiner Gesamtheit dar, so wie es aussehen sollte, wenn alle vorher erwähnten Schritte richtig durchgeführt wurden.

8. Zusammenfassung

In dieser Arbeit wurde zunächst auf die Multi-Level Modellierung an sich und deren praktische Bedeutung näher eingegangen. In diesem Zusammenhang wurden auch bereits bekannte Techniken zur Multi-Level Modellierung kurz erwähnt. Die Multi-Level Modellierung mit M-Objects und M-Relationships wurde hingegen ausführlicher in den Grundlagen beschrieben und durch ein Beispiel illustriert. Auch die Open-Source Plattform Protégé wurde näher beschrieben, wobei das Hauptaugenmerk auf dem zugrundeliegenden Wissensmodell lag.

Das Mapping von M-Objects und M-Relationships in Protégé war von zentraler Bedeutung für die weiterführende Arbeit. Mit den in Protégé vorhandenen Elementen wurde ein Modell erstellt, das beschreibt, wie M-Objects, M-Relationships, die verschiedenen Abstraktionsebenen, Attribute, etc. in Protégé repräsentiert werden. Dieses Modell spiegelt das strukturelle Mapping wieder, indem die Darstellung des M-Object- und M-Relationshipansatzes in Protégé bis ins Detail beschrieben wurde. Auch das operative Mapping, also die Auswirkungen von Operationen auf M-Objects bzw. M-Relationships, wurden unterstützt durch ein Beispiel näher erläutert.

Im Mittelpunkt dieser Arbeit stand neben der Erstellung des Mappings die Implementierung eines Plug-Ins für Protégé. Dieses Plug-In stellt eine Benutzeroberfläche zur Verfügung, die die einfache und übersichtliche Erstellung und Verwaltung von Modellen im Sinne des M-Object- und M-Relationshipansatzes in Protégé gewährleistet. Diese Benutzeroberfläche inklusive ihren verschiedenen Elementen und Funktionen wurde durch Grafiken, einer genauen Beschreibung und einem Benutzerhandbuch näher erläutert.

Diese Arbeit hat aufgezeigt, dass sich der M-Object- und M-Relationshipansatz in Protégé-Frames umsetzen lässt. Einerseits war das Mapping durchaus problemlos, da sich alle Eigenheiten der M-Objects und M-Relationships gut durch die vorhandenen Konstrukte in Protégé darstellen liesen, anderseits bietet Protégé an sich eine gute Unterstützung für die Erstellung von Plug-Ins an, was natürlich die Programmierung erheblich erleichtert hat.

Literaturverzeichnis

[DPZ02] Dahchour, M., Pirotte, A., Zimányi, E.: *Materialization and Its Metaclass Implementation*. Knowledge an Data Engineering, IEEE Transactions. 2002.

[Ecli15a] *Eclipse: What is Eclipse?* http://help.eclipse.org [letzter Besuch 28.08.2015]

[Ecli15b] *Eclipse: About the Eclipse Foundation.* https://eclipse.org/org/ [letzter Besuch 29.08.2015]

[Ecli15c] *Eclipse: Releases*. https://projects.eclipse.org/releases/ [letzter Besuch 29.08.2015]

[Hub11] Huber, J.: *Multi-Level Modellierung und OWL -Implementierung von Mapping und Werkzeug-Unterstützung.* Institut für Wirtschaftsinformatik – Data & Knowledge Engineering. Linz, Austria, 2011.

[NGS09] Neumayr, B., Grün, K., Schrefl, M.: Multi-*Level Domain Modeling with M-Objects and M-Relationships*. Proceedings of the 6th Asia-Pacific Conference on Conceptual Modeling, Wellington, New Zealand, 2009.

[Neu10] Neumayr, B.: *Multi-Level Modeling with M-Objects und M-Relationships.* Johannes Kepler Universität Linz – Institut für Wirtschaftsinformatik – Data & Knowledge Engineering. Linz, Austria, 2010.

[NeSc08] Neumayr, B., Schrefl, M.: *Comparison Criteria for Ontological Multi-Level Modeling.* Institut für Wirtschaftsinformatik – Data & Knowledge Engineering, Linz, Austria, 2008.

[NeSc09] Neumayr, B., Schrefl, M.: *Multi-Level Conceptual Modeling and OWL*. Proceedings of the Joint International Workshop on Metamodels, Ontologies, Semantic Technologies and Information Systems for the Semantic Web (MOST-ONISW 2009) held in conjunction with the 28th International Conference on Conceptual Modeling. Gramado, Brasilien, 2009.

[NFM00] Noy, N.F., Fergerson, R.W., Musen, M.A.: *The Knowledge Model ofProtégé-2000: Combining Interoperability and Flexibility.* Proceedingsof the 2nd International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000). Juan-les-Pins, Frankreich, 2000.

[NJSS14] Neumayr, B., Jeusfeld, M. A., Schrefl, M., Schütz C.: Dual Deep Instantiation and Its ConceptBase Implementation. CAiSE, LNCS 8484, pp. 503–517, 2014.

[NST08] Neumayr, B., Schrefl, M., Thalheim, B.: *Modeling Techniques for Multi-level Abstraction. The Evolution of Conceptual Modeling.* Springer Verlag. 2011.

[NST10] Neumayr, B., Schrefl M., Thalheim B.: *Hetero-homogeneous hierarchies in* data warehouses. Proceedings of the Seventh Asia-Pacfic Conference on Conceptual Modelling (APCCM 2010). Brisbane, Australia, 2010.

[Sch10] Schütz, C.: *Extending data warehouses with hetero-homogeneous dimension hierarchies and cubes: A proof-of-concept prototype in Oracle*. Institut für Wirtschaftsinformatik – Data & Knowledge Engineering. Linz, Austria, 2010.

[Stan15a] *Index of plugins.* Homepage of the Stanford Center for Biomedical Informatics Research. http://bmir-stage.stanford.edu/doc/pdk/plugins/
[letzter Besuch: 10.08.2015]

[Stan15b] *What is Protégé?* Homepage of the Stanford Center for Biomedical Informatics Research. http://protegewiki.stanford.edu/wiki/Main_Page
[letzter Besuch: 12.08.2015]

[Stan15c] *Protégé-Frames*. Homepage of the Stanford Center for Biomedical Informatics Research. http://protegewiki.stanford.edu/wiki/Protege-Frames
[letzter Besuch: 12.08.2015]

[Stan15d] *Protégé*. Homepage of the Stanford Center for Biomedical Informatics Research. http://protegewiki.stanford.edu/wiki/Protege
[letzter Besuch: 14.08.2015]

[Stan15f] *Protégé-OWL*. Homepage of the Stanford Center for Biomedical Informatics Research. http://protegewiki.stanford.edu/wiki/Protege-OWL
[letzter Besuch: 14.08.2015]

[Stan15g] *Protégé: Choosing between versions of Desktop Protégé*. Homepage of the Stanford Center for Biomedical Informatics Research.
http://protegewiki.stanford.edu/wiki/Protege4Migration
[letzter Besuch: 29.08.2015]

9. Anhang

Der erste Teil des Anhangs beinhaltet die Installationsanleitung, die beschreibt, wie das in dieser Arbeit erstellte Plug-In in Protégé eingebunden werden muss, um es darstellen zu können. Des Weiteren wird unter Punkt A.2 auf die verwendete Version von Protégé 3.5 und die Entwicklungsumgebung Eclipse SDK 4.2.1 näher eingegangen. Zum Abschluss werden noch erwähnenswerte Funktionen der Implementierung beschrieben.

A.1 Installationsanleitung

Das im Zuge dieser Arbeit erstellte Plug-In für Protégé wurde als Eclipse Projekt erstellt und ist in dem beiliegenden Ordner dieser Arbeit „mproject“ gespeichert. In diesem Verzeichnis bzw. den Unterverzeichnissen befinden sich alle Dateien die zum Einbinden und Ausführen des Plug-Ins in Protégé notwendig sind. Weiters enthält dieser Ordner noch das mit dem Plug-In erstellte durchgängige Beispiel und eine Beschreibung der Implementierung durch Javadoc. Im Nachfolgenden werden die einzelnen Schritte beschrieben, die notwendig sind, um das Plug-In in Protégé einbinden zu können.

Schritt 1:

Download von Protégé 3.5 von der Protégé Homepage unter http://protege.stanford.edu/download/protege/3.5/installanywhere/Web_Installers/ und Installation in ein gewünschtes Verzeichnis.

Schritt 2:

Kopieren des Ordners „mproject“ in das Installationsverzeichnis von Protégé unter „…\Protege_3.5_beta\plugins“. Im Verzeichnis „mproject“ sind sämtliche Dateien, die zum Ausführen des Plug-Ins benötigt werden, gespeichert. Dieses Verzeichnis beinhaltet auch als Subverzeichnis „meta-Inf“ mit der Datei „manifest.mf“, die zum Einbinden in Protégé benötigt wird und bereits vorkonfiguriert ist.

Schritt 3:

Starten von Protégé und Erstellen eines neuen Projektes mit dem Projektyp „Protégé Files (.pont and .pins)“. Optional dazu kann auch das bestehende Beispielprojekt, indem das in dieser Arbeit durchgängige Beispiel dargestellt wird, geöffnet werden. In diesem Fall muss beim Start die Funktion „Open Other“ und anschließend die Datei „…\mproject\mobject-mrelationship-beispiel.pprj“ ausgewählt werden.

Schritt 4:

Damit Protégé das Plug-In anzeigen kann, muss direkt in Protégé der Menüpunkt „project\configure“ ausgewählt werden. Es erscheint ein Dialog mit den verfügbaren Plug-Ins, wobei hier das „MObjectMRelationshipTab“ ausgewählt werden muss. Dieser Schritt ist nur bei der Erstellung eines neuen Projektes nötig. Bei einem bestehendem Projekt, das bereits bearbeitet wurde, ist dieser Schnitt nicht mehr notwendig.

A.2 Verwendete Programmversionen

Unter diesem Punkt werden die verwendeten Werkzeuge und Programme im Zusammenhang mit der Implementierung des Plug-Ins in genauer beschrieben.

2.1 Eclipse SDK Version 4.2.1

Für die Programmierung des Plug-Ins wurde die Version „Eclipse SDK Version 4.2.1“ verwendet. Diese umfasst die Eclipse-Plattform, verschiedene Werkzeuge für die Java-Entwicklung, sowie eine Umgebung zur Entwicklung von Eclipse Plug-Ins. Im Nachfolgenden wird die Entwicklungsplattform Eclipse kurz beschreiben.

Was ist Eclipse?

Eclipse ist eine Plattform, die Werkzeuge für die Entwicklung von Softwareanwendungen bereitstellt. Dabei stellt Eclipse an sich, abgesehen von den Grundelementen zur Softwareentwicklung, nicht viel Funktionalität für den Benutzer zur Verfügung, sondern der Wert der Plattform liegt in der Erweiterbarkeit durch Plug-Ins. Eclipse an sich ist also nur der Kern der zuständig ist für die dynamische Entdeckung, das Laden und den Betrieb der Plug-

Ins, wobei für Eclipse derzeit bereits über tausend verschiedene Plug-Ins bereitstehen, die über den „Eclipse marketplace“ unter https://marketplace.eclipse.org erhältlich sind. Für die Einbindung der Plug-Ins stellt Eclipse eine gemeinsame Benutzerschnittstelle zur Verfügung, wobei die Entwicklungsumgebung offen gestaltet wurde, sodass sie für die verschiedensten Betriebssysteme eingesetzt werden kann. [Ecli15a] [Ecli15b]

Entstehung von Eclipse und der „Eclipse Foundation“

Eclipse bzw. das „Eclipse Project“ wurde ursprünglich im Jahr 2001 von IBM mit Unterstützung der Softwareanbieter Borland, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft und Wegbain ins Leben gerufen. Im Jahr 2004 wurde die „Eclipse Foundation“ als unabhängige Non-Profit-Oranisation gegründet, damit sich eine herstellerneutrale, offene und transparente „Eclipse community“ entwickeln konnte. Diese Gemeinschaft besteht mittlerweile aus vielen Mitgliedern, die hauptsächlich im Bereich der Softwareindustrie tätig sind. Zur Unterstützung der „Eclipse community“ bietet die „Eclipse Foundation“ folgende Dienste an:

- IT-Infrastruktur: Die IT-Infrastruktur für die Open-Source Gemeinschaft wird von der „Eclipse Foundation“ verwaltet. Es werden bspw. entwicklungsorientierten Foren, Downloadseiten, Webseiten, Bugzilla Datenbanken, etc. für die Gemeinschaft bereitgestellt.
- IP-Management: Ein sehr wichtiger Aspekt von Eclipse ist, dass alle Softwareprodukte, die mit Eclipse erstellt werden und erstellt wurden, lizensiert sind unter der „Eclipse Public License (EPL)“. Dies ermöglicht es Softwareherstellern die Open-Source Technologie Eclipse zu nutzen, um ihre eigenen kommerziellen Softwareprodukte herzustellen.
- Entwicklungsunterstützung für die „ Eclipse community“: Die „Eclipse Foundation“ hilft der Gemeinschaft qualitativ hochwertige Software zu entwickeln, indem sie Dienstleistungen zur Verfügung stellt, die bspw. zu einer besseren Koordination und Interaktion innerhalb eines Softwareprojektes beitragen. [Ecli15b]
- Ecosystem Entwicklung: Ein zentraler Punkt der „Eclipse community“ und der „Eclipse Foundation“ ist die aktive Vermarktung und Bewerbung von Eclipse Projekten und in

weiterer Folge des „Eclipse Ecosystems“. Dieses Ökosystem, das weit über die „Eclipse community“ hinaus geht, beinhaltet bspw. kommerzielle Produkte die mit Eclipse erstellt wurden, Magazine, Online-Portale, Bücher, etc.

Eclipse Versionen

Wie bereits erwähnt, wurde für die Programmierung des Plug-Ins die Version „Eclipse SDK Version 4.2.1“ verwendet, die seit 2012 erhältlich ist und auch als „Eclipse Juno“ bezeichnet wird. Seit 2006 wurden zehn Grundversionen von Eclipse veröffentlicht, wobei die derzeit aktuelle Version 4.5 „Eclipse Mars“ seit Juni 2015 angeboten wird. Sie ist, sowie auch schon die Versionen zuvor, als Grundversion odert in weiteren Ausführungen mit verschiedenen Plug-In Paketen erhältlich. [Ecli15c]

Genauere Informationen über die verschiedenen Versionen von Eclipse bzw. die einzelnen Eclipse Versionen selbst, stehen zum freien Download unter https://eclipse.org/downloads/packages/all zur Verfügung.

2.2 Protégé-Frames Version 3.5

Das in dieser Arbeit implementierte Plug-In wurde für die Version „Protégé-Frames 3.5“ erstellt, die seit 2013 erhältlich ist. Diese Version, die Nachfolgerversionen sowie der Source Code von Protégé sind frei verfügbar im Downloadbereich auf der Homepage der Stanford Universität unter http://protegewiki.stanford.edu/wiki/Protege_Desktop_Old_Versions. Zwischen diesen Versionen gibt es allerdings große Unterschiede, wobei jede Version gewisse Vor- und Nachteile aufweist. Die Wahl der richtigen Version hängt davon ab, welche für das zu erstellende Projekt am besten geeignet ist. Beispielsweise ist die beste Wahl für die Erstellung von framebasierten Ontologien die Version 3.5, wohingegen für reine OWL Anwendungen eine Version der Generation ab 4.0 empfohlen wird. [Stan15g]

Da bereits auf Protégé-Frames in Kapitel 3 ausführlich eingegangen wurde, wird an dieser Stelle nur mehr kurz auf die wichtigsten Eigenschaften bzw. Unterschiede der verwendeten Version 3.5 und der Nachfolgeversion 4.3 eingegangen.

- Frames Support: Die Version 3.5 beinhaltet den „Protégé-Frames Editor“. In der Version 4.3 wird derzeit noch keine Unterstützung für Frames angeboten.
- OWL Unterstützung: Beide Versionen unterstützen OWL und SWRL, wobei Version 3.5 OWL 1.0 und die Version 4.3 OWL 2.0 verwendet. Abfragen mit SPARQL sind nur in Version 3.5 verfügbar.
- Plug-Ins: Für alle Versionen gibt es zahlreiche Plug-Ins, die sowohl von der Stanford Universität als auch von der Protégé Gemeinschaft entwickelt wurden. Die Versionen unterscheiden sich darin, dass es in Version 4.3 mehr Möglichkeiten gibt, Plug-Ins für Protégé zu erstellen.
- User Interface: Die Darstellung der Benutzeroberfläche in Version 3.5 wird hauptsächlich durch Tab- und Slotwidgets realisiert, wobei mittels diesen und durch Veränderung des Metamodells die Benutzerschnittstelle neu gestaltet werden kann. In Versionen 4.3 bestehen alle Elemente der Benutzeroberfläche aus Plug-Ins und somit kann die gesamte Benutzerschnittstelle neu konfiguriert werden.
- Multi-User Support: In Version 3.5 können mehrere Benutzer gleichzeitig dieselbe Ontologie bearbeiten. In der Version 4.3 ist eine Unterstützung für mehrere Benutzer derzeit noch nicht realisiert.
- Database Storage Model: Version 3.5 bietet die Möglichkeit Ontologien in JDBC-Datenbanken zu speichern, wohingegen Version 4.3 derzeit noch keine Unterstützung in dieser Hinsicht bietet. [Stan15g]

A.3 Ausgewählte Teile der Implementierung

Die wichtigsten Funktionen der Implementierung, die für das Erstellen, Löschen und Ändern von M-Objects, M-Relationships, Level, ConnectionLevel und Attribute zuständig sind, wurden in der Implementierung in der Klasse „MMainUtilities.java“ zusammengefasst. Nachfolgend wird eine Auswahl dieser Funktionen kurz durch den Inhalt der erstellten Javadocs beschrieben.

Method	createMObject(java.lang.String newname, edu.stanford.smi.protege.model.Instance parent)
Modifier and Type	static edu.stanford.smi.protege.model.Instance
Desription	This function creates a new M-Object.
Parameter	newname - is the name for the new M-Object parent - is the parent M-Object of the new M-Object
Return	the new M-Object

Tabelle 8: Javadoc - Methode *createMObject*

Method	createMObjectAndLevel(java.lang.String newmobjectname, java.lang.String newlevelname, edu.stanford.smi.protege.model.Instance parent)
Modifier and Type	static edu.stanford.smi.protege.model.Instance
Desription	This function creates a new M-Object and a new Level.
Parameter	newmobjectname - is the new name of the MObject newlevelname - is the new Name oft new Level parent - is the parent M-Object of the new M-Object
Return	the new M-Object

Tabelle 9: Javadoc - Methode *createMObjectAndLevel*

Method	deleteMObject(edu.stanford.smi.protege.model.Instance delins)
Modifier and Type	static void
Desription	This function deletes a M-Object.
Parameter	delins - is the current M-Object
Return	-

Tabelle 10: Javadoc - Methode *deleteMObject*

Method	renameMObject(edu.stanford.smi.protege.model.Instance mobject, java.lang.String newname)
Modifier and Type	static edu.stanford.smi.protege.model.Instance
Desription	This function renames an M-Object.
Parameter	mobject - is the current M-Object newname - is the new name for the M-Object
Return	the renamed M-Object

Tabelle 11: Javadoc - Methode rename*MObject*

Method	createMRelationship(edu.stanford.smi.protege.model.Instance parent, java.lang.String label, edu.stanford.smi.protege.model.Instance source, edu.stanford.smi.protege.model.Instance target)
Modifier and Type	static edu.stanford.smi.protege.model.Instance
Desription	This function creates a M-Relationship
Parameter	parent - is the parent M-Relationship of the new M-Relationship label - is the label of the new M-Relationship source - is the source M-Object of the new M-Relationship target - is the target M-Object of the new M-Relationship
Return	the new Level

Tabelle 12: Javadoc - Methode *createMRelationship*

Method	deleteMRelationship(edu.stanford.smi.protege.model.Instance mrelationship)
Modifier and Type	static java.util.ArrayList<edu.stanford.smi.protege.model.Instance>
Desription	This function deletes a M-Relationship.
Parameter	mrelationship - is the current M-Relationship
Return	a list of deleted M-Relationships

Tabelle 13: Javadoc - Methode *deleteMRelationship*

Method	renameMRelationship(edu.stanford.smi.protege.model.Instance mrelationship, java.lang.String newname)
Modifier and Type	static edu.stanford.smi.protege.model.Instance
Desription	This function renames a M-Relationship.
Parameter	mrelationship - is the current M-Relationship newname - is the new name of M-Relationship
Return	the renamed M-Relationship

Tabelle 14: Javadoc - Methode *renameMRelationship*

Method	createLevel(java.lang.String newlevelname, edu.stanford.smi.protege.model.Instance mobject, edu.stanford.smi.protege.model.Cls atposition)
Modifier and Type	static java.util.ArrayList<edu.stanford.smi.protege.model.Instance>
Desription	This function creates a new Level.
Parameter	newlevelname - is the new label for the Level mobject - is the M-Object at which the Level becomes introduced atposition - is the class above at which the new Level becomes introduced
Return	an ArrayList of new M-Objects which become created during process of creating a new Level

Tabelle 15: Javadoc - Methode *createLevel*

Method	deleteLevel(edu.stanford.smi.protege.model.Instance deletelevel, edu.stanford.smi.protege.model.Instance definedAtMObject)
Modifier and Type	static java.util.ArrayList<edu.stanford.smi.protege.model.Instance>
Desription	This function deletes a Level.
Parameter	deletelevel - is the currentLevel definedAtMObject - is the M-Object where the Level was introduced

Return	an ArrayList of M-Objects which become deleted during process of deleting a Level

Tabelle 16: Javadoc - Methode *deleteLevel*

Method	`renameLevel(`
	`edu.stanford.smi.protege.model.Cls topcls,`
	`edu.stanford.smi.protege.model.Instance oldlevel,`
	`java.lang.String newname)`

Modifier and Type	`static void`
Desription	This function renames a Level.
Parameter	`topcls` - is the M-Object-Class where the Level was introduced `oldlevel` - is the current Level `newname` - is the new name for the Level
Return	-

Tabelle 17: Javadoc - Methode *renameLevel*

Method	`createConnectionLevel(`
	`edu.stanford.smi.protege.model.Instance mrelationship,`
	`edu.stanford.smi.protege.model.Instance source,`
	`edu.stanford.smi.protege.model.Instance target)`

Modifier and Type	`static edu.stanford.smi.protege.model.Instance`
Desription	This function creates a connectionLevel.
Parameter	`mrelationship` - is the current M-Relationship `source` - is the source of the ConnectionLevel `target` - is the target of the ConnectionLevel
Return	the new ConnectionLevel

Tabelle 18: Javadoc - Methode *createConnectionLevel*

Method	**deleteConnectionLevel**(edu.stanford.smi.protege.model.Instance connectionlevel)
Modifier and Type	static void
Desription	This function deletes a connectionLevel.
Parameter	connectionlevel - is the current ConnectionLevel
Return	-

Tabelle 19: Javadoc - Methode *deleteConnectionLevel*

Method	**createOwnAttribute**(edu.stanford.smi.protege.model.Instance currentins)
Modifier and Type	static edu.stanford.smi.protege.model.Slot
Desription	This function creates a new attribute.
Parameter	currentins - is the current M-Object attributename - is the name of the attribute
Return	the new attribute

Tabelle 20: Javadoc - Methode *createOwnAttribute*

Method	**createTemplateAttribute**(edu.stanford.smi.protege.model.Cls currentcls, java.lang.String attributename)
Modifier and Type	static edu.stanford.smi.protege.model.Slot
Desription	This function creates a new attribute for a M-Object-Class.
Parameter	currentcls - is the current M-Object-Class attributename - is the name of the attribute
Return	the new attribute

Tabelle 21: Javadoc - Methode createTemplateAttribute

Method	**deleteAttribute**(edu.stanford.smi.protege.model.Slot currentattribute)
Modifier and Type	static void
Desription	This function deletes an attribute.
Parameter	currentattribute - is the current attribute
Return	-

Tabelle 22: Javadoc - Methode *deleteAttribute*

Method	**changeAttribute**(edu.stanford.smi.protege.model.Instance currentins, edu.stanford.smi.protege.model.Slot currentslot, java.lang.String slotname, java.lang.String slotvalue, java.lang.String valuetype, java.lang.String documentation)
Modifier and Type	static edu.stanford.smi.protege.model.Slot
Desription	This function sets different values of a slot.
Parameter	currentins - is the instance to which the slot is connected currentslot - is the current slot slotname - is the name of the slot valuetype - is the value type of the slot defaultvalue - is the default value of the slot documentation - is the documentation of the slot
Return	the current attribute

Tabelle 23: Javadoc - Methode *changeAttribute*

Method	**changeParentMObject**(edu.stanford.smi.protege.model.Instance newparent, edu.stanford.smi.protege.model.Instance mobject, edu.stanford.smi.protege.model.KnowledgeBase base)
Modifier and Type	static void
Desription	This function changes the parent of the M-Object.

Parameter	`newparent` - is the new parent M-Object `mobject` - is the M-Object, whose parent is going to change `base` - is the KnowledgeBase
Return	-

Tabelle 24: Javadoc - Methode *changeParentMObject*

Method	`createImportantClasses(` `edu.stanford.smi.protege.model.Project project)`
Modifier and Type	`static void`
Desription	This function generates the important classes like MObjectClass, MObject, MRelationship, etc.
Parameter	`project` - is the current project
Return	-

Tabelle 25: Javadoc - Methode *createImportantClasses*