



Tutorial

04.10.2010

Manuel Renz & Matthias Ziegler



XENOVATION

1 Inhaltsverzeichnis

1	Inhaltsverzeichnis	2
2	License	4
3	Einleitung	9
1	Installation der Umgebung	10
1.1	Voraussetzungen	10
1.1.1	Mindest-Systemvoraussetzungen	10
1.2	Installation des Enterprise Architect	11
1.3	Installation und Konfiguration der Entwicklungsumgebung	11
1.3.1	Vorbereitungen	11
1.3.2	Installation des JDKs	12
1.3.3	Installation von Eclipse	12
1.3.4	Installation von openArchitectureWare	12
1.3.5	Einstellen des Workspace Verzeichnisses	12
1.3.6	Einstellen des Java Compiler compliance Levels	13
1.3.7	Installation weiterer Plug-Ins	15
1.3.8	Bereitstellen der Enterprise Architect API	16
1.4	Import der bereitgestellten Cartridges	17
1.4.1	Importieren der Cartridges	19
1.4.2	Starten der Generatoren	20
3	Anwendung des BVA-Generators anhand eines Beispiels	22
3.1	Erstellen des Beispiel Modells	22
3.2	Anwendung der Standard BVA Stereotypen	24
3.3	Anwendung Benutzerdefinierter Validatoren	25
3.3.1	Erstellen eines benutzerdefinierten BVA Profils	25
3.3.2	Erstellen eines benutzerdefinierten BVA Stereotypen	27
3.3.3	Verwenden des benutzerdefinierten BVA Profils	31
3.3.4	Implementieren des benutzerdefinierten Validators	31
3.4	Durchführen des Exports mit dem bva.exporter	34
3.4.1	Anpassen der exporter.properties	34
3.4.2	Starten des Exportvorgangs	35
3.5	Durchführen der Code Generierung mit dem bva.generator	37
3.6	Anmerkung zum Beispielmodell	38
4	Anwendung des JPA-Generators anhand eines Beispiels	40
4.1	Überblick	40
4.2	Erstellen des Beispiel Modells	41



4.2.1	Exportvorgang aus dem Enterprise Architect in XML	45
4.2.2	Generierung von Java-Code	48
4.2.3	DDL-Generierung	55
5	Tutorial für die gemeinsame Verwendung des JPA- und BVA-Generators	57
3.6.1	UML-Modell	57
3.6.2	Exportvorgang aus dem Enterprise Architect in XML	58
3.6.3	Generierung von Java-Code	58
4	Wichtige Hinweise zur Modellierung mit dem Enterprise Architect	60
4.1	Neues Enterprise Architect Projekt erstellen	60
4.2	Neue UML-Klasse erstellen	61
4.3	Attribute erstellen und bearbeiten	62
4.4	Methoden erstellen und bearbeiten	63
4.5	Beziehungen erstellen	64
4.6	UML-Profil einhängen	65
4.7	Stereotypen auf ein UML-Element anwenden und löschen	66
4.8	Tagged Values bearbeiten	67
4.9	Bekannte Bugs	69
4.9.1	Tagged Values mit Option <i>Show Fully Qualified Tags</i>	69
4.9.2	Mehrere gleiche Tagged Values für ein UML-Element	70
4.9.3	Beim Löschen von Stereotypen werden zugehörige Tagged Values nicht mit gelöscht	72
6	Abbildungsverzeichnis	74
7	Tabellenverzeichnis	76
8	Literaturverzeichnis	77
9	Stichwortverzeichnis	78

2 License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

- a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
- b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.
- c. **"Creative Commons Compatible License"** means a license that is listed at <http://creativecommons.org/compatiblelicenses> that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under that license under this License or a Creative Commons jurisdiction license with the same License Elements as this License.
- d. **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.
- e. **"License Elements"** means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.
- f. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
- g. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
- h. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in

dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

- i. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.
- j. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
- k. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
- b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
- c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
- d. to Distribute and Publicly Perform Adaptations.
- e. For the avoidance of doubt:
 - i. **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
 - ii. **Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor waives the exclusive right to collect such royalties for any exercise by You of the rights granted under this License; and,
 - iii. **Voluntary License Schemes.** The Licensor waives the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.
- b. You may Distribute or Publicly Perform an Adaptation only under the terms of: (i) this License; (ii) a later version of this License with the same License Elements as this License; (iii) a Creative Commons jurisdiction license (either this or a later license version) that contains the same License Elements as this License (e.g., Attribution-ShareAlike 3.0 US); (iv) a Creative Commons Compatible License. If you license the Adaptation under one of the licenses mentioned in (iv), you must comply with the terms of that license. If you license the Adaptation under the terms of any of the licenses mentioned in (i), (ii) or (iii) (the "Applicable License"), you must comply with the terms of the Applicable License generally and the following provisions: (I) You must include a copy of, or the URI for, the Applicable License with every copy of each Adaptation You Distribute or Publicly Perform; (II) You may not offer or impose any terms on the Adaptation that restrict the terms of the Applicable License or the ability of the recipient of the Adaptation to exercise the rights granted to that recipient under the terms of the Applicable License; (III) You must keep intact all notices that refer to the Applicable License and to the disclaimer of warranties with every copy of the Work as included in the Adaptation You Distribute or Publicly Perform; (IV) when You Distribute or Publicly Perform the Adaptation, You may not impose any effective technological measures on the Adaptation that restrict the ability of a recipient of the Adaptation from You to exercise the rights granted to that recipient under the terms of the Applicable License. This Section 4(b) applies to the Adaptation as incorporated in a Collection, but this does not require the Collection apart from the Adaptation itself to be made subject to the terms of the Applicable License.
- c. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and (iv) , consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.
- d. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to

the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

- a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
- b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
- c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
- f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional



rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

3 Einleitung

Dieses Tutorial besteht im Wesentlichen aus den Praxisteilen der Diplomarbeiten „Erstellung eines MDA-Generators für Validierungen auf Basis der Bean-Validation-API“ und „Erstellung eines MDA-Generators für CRUD-Operationen in einer Persistenzschicht auf Basis der Java Persistence API“, welche die Grundlage für droMDAry bilden. Die Kapitel wurden für das droMDAry Tutorial angepasst und zusammengefasst.

Das Tutorial gliedert sich in folgende Abschnitte:

- Installation und Konfiguration der verschiedenen Softwarepakete
- Verwendung der Generatoren (BVA/JPA)
- Hinweise zur Verwendung des Enterprise Architects

1 Installation der Umgebung

Prinzipiell lässt sich die Installation in folgende Punkte gliedern:

- Download und Installation der verschiedenen Softwarepakete
- Konfiguration der Eclipse Umgebung und installation diverser Plugins
- Import der Generatoren

1.1 Voraussetzungen

Die Entwicklung von droMDAry wurde unter Windows durchgeführt. Verwendete Software-Tools sowie auch der MDA-Generator funktionieren unter Windows XP / Vista / 7 einwandfrei. Die Kompatibilität mit Linux/Unix/Mac wurde nicht getestet. Für das Software-Werkzeug *Enterprise Architect* gibt es jedoch eine CrossOver-Version¹ für Linux und Mac. Weitere Hinweise dazu findet man auf der folgenden Homepage:

http://www.sparxsystems.com/support/faq/ea_on_linux.html

Falls die Entwicklungsumgebung Eclipse bereits installiert ist, sollte unbedingt geprüft werden ob die im Kapitel 1.3.3 und 1.3.7 angegebenen Anforderungen erfüllt werden, um sicherzustellen, dass alle benötigten Plug-Ins in der korrekten Version vorliegen.

Eclipse gibt es auch in einer Version für Linux und Mac. Mehr Informationen dazu findet man auf der Homepage von Eclipse unter

<http://www.eclipse.org/>.

1.1.1 Mindest-Systemvoraussetzungen

- CPU mit 1,8 GHz
- 1 GB Arbeitsspeicher
- 600 MB Festplattenspeicher

Diese Angaben gelten für das Windows Betriebssystem XP/Vista. Weitere Betriebssysteme wurden nicht getestet.

¹ Siehe dazu CrossOver im Literaturverzeichnis

1.2 Installation des Enterprise Architect

Die aktuelle Trial Version kann von der Homepage der Herstellers unter <http://www.sparxsystems.de/enterprise-architect/download-trial/> heruntergeladen werden.

Auf dieser Homepage gibt es die Möglichkeit, den Punkt „EA Trial Version“ anzuklicken, um sich die aktuelle Version des *Enterprise Architect* auf den eigenen Rechner zu speichern. Die Trial Version ist für 30 Tage gültig. Nach diesem Zeitraum kann mit dem Tool nicht mehr gearbeitet werden. Unter <http://www.sparxsystems.de/enterprise-architect/ea-price/> findet man alle nötigen Informationen zum Kauf des Werkzeugs *Enterprise Architect*.

Während der Implementierung der Generatoren wurde die **Version 7.5 Build 845** verwendet.

Nachdem der Installationsvorgang gestartet wurde, müssen nur die wenigen Anweisungen des Installationsassistenten bestätigt werden. Das Tool ist somit fertig installiert.

1.3 Installation und Konfiguration der Entwicklungsumgebung

Im folgenden Abschnitt werden die nötigen Schritte zur Installation einer voll funktionsfähigen Entwicklungsumgebung basierend auf *Eclipse* und *openArchitectureWare* erläutert.

1.3.1 Vorbereitungen

Bevor mit der Installation und Konfiguration begonnen werden kann, müssen alle benötigten Softwarekomponenten heruntergeladen worden sein. Im folgenden eine Liste der Pakete mit den entsprechenden Versionsnummern sowie eines Links zur Downloadseite des Herstellers.

1. Java SE Development Kit 6u14 (JDK)
<http://java.sun.com/javase/downloads/index.jsp>
2. Eclipse IDE for Java EE Developers 3.4 SR2 (Ganymede)
<http://www.eclipse.org>
3. openArchitectureWare 4.3.1 SDK (oAW) als ZIP Datei
<http://www.openarchitectureware.org/staticpages/index.php/download>

4. droMDAry-Alpha

<http://sourceforge.net/projects/dromdary/files/droMDAry-Alpha/droMDAry-alpha.zip/download>

1.3.2 Installation des JDKs

Die gesamte Toolchain ist auf eine funktionsfähige Java Umgebung angewiesen, daher muss diese zuerst installiert werden. Java muss mindestens in der Version 1.5 verfügbar sein. Neuere 1.6er Versionen sind laut „*openArchitectureWare*“ FAQ² ebenso unterstützt. Nichtsdestotrotz kann die aktuellste Version installiert werden, da das „*Compiler Compliance Level*“ im Nachgang auf 1.5 eingestellt werden kann.

Die Installation des JDK kann ohne weitere Anpassungen mit den Standardvorgaben des Installationsassistenten durchgeführt werden.

1.3.3 Installation von Eclipse

Die heruntergeladene ZIP Datei *eclipse-jee-ganymede-SR2-win32.zip* entpackt man in einen beliebigen Ordner. Die J2EE Edition wurde gewählt da sie schon einige der benötigten Plug-Ins beinhaltet und diese somit nicht mehr separat heruntergeladen werden müssen.

In dieser Beschreibung wird das Archive direkt nach *c:* entpackt, d.h. die Entwicklungsumgebung ist unter *c:\eclipse* zugänglich.

1.3.4 Installation von openArchitectureWare

Das heruntergeladene *openArchitectureWare* ZIP Archive wird in dasselbe Verzeichnis entpackt, in das auch das Eclipse Archive entpackt wurde, in diesem Fall also *c:*. Dadurch wird das *openArchitectureWare* Plug-In als neues Feature zur Eclipseinstallation hinzugefügt.

1.3.5 Einstellen des Workspace Verzeichnisses

Die Entwicklungsumgebung wird nun mittels *c:\eclipse\ecclipse.exe* gestartet.

Beim ersten Start wird automatisch ein neuer Workspace unter *c:\Dokumente und Einstellungen\<Benutzer>\workspace* angelegt. Da die Leerzeichen im Pfad später zu Problemen führen können, wird besser ein eigener Workspace in ei-

² Siehe dazu openArchitectureWare Website im Literaturverzeichnis

nem anderen Verzeichnis erstellt. Prinzipiell kann auch ein bereits bestehender Workspace verwendet werden. Sollte es dabei jedoch zu unvorhergesehenen Problemen kommen, empfiehlt es sich einen separaten Workspace anzulegen. Um dies zu erreichen wählt man über das Menü „*File → Switch Workspace → Other*“, den gewünschten Workspace-Ordner aus (siehe dazu Abbildung 1: Screenshot vom Workspace Launcher).

Im Folgenden wird angenommen, dass ein eigener, noch leerer Workspace als `c:\workspace` angelegt wurde.

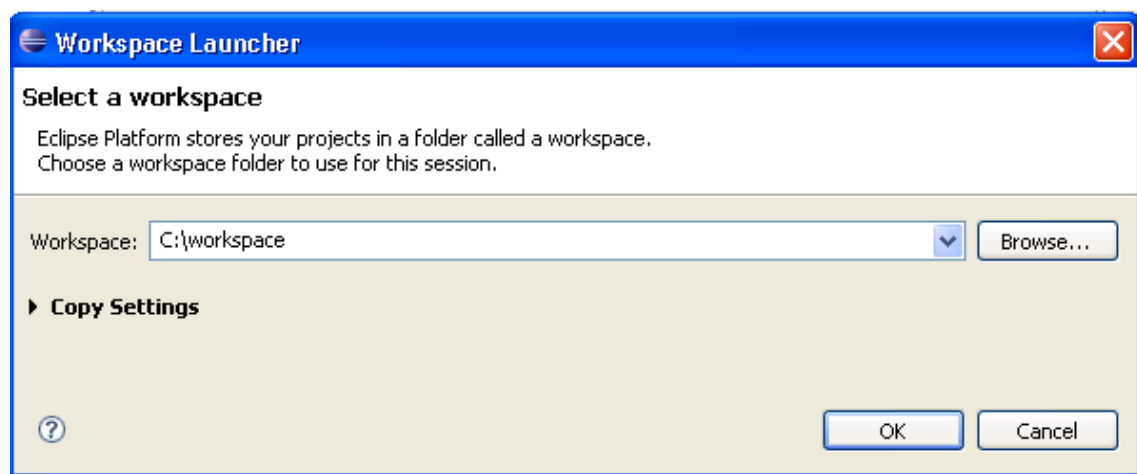


Abbildung 1: Screenshot vom Workspace Launcher

1.3.6 Einstellen des Java Compiler compliance Levels

Da die Toolchain nicht 100% mit Java 1.6 kompatibel ist, muss das Java Compiler Compliance Level auf 1.5 eingestellt werden. Dies geschieht im „*Preferences*“ Dialog den man über den Menüpunkt „*Window → Preferences*“ erreicht. Dort kann man unter dem Eintrag „*Java → Compiler → Compiler compliance level*“ die Java Version auf die Version 1.5 setzen (siehe dazu Abbildung 2: Screenshot vom Preferences Dialog des Java Compiler).

Mit dieser Einstellung wird die Java Version für den gesamten Workspace auf 1.5 festgelegt. Alternativ kann man dies auch mit den projektspezifischen Einstellungen bei jeder Cartridge separat einstellen. Dies kann im Einzelfall jedoch leicht einmal vergessen werden und ist daher recht fehleranfällig. Da in diesem Tutorial mit einem separaten Workspace gearbeitet wird, ist es ohnehin unproblematisch diese Einstellung für alle Projekte vorzugeben.

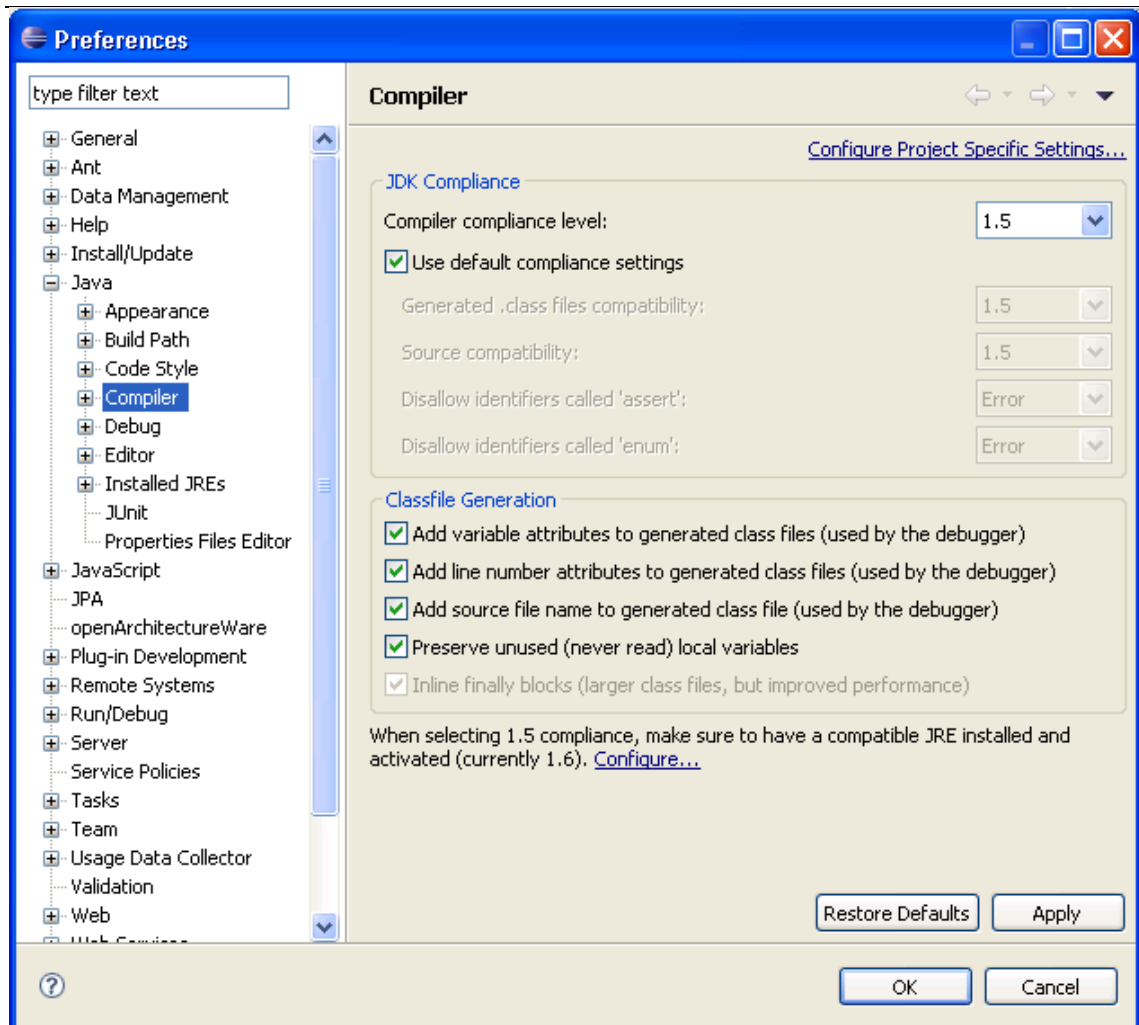


Abbildung 2: Screenshot vom Preferences Dialog des Java Compilers

In der verwendeten J2EE Eclipse Distribution sind schon die meisten benötigten Plug-Ins enthalten, die für ein funktionierendes *openArchitectureWare* benötigt werden. Dazu gehören:

- Eclipse Modeling Framework (EMF) in der Version 2.4.2
- Eclipse Web Standard Tools (WST) in der Version 3.0.2
- Eclipse Plug-in Development Environment (PDE) in der Version 3.4.2

Damit *openArchitectureWare* jedoch mit UML2 bzw. XMI Daten umgehen kann, müssen noch zwei weitere Plug-Ins installiert werden. Diese Plug-Ins können mit dem integrierten Update Manager von der Eclipse Ganymed Update Site heruntergeladen werden. Dazu öffnet man über das Menü „*Help* → *Software Updates*“ den „*Software Updates und Add-ons*“ Dialog und aktiviert das Register „*Available Software*“. In der nun dargestellten Liste müssen die Einträge „*Ga-*

„nymede Update Site“ und darunter „Model and Model Development“ geöffnet werden. Hier wählt man nun die Plug-Ins „UML2 End-User Features 2.2.2“ sowie „UML2 Extender SDK 2.2.2“ aus und installiert diese per Klick auf „Install“ (siehe dazu Abbildung 3: Screenshot vom Eclipse Software Update Manager).

Die Plug-Ins werden nun automatisch heruntergeladen und installiert. Wenn dieser Vorgang erfolgreich abgeschlossen wurde, wird man aufgefordert den Eclipse erneut zu starten, was man mit einem Klick auf „Yes“ bestätigt.

1.3.7 Installation weiterer Plug-Ins

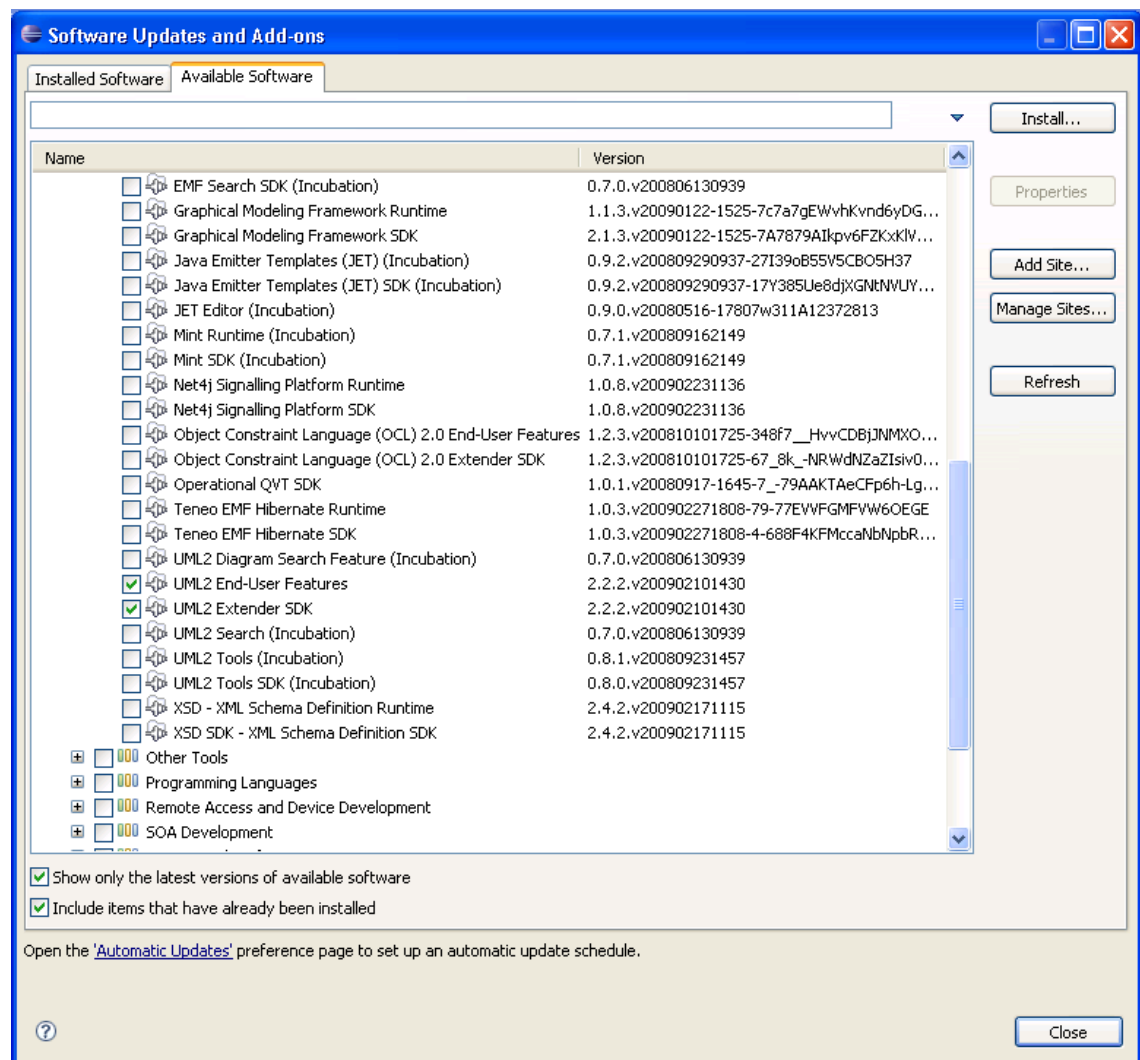


Abbildung 3: Screenshot vom Eclipse Software Update Manager

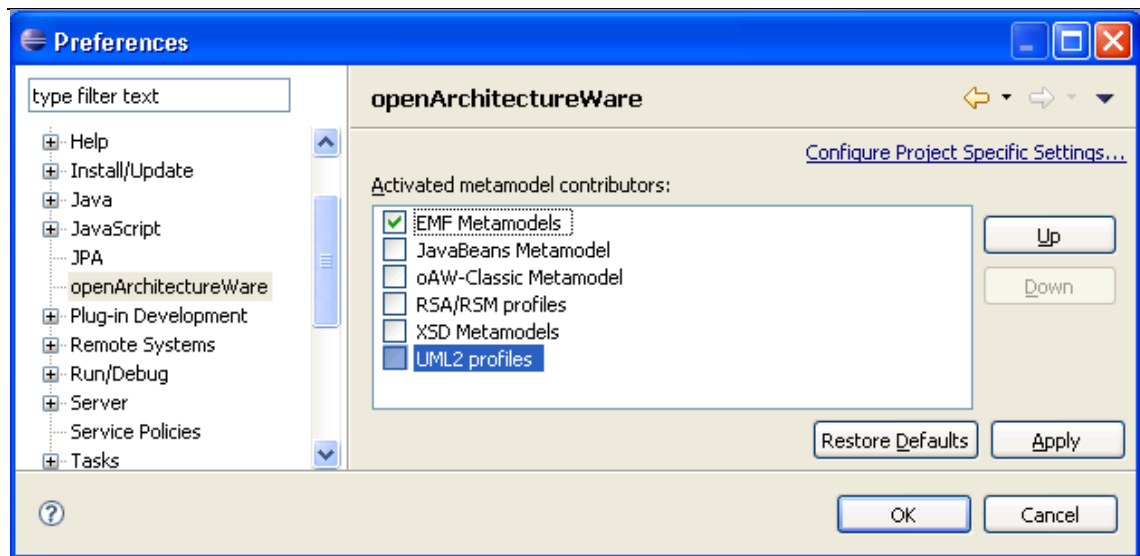


Abbildung 4: Screenshot vom Preferences Dialog zum Verwalten der verfügbaren Metamodelle in *openArchitectureWare*

Hier muss nun der Eintrag „UML2 profiles“ aktiviert und an die erste Stelle verschoben werden (siehe dazu Abbildung 4: Screenshot vom Preferences Dialog zum Verwalten der verfügbaren Metamodelle in *openArchitectureWare* und Abbildung 5: UML2 Profile in openArchitectureWare aktivieren).

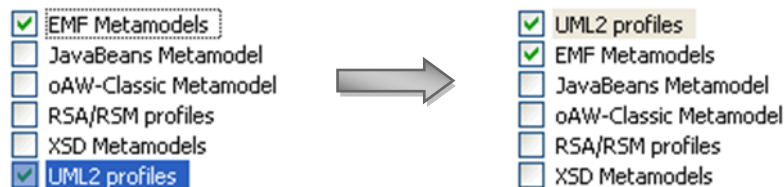


Abbildung 5: UML2 Profile in openArchitectureWare aktivieren

1.3.8 Bereitstellen der Enterprise Architect API

Damit der eingesetzte UML2 Exporter über das *Enterprise Architect* API auf die Projektdateien zugreifen kann, muss die DLL die diese API zur Verfügung stellt, in einem Verzeichnis verfügbar sein auf das über die Umgebungsvariable PATH zugegriffen werden kann.

Die benötigte DLL liegt im *Enterprise Architect* Installationsverzeichnis, also „c:\Program Files\Sparx Systems\EA\Java API“ oder „c:\Program Files\Sparx Systems\EA Trial\Java API“, heißt „SSJavaCOM.dll“ und kann bspw. einfach in das System32 Verzeichnis der Windowsinstallation kopiert werden.

Nun sind alle Voraussetzungen an die Umgebung erfüllt um die bereitgestellten Cartridges benutzen zu können.

1.4 Import der bereitgestellten Cartridges

Im letzten Kapitel wurde die Installation und Konfiguration der Entwicklungsumgebung erläutert. Jetzt können die Cartridges, die unter <http://sourceforge.net/projects/dromdary/files/droMDAry-Alpha/droMDAry-alpha.zip/download> zu finden sind, in den zuvor angelegten Eclipse Workspace importiert werden.

Insgesamt müssen sechs Cartridges in den Workspace importiert werden.

- bva.exporter
- bva.generator
- jpa.exporter
- jpa.generator
- common.exporter
- common.generator

Die Exporter Cartridges sind jeweils für den Export der *Enterprise Architect* Modelle zuständig und die Generator Cartridges für die Generierung des Java Quellcodes aus den zuvor exportierten Modellen.

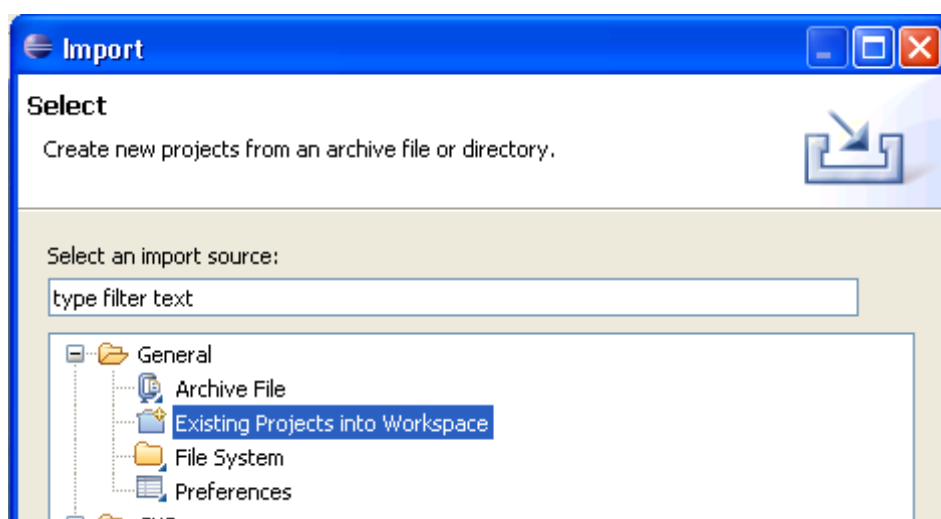


Abbildung 6: Screenshot des Importassistenten

Da es sich um zwei getrennte Cartridges handelt, die einzeln lauffähig sind, gibt es die Exporter und Generatoren jeweils für beide Cartridges als separate Projekte.

Die „Common“ Cartridges haben hierbei eine Sonderfunktion, denn sie verbindet dir BVA und JPA Cartridges für den Fall, dass aus einem *Enterprise Architect* Modell in einem Durchlauf beide Generate, also BVA und JPA, erzeugt werden sollen.

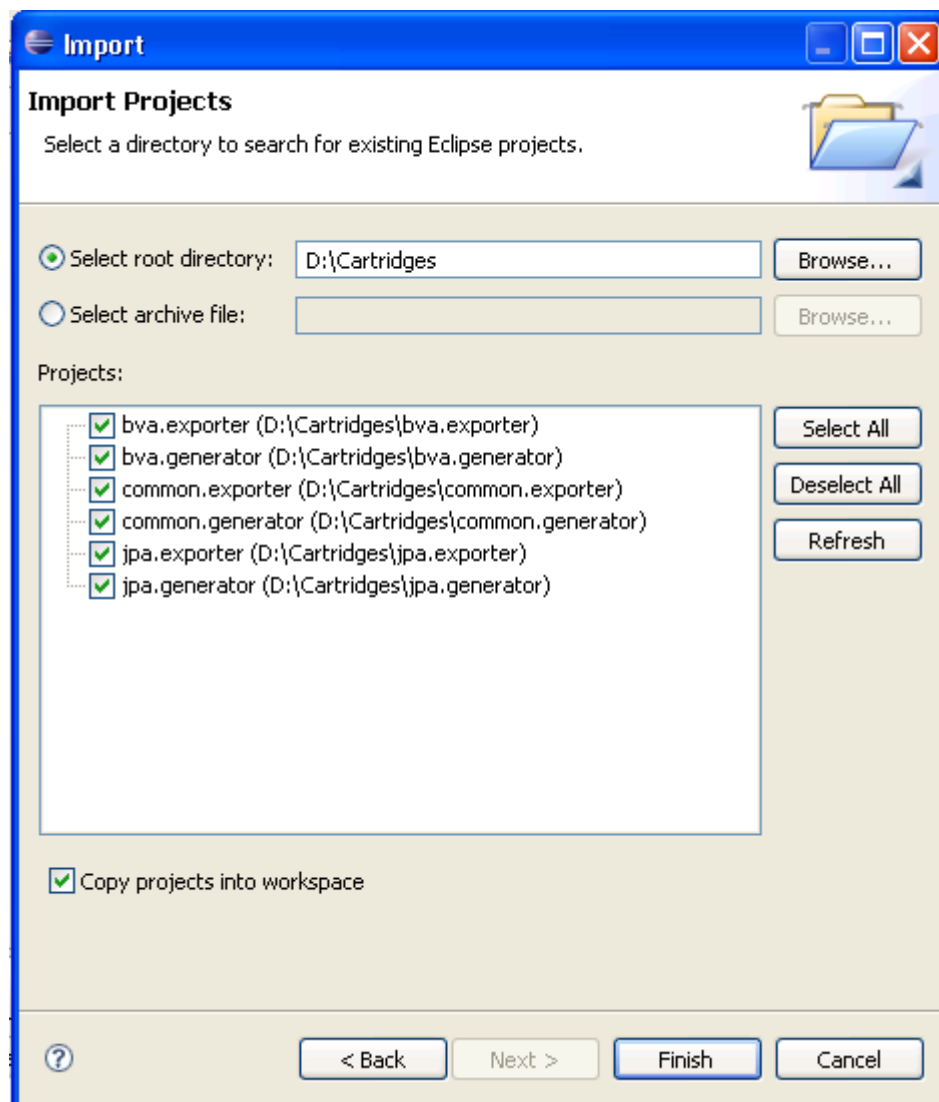


Abbildung 7: Screenshot des Assistenten zum auswählen der Cartridges

1.4.1 Importieren der Cartridges

Über den Menüpunkt „File → Import“ wird der Importassistent aufgerufen. Dort öffnet man den Eintrag „General“ und wählt „Existing Projects into Workspace“ aus (siehe dazu Abbildung 6: Screenshot des Importassistenten).

Mit einem Klick auf „Next“ gelangt man zur Auswahl der zu importierenden Projekte. Die Option „Select root directory“ ist bereits aktiviert, so dass man nun mit einem Klick auf „Browse“ den Ordner der zuvor von der droMDAry Homepage heruntergeladenen Cartridges auswählen kann und diesen mit einem Klick auf OK bestätigt.

Nun werden alle verfügbaren Eclipse Projekte die sich im ausgewählten Ordner befinden aufgelistet. Da die Projektdateien von Eclipse nicht nur verknüpft sondern richtig in das Workspace Verzeichnis kopiert werden sollen, muss die Option „Copy into workspace“ aktiviert sein (siehe dazu Abbildung 7: Screenshot des Assistenten zum auswählen der Cartridges). Mit einem Klick auf „Finish“ wird der Importvorgang gestartet. Alle Projekte werden nun im Workspace angelegt und die dazugehörige Dateien kopiert. Sobald dieser Vorgang abgeschlossen ist sollten die Cartridges im „Project Explorer“ angezeigt werden (siehe dazu Abbildung 9: Screenshot des Project Explorers nach dem erfolgreichen Import).

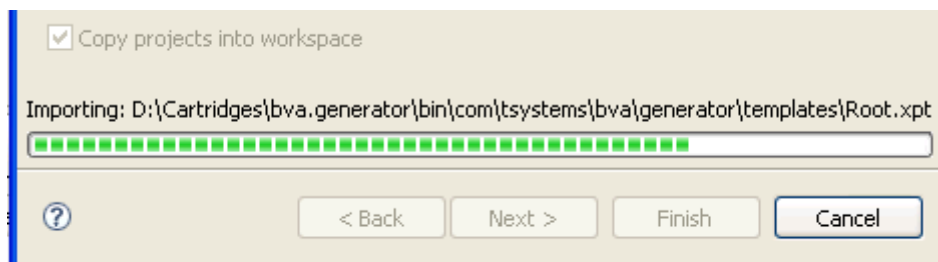


Abbildung 8: Screenshot des Importvorgangs

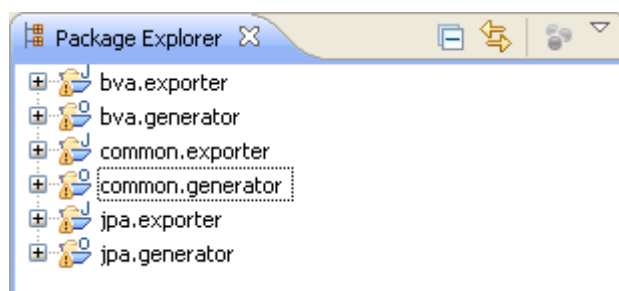


Abbildung 9: Screenshot des Project Explorers nach dem erfolgreichen Import

Nach dem erfolgreichen Import steht ein bereits Demonstrations Modell zur Verfügung, mit dem die Umgebung auf Funktionsfähigkeit geprüft werden kann.

1.4.2 Starten der Generatoren

Die Code Generierung ist in zwei Schritte aufgeteilt, zuerst muss das EA Modell in EMF exportiert werden und dann im zweiten Schritt wird basierend auf dem EMF Modell der Code generiert.

Der Exportvorgang für die Profile wird mit einem Rechtsklick auf die entsprechende Workflow Datei (*common.exporter/oaw/profile.oaw*) und anschließend mit einem Klick auf „Run As... → oAW Workflow“ gestartet. In der Eclipse Konsole kann der weitere Fortschritt beobachtet werden. Sobald der Workflow abgeschlossen ist erscheint folgende Meldung im Konsolenfenster.

```
2      75061 INFO WorkflowRunner      - workflow completed in 74026ms!
```

Hinweise: Sollte sich eine Demo Version des *Enterprise Architects* im Einsatz befinden, öffnet sich im Hintergrund mehrmals eine Meldung, die per Klick auf *Continue Trial* bestätigt werden muss.

Um den Export des eigentlichen Modells zu starten, muss nun auch noch die Datei *common.exporter/oaw/workflow.oaw* gestartet werden.

Jetzt liegen alle benötigten Modell Information im EMF vor und die Quellcode-generierung kann gestartet werden.

Dazu wird die Datei *common.generator/src/main/resources/workflow.oaw* auf dieselbe Weise wie zuvor gestartet. Nun werden die abstrakten sowie die Interfaceklassen im Verzeichnis */src/generated/java* mit samt den Packages generiert. Die Implementierungsklassen werden hingegen einmalig unter */src/main/java* erstellt. Bei allen weiteren Generierungsvorgängen werden diese Dateien nicht überschrieben um manuelle Änderungen am Quellcode zu schützen. Wenn diese Klassen auch neugeneriert werden sollen, müssen diese manuell gelöscht werden.

Per default werden die BVA und JPA Generatoren ausgeführt, die kann jedoch auf einfache Weise in der *common.generator/src/main/resources/workflow.properties* Datei ab Zeile 32 angepasst werden.

Auch hier kann der Vorgang in der Eclipse Konsole beobachtet werden. Die erfolgreiche Generierung wird mit der Meldung



```
INFO - Written 9 files to outlet [default] (./src/generated/java)
INFO - workflow completed in 3335ms!
```

quittiert.

3 Anwendung des BVA-Generators anhand eines Beispiels

In diesem Kapitel wird anhand eines anschaulichen Beispiels die Verwendung des BVA Generators erklärt. Dies geschieht in mehreren Schritten. Zunächst wird ein Modell erstellt, in welchem die BVA Stereotypen angewendet werden sollen. Dabei wird zum einen ein Standardvalidator verwendet und zum anderen wird erklärt wie ein benutzerdefinierter Validator erstellt und verwendet wird. Dann wird in einem weiteren Schritt das Modell mit dem *common.exporter* in XML transformiert sowie mit dem *common.generator* der Quellcode generiert.

3.1 Erstellen des Beispiel Modells

Zuerst muss ein UML-Modell erstellt werden das als Basis für die Generierung genutzt werden soll. Als Grundlage für dieses Beispiel soll ein Klassenmodell dienen das einen einfachen Buchkatalog beschreibt. Dieser Buchkatalog soll aus drei Klassen bestehen:

- *Autor*
- *Verlag*
- *Buch*

Es ist hierbei nicht das Ziel die Beziehung zwischen den Klassen im Detail zu modellieren und zu generieren, da dies die Aufgabe des JPA Generators ist.

Das fertige Modell, wie es in dieser Anleitung erstellt wird, ist bereits im Workspace unter „*bva.exporter/Model/BVA_Tutorial.EAP*“ zu finden.

Für die einzelnen Klassen sollen die Attribute wie folgt angelegt werden (Hinweise zur Modellierung mit dem *Enterprise Architect* sind im Kapitel 4 zu finden):

Autor

- *alter* (int)
- *nachname* (String)
- *vorname* (String)
- *strasse* (String)
- *plz* (String)

Verlag

- *strasse* (String)
- *verlagName* (String)
- *plz* (String)

Buch

- *auflage* (int)
- *buchId* (int)
- *titel* (String)
- *isbn* (String)

Wenn das Modell fertiggestellt ist sollte es ähnlich wie in Abbildung 10 dargestellt, aussehen.

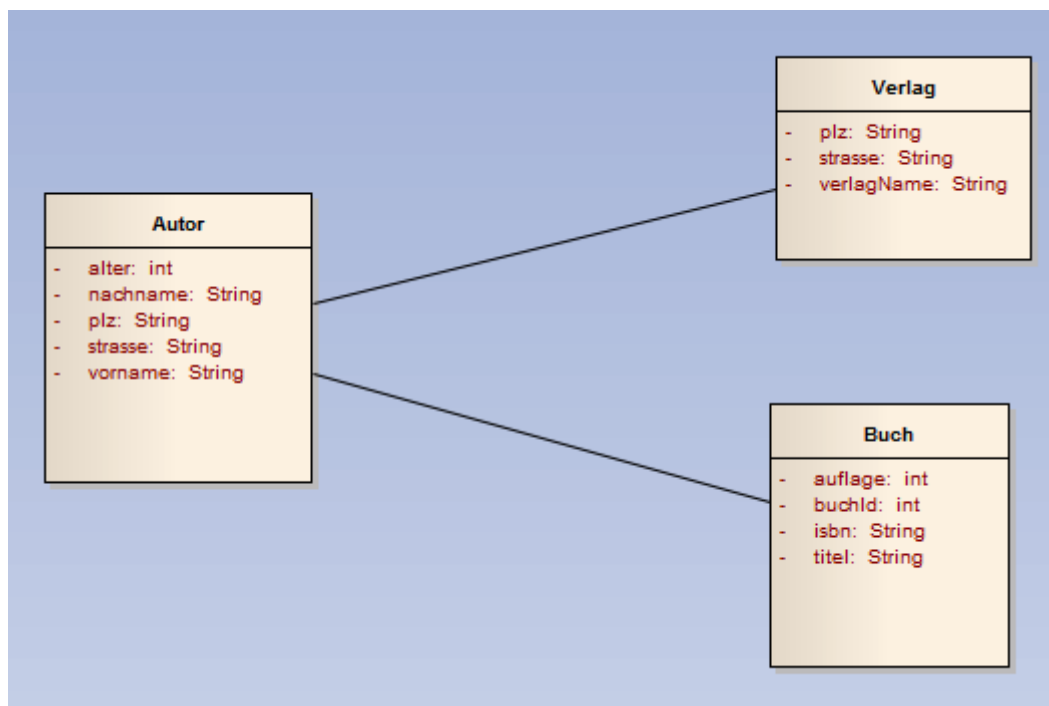


Abbildung 10: Einfaches UML-Modell für das BVA Beispiel

Um nun die Möglichkeiten der BVA Cartridge zu veranschaulichen, sollen folgende Attribute mit entsprechenden Regeln validiert werden:

- *Autor::plz* → genau 5 Stellen, bestehend aus Zahlen
- *Verlag::plz* → genau 5 Stellen, bestehend aus Zahlen
- *Buch::isbn* → 13 Stellen, nur Ziffern, mit Prüfsummenberechnung

3.2 Anwendung der Standard BVA Stereotypen

Zunächst muss das mitgelieferte Profil „SimpleJSR303“, welches droMDAry Package unter „/bva.exporter/Profil/SimpleJSR303_profile.xml“ verfügbar ist, in das Modell eingehängt werden. Eine ausführliche Anleitung zum Einhängen von UML Profilen findet man in Kapitel 4.6.

Nun stehen alle Standardvalidatoren über die „Ressource View“ des *Enterprise Architects* zur Verfügung und der Stereotyp „BVA_Pattern“ kann nun, wie in Kapitel 4.7. beschrieben, auf die Attribute *Autor::plz* und *Verlag::plz* angewendet werden.

Bei beiden *plz* Attributen wird nun mittels der Tagged Values des Stereotypen *BVA_Pattern*, die Validierungsregel für die Postleitzahl hinterlegt (Siehe dazu Tabelle 1: Tagged Values für die PLZ Validierung und Abbildung 11: Screenshot der Tagged Values des Stereotyp "BVA_Pattern").

Tag Name	Value
message	Invalid PLZ
regexp	[0-9]{5,5}

Tabelle 1: Tagged Values für die PLZ Validierung

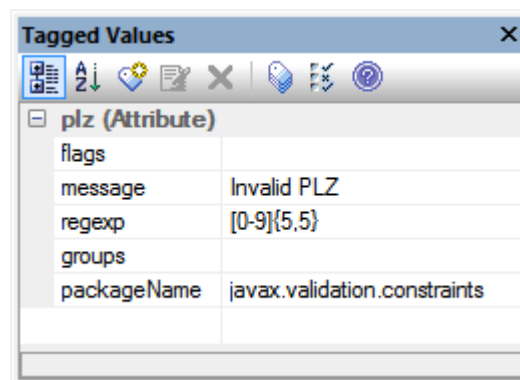


Abbildung 11: Screenshot der Tagged Values des Stereotyp "BVA_Pattern"

3.3 Anwendung Benutzerdefinierter Validatoren

Im Falle des Attributes `Buch::isbn` handelt es sich um eine 13 stellige Zahl, die zum EAN Code kompatibel ist. Die letzte Ziffer ist eine Prüfsumme die aus den zwölf vorherigen Ziffern berechnet werden kann. Da diese Validierung nicht mit den im JSR-303 definierten Standardvalidatoren realisiert werden kann, muss nun ein benutzerdefinierter Validator erstellt werden. Dies geschieht grundsätzlich in drei Schritte:

1. Anlegen des benutzerdefinierten Profils und des Stereotyps
2. Anlegen der benutzerdefinierten Java Annotation
3. Erstellen einer Implementierung des benutzerdefinierten Validators

3.3.1 Erstellen eines benutzerdefinierten BVA Profils

Da für den benutzerdefinierten Validator „Isbn“ noch kein Stereotyp existiert, muss dieser nun in einem eigenem Profil definiert werden.

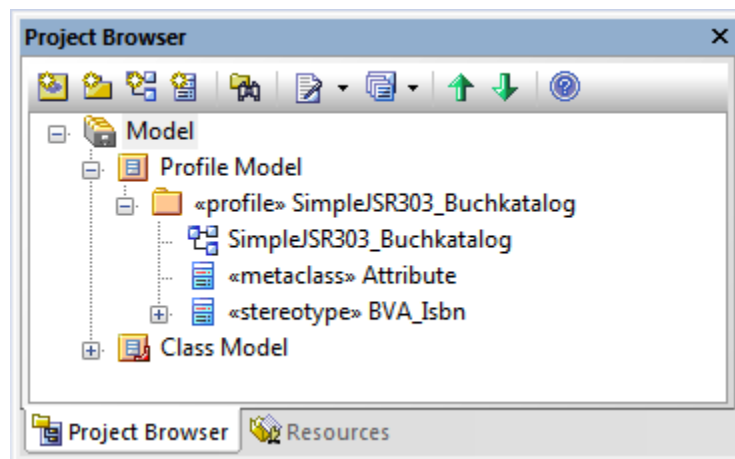


Abbildung 12: Beispiel Profilstruktur für benutzerdefinierte Validatoren

Für diesen Zweck soll eine Profilstruktur wie in Abbildung 12 dargestellt, angelegt werden. Dazu geht man wie folgt vor:

1. Rechtsklick auf *Model*
2. „New View...“
3. Im „Create New View“ Dialog im Feld *Name* „*Profile Model*“ eingeben und mit OK bestätigen.

4. Rechtsklick auf die soeben erstellte View und „Add → Add Package...“ auswählen
5. Im nun geöffneten „New Package Dialog“ *SimpleJSR303_Buchkatalog* als Package Namen eingeben (siehe Abbildung 13: Screenshot vom "New Model Package" Dialog im EA) und mit OK bestätigen.
6. Den Dialog „New Diagramm“ mit Okay bestätigen.

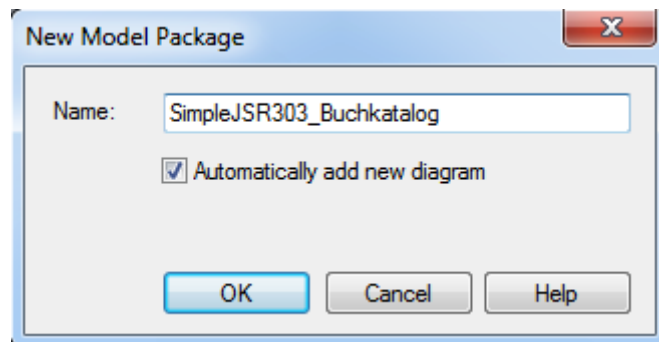


Abbildung 13: Screenshot vom "New Model Package" Dialog im EA

Die Projektstruktur ist jetzt bereits vollständig (siehe dazu Abbildung 14: Screenshot Projektstruktur ohne Stereotyp <<profile>>). Das Package *SimpleJSR303_Buchkatalog* muss nun lediglich mit dem Stereotyp <<profile>> ausgezeichnet werden, damit es als Profil einsatzbereit ist. Dazu klickt man doppelt auf das Package *SimpleJSR303_Buchkatalog* und wählt im nun geöffneten Dialog im Feld *Stereotype* den Eintrag *profile* aus (siehe dazu Abbildung 15: Screenshot vom Package Dialog im EA).

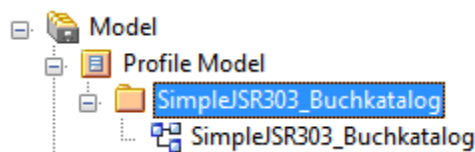


Abbildung 14: Screenshot Projektstruktur ohne Stereotyp <<profile>>

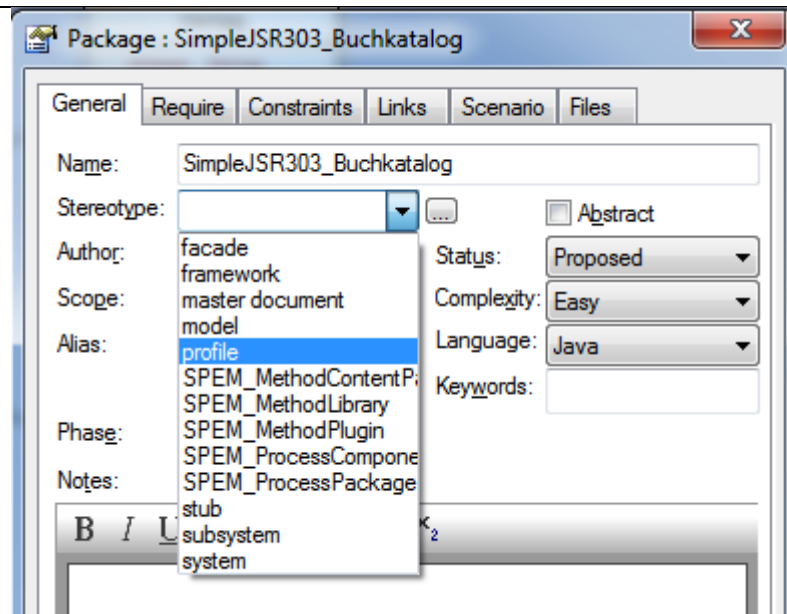



Abbildung 15: Screenshot vom Package Dialog im EA

3.3.2 Erstellen eines benutzerdefinierten BVA Stereotypen

Um nun einen benutzerdefinierten BVA Stereotypen anzulegen öffnet man das SimpleJSR303_Buchkatalog Diagramm, das automatisch unterhalb des neuen Packages *SimpleJSR303_Buchkatalog* angelegt wurde mit einem Doppelklick auf  SimpleJSR303_Buchkatalog .

Im *Enterprise Architect* sollte auf der linken Seite automatisch die *Toolbox* mit den Profil Elementen geöffnet worden sein (siehe dazu Abbildung 16: Screenshot der Profil Toolbox im EA).

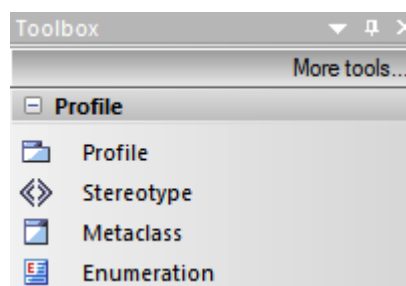


Abbildung 16: Screenshot der Profil Toolbox im EA

Der neue Stereotyp soll nur auf Attribute angewendet werden können, daher muss er von der Metaklasse *Attribute* abgeleitet werden. Dazu zieht man das Toolboxelement *Metaclass* per Drag'n'Drop auf die Diagrammfläche. Beim freigeben der Maustaste wird automatisch der *Create New Metaclass* Dialog geöffnet.

net. In diesem Dialog wählt man als Element *Attribute* aus und bestätigt diesen mit einem Klick auf OK (siehe dazu Abbildung 17: Screenshot des Create New Metaclass Dialogs im EA). Als nächstes zieht man ebenfalls per Drag'n'Drop das Toolboxelement *Stereotype* auf die Diagrammfläche. Auch hier öffnet sich automatisch ein Dialog, in welchem man den gewünschten Namen, nämlich *BVA_Isbn*, eingibt. Der Präfix *BVA_* ist hierbei zwingend erforderlich. Zusätzlich müssen dem Stereotyp nun noch einige Attribute hinzugefügt werden, die später im Modell als Tagged Values zur Verfügung stehen sollen.

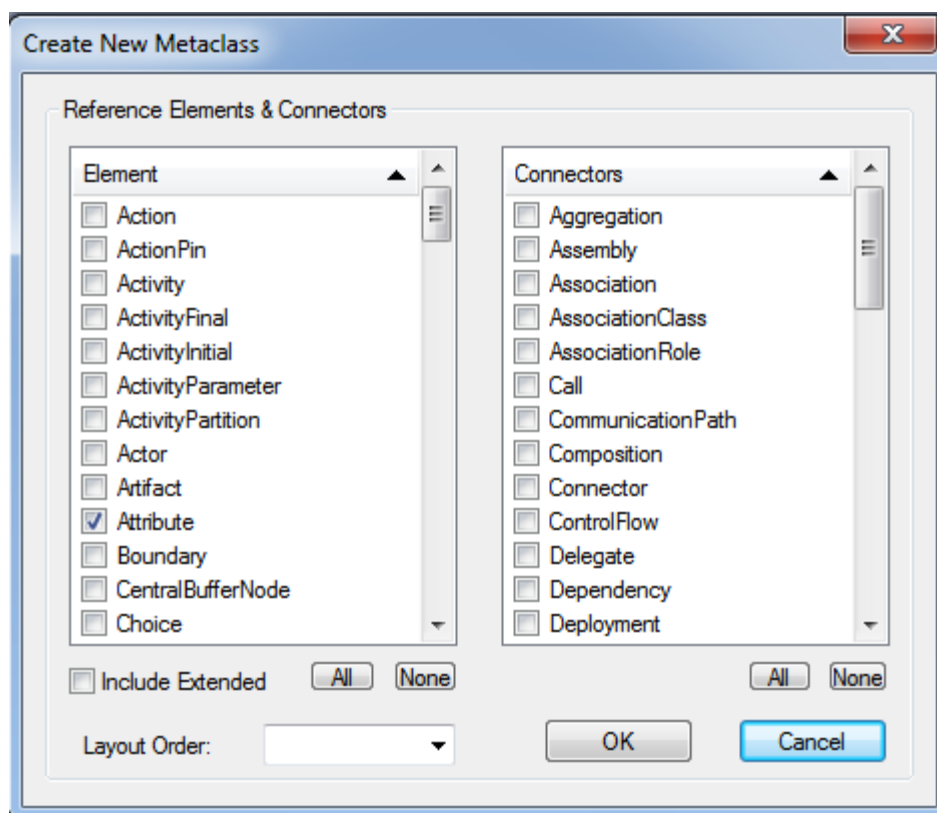


Abbildung 17: Screenshot des Create New Metaclass Dialogs im EA

Dazu klickt man auf den Registerreiter *Details* und dort auf den Button *Attributes*. Im nun geöffneten Dialog „*BVA_Isbn Attributes*“ werden folgende Attribute angelegt (siehe dazu

Attribut Name	Initial Value
message (String)	Invalid ISBN
packageName (String)	com.tsystems.bva.buchkatalog

Tabelle 2: Attribute des BVA_Isbn Stereotyps und Abbildung 18: Screenshot der BVA_Isbn Attribute).

Attribut Name	Initial Value
message (String)	Invalid ISBN
packageName (String)	com.tsystems.bva.buchkatalog

Tabelle 2: Attribute des BVA_Isbn Stereotyps

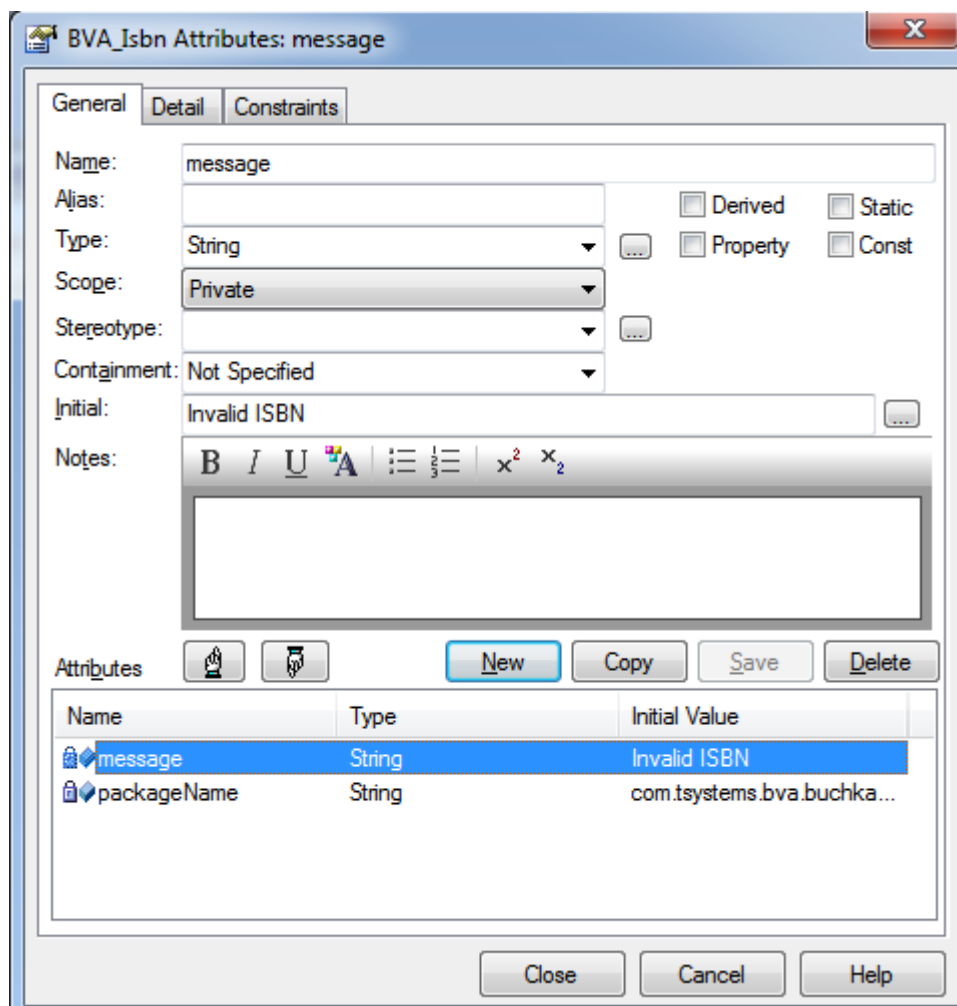


Abbildung 18: Screenshot der BVA_Isbn Attribute

Eine ausführliche Beschreibung zum Anlegen von Attributen findet man in Kapitel 4.3. Mit einem Klick auf *Close* wird dieser Vorgang abgeschlossen.

Nun muss der Stereotyp nur noch von der Metaklasse ableiten und das Profil ist fertig. Dazu klickt man einmal mit der linken Maustaste auf den Stereotypen, sodass die sogenannte „Quick Link“ Funktion, wie in Abbildung 19 dargestellt, eingeblendet wird.

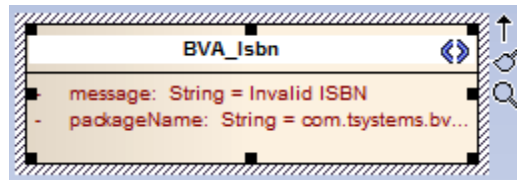


Abbildung 19: Stereotyp mit eingeblendetem Quick Link Symbol

Nun kann man den kleinen schwarzen Pfeil anklicken und per Drag'n'Drop auf die Metaklasse ziehen. Beim Loslassen erscheint ein Popupmenü in welchem man auf *Extend* klickt (siehe dazu Abbildung 20: Ableiten von einer Metaklasse). Fertig ist das benutzerdefinierte Profil.

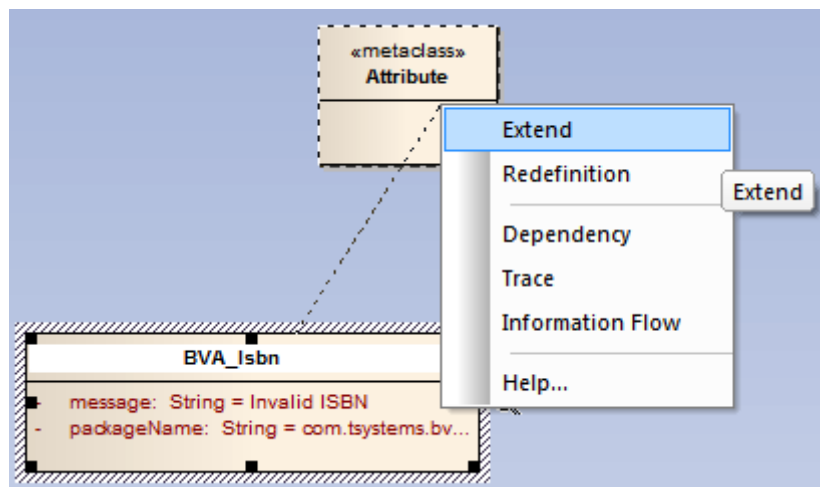


Abbildung 20: Ableiten von einer Metaklasse

Um das eben erstellte Profil im Beispiel Modell verwenden zu können muss dieses zuerst exportiert werden, damit es später über die „Ressourcen View“ wieder importiert werden kann.

Mit einem Rechtsklick auf eine freie Stelle im Profildigramm öffnet man das Kontextmenü und wählt dort den Punkt „Save as Profile...“ um den Exportdialog zu öffnen. Im nun angezeigten Dialog wählt man eine passende Stelle zum Speichern aus (siehe dazu Abbildung 21: Screenshot vom Profil Export Dialog).

Das exportierte Profil ist bereits Workspace unter „bva.exporter/Profile/-SimpleJSR303_Buchkatalog.xml“ zu finden.

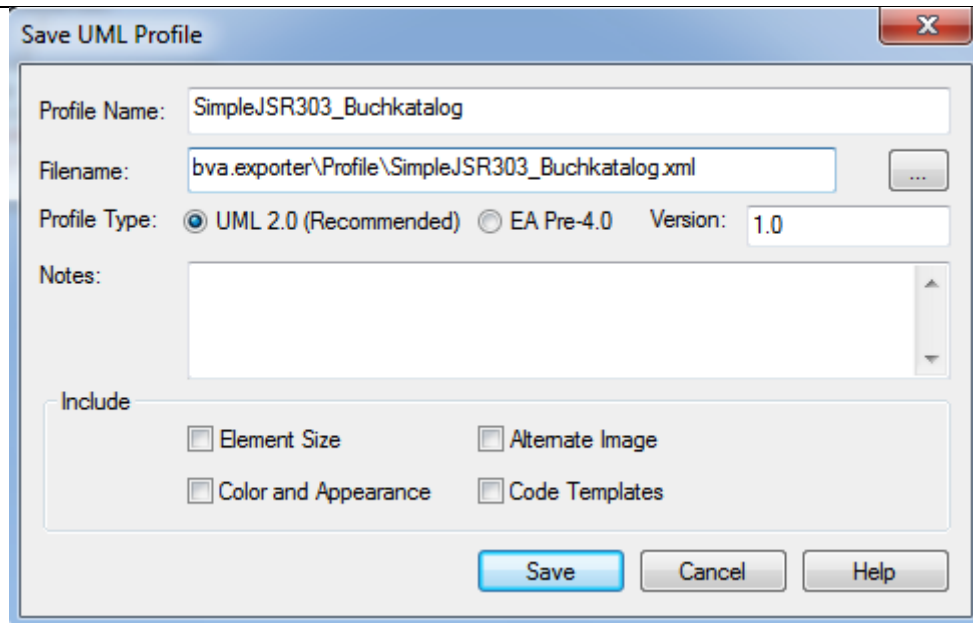


Abbildung 21: Screenshot vom Profil Export Dialog

3.3.3 Verwenden des benutzerdefinierten BVA Profils

Das exportierte Profil muss nun noch in das Beispiel Modell „eingehängt“ werden. Eine Anleitung dazu ist unter Kapitel 4.6 zu finden. Jetzt kann der neue benutzerdefinierte Stereotyp auf das Attribut *Buch::isbn* wie im Kapitel 4.7 erklärt angewendet werden.

3.3.4 Implementieren des benutzerdefinierten Validators

Damit der benutzerdefinierte Validator auch tatsächlich funktioniert wenn später Quellcode generiert wird und nicht zu einem Fehler führt, muss dieser natürlich auch Java seitig implementiert sein.

Da die verwendeten BVA Stereotypen auf echte Java Annotationen „gemapped“ werden, muss für den benutzerdefinierten Stereotyp *BVA_Isbn* die entsprechende Java Annotation *@Isbn* implementiert werden. Die Validierungslogik wird allerdings nicht direkt in der Annotation implementiert sondern in einer separaten Java-Klasse namens *IsbnValidator*. Wenn die Implementierung des Validators bereits vorliegt, muss nur sichergestellt sein, dass das Package im *bva.generator* Projekt im Build Path korrekt konfiguriert ist.

```
package com.tsystems.bva.buchkatalog;
```

```
import static java.lang.annotation.ElementType.*;
import static java.lang.annotation.RetentionPolicy.*;
```

```
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import javax.validation.Constraint;

import com.tsystems.bva.buchkatalog.IsbnValidator;

@Target( { FIELD })
@Retention(RUNTIME)
@Constraint(validatedBy = IsbnValidator.class)
@Documented
public @interface Isbn {

    String message() default "{validator.isbn}";

    Class<?>[] groups() default {};

}
```

Abbildung 22: Quellcode der Java Annotation @Isbn

Benutzerdefinierte Java Annotationen, wie in Abbildung 22 dargestellt, sind seit Java 1.5 möglich. Wichtig sind die Stichwörter *@Target*, *@Retention* und *@Constraint*.

@Target definiert eine Liste der Java Elemente die annotiert werden dürfen. Möglich sind u.a. *Type*³, *Field*, *Method* und *Annotation_Type*⁴.

@Retention muss auf *RUNTIME* eingestellt sein, da die Annotationen eben auch noch zur Laufzeit vorhanden sein müssen.

@Constraint(validatedBy = IsbnValidator.class) gibt die Klasse an die zur Validierung der Daten genutzt werden soll.

Die Methoden *message()* und *groups()* müssen bei der BVA deklariert sein. Weitere eigene Methoden sind möglich und sollten parallel zu den definierten Tagged Values erstellt werden.

Nun muss noch der Validator wie in Abbildung 23: Quellcodeausschnitt der Klasse *IsbnValidator* gezeigt, implementiert werden. Der Validator implementiert das *ConstrainValidator* Interface, welches zwei Typ Parameter spezifiziert. Der erste Parameter spezifiziert den Annotation Typ und der zweite den Daten-

³ Class, interface, enum etc.

⁴ Der *Annotation_Type* wird benötigt, wenn man eine Annotation mit einer weiteren Annotation auszeichnen möchte. Ermöglicht sozusagen verschachtelte Annotationen.

typ der Elemente, die der Validator verarbeiten kann. Mit der *inititalize()* Methode kann auf alle Attribute zugegriffen werden, die die Annotation definiert (Bei *Pattern* wäre das bspw. *regex* und *flag*). Die eigentlichen Validierungsregeln werden in der Methode *isValid(...)* implementiert. Die im droMDAry Package vorhandene Implementierung beinhaltet zusätzlich eine einfache Prüfsummenberechnung.

```
package com.tsystems.bva.buchkatalog;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

import com.tsystems.bva.buchkatalog.Isbn;

public class IsbnValidator implements ConstraintValidator<Isbn,
String> {

    public void initialize(Isbn constraintAnnotation) {
        // nothing to do
    }

    public boolean isValid(String isbn,
        ConstraintValidatorContext constraintContext) {
        // Here the ISBN-13 validation rules must be implemented
        // according to Wikipedia http://de.wikipedia.org/wiki/ISBN
        if (isbn == null)
            return false;

        if (isbn.length() != 13)
            return false;

        // Calculate checksum
        // ...
    }
}
```

Abbildung 23: Quellcodeausschnitt der Klasse IsbnValidator

Beide Klassen werden in dem Package erstellt das zuvor in der

Attribut Name	Initial Value
message (String)	Invalid ISBN
packageName (String)	com.tsystems.bva.buchkatalog

Tabelle 2: Attribute des BVA_Isbn Stereotyps, unter *packageName* festgelegt wurde.

3.4 Durchführen des Exports mit dem `bva.exporter`

In diesem Beispiel nutzen wir nicht den `common.exporter` sondern explizit den `bva.exporter`. Die hier beschriebenen Einstellungen können aber auch auf dieselbe Weise im `common.exporter` vorgenommen werden.

Bevor der Exporter gestartet werden kann muss erst noch sichergestellt werden, dass das zuvor in *Enterprise Architect* erstellte Modell sowie das benutzerdefinierte Profil, korrekt in den Prozess eingebunden sind.

3.4.1 Anpassen der `exporter.properties`

Dazu müssen in der Datei `bva.exporter/oaw/exporter.properties` noch einige Dinge beachtet werden.

Im ersten Schritt muss das erstellte Modell zur Verwendung konfiguriert werden. Dazu muss nun im Abschnitt „*Model definition*“ das erstellte Modell referenziert werden. Mit der Property `model.ea_file` gibt man den relativen oder auch absoluten Pfad zur *Enterprise Architect* Datei an. Die `model_file` Angabe gibt an, an welcher Stelle das XML des Modells abgelegt werden soll (die Endung `*.uml` ist beabsichtigt). Die Property `model_pkg` gibt den voll qualifizierten Namen des Packages an, unterhalb dessen sich das Modell befindet (Angaben wie in Kapitel 3.1). Weiterhin muss noch der Diagramm Name mit der Property `model_name` Angegeben werden. Zu guter Letzt gibt es noch die Property `output_slot` die nicht verändert werden darf, da sie für die interne Weiterverarbeitung notwendig ist.

```
# Model definition
model.ea_file = Model/BVA_Tutorial.EAP
model_file    = Model/model.uml
model_pkg     = Model/Class Model/BVATutorial
model_name    = BVATutorial
output_slot   = model
```

Abbildung 24: Konfiguration des Modells mit der `exporter.properties` Datei

Die Angabe bei `custom_profile_pkg` bezeichnet den vollqualifizierten Namen des Packages wie es im *Enterprise Architect* angelegt wurde und bei `custom_profile_name` wird dann entsprechend nur der tatsächliche Profilname aufgeführt.

Besonders wichtig ist die Angabe der *custom_profile_file* Property, da diese den Speicherplatz für das exportierte EMF kompatible XMI angibt. Hierbei ist es wichtig dass diese Datei bereits existiert bevor mit dem nächsten Schritt fortgefahren wird.

Hinweis: Mit *custom_profile_file* ist nicht die mit dem Profile Exporter des *Enterprise Architects* manuell erzeugte Datei aus Kapitel 3.3.1 gemeint, sondern die Datei die das Ergebnis der Model-to-Model Transformation des *bva.exporters*, enthalten soll. Auch die Endung *.uml ist kein Tippfehler sondern so beabsichtigt

Die Property *custom_profile_ea_file* verweist auf die *Enterprise Architect* Datei, in welcher das Profil erstellt wurde. In diesem Fall ist es die *Model/BVA_Tutorial.EAP* Datei.

Hinweis: Es ist nicht zwingend notwendig, dass das Projekt Modell und das Profil in derselben *Enterprise Architect* Datei modelliert werden.

In Abbildung 25 ist der komplette Abschnitt, der das benutzerdefinierte Profil betrifft, abgebildet.

```
# Custom Profile definition
custom_profile_pkg      = Model/Profile Model/SimpleJSR303_Buchkatalog
custom_profile_name     = SimpleJSR303_Buchkatalog
custom_profile_file     = Profile/SimpleJSR303_Buchkatalog.profile.uml
custom_profile_ea_file  = Model/BVA_Tutorial.EAP
custom_profile_dir      = Profile
```

Abbildung 25: Konfiguration des benutzerdefinierten Profils mit der *exporter.properties* Datei

3.4.2 Starten des Exportvorgangs

Da nun alle Vorbereitungen getroffen wurden kann der Exportvorgang mit einem Rechtsklick auf die Datei *bva.exporter/oaw/profile_exporter.oaw* und anschließend mit einem Klick auf „Run As... → oAW Workflow“ gestartet werden. In der Eclipse Konsole kann der weitere Fortschritt beobachtet werden. Sobald der Workflow abgeschlossen ist erscheint folgende Meldung im Konsofenster.

```
75061 INFO WorkflowRunner - workflow completed in 74026ms!
```

Hinweise: Sollte sich eine Demo Version des *Enterprise Architects* im Einsatz befinden, öffnet sich im Hintergrund mehrmals eine Meldung, die per Klick auf *Continue Trial* bestätigt werden muss.



Die selbe Vorgehensweise muss nun auch noch auf die Datei *bva.exporter/oaw/model_exporter.oaw* angewendet werden.

Nun liegen alle benötigten Modelle im EMF kompatiblen XMI Format vor.

3.5 Durchführen der Code Generierung mit dem `bva.generator`

Um die Quellcodegenerierung zu starten klickt man mit einem Rechtsklick auf die Datei `bva.generator/src/main/resources/profile_exporter.oaw` und anschließend auf „Run As... → oAW Workflow“. Nun werden die abstrakten sowie die Interfaceklassen im Verzeichnis `bva.generator/src/generated/java` mit samt den Packages generiert. Die Implementierungsklassen werden hingegen einmalig unter `bva.generator/src/main/java` erstellt. Bei allen weiteren Generierungsvorgängen werden diese Dateien nicht überschrieben um manuelle Änderungen am Quellcode zu schützen. Wenn diese Klassen auch neugeneriert werden sollen, müssen diese manuell gelöscht werden.

Auch hier kann der Vorgang in der Eclipse Konsole beobachtet werden. Die erfolgreiche Generierung wird mit der Meldung

```
INFO - Written 9 files to outlet [default] (./src/generated/java)
INFO - workflow completed in 3335ms!
```

quittiert.

Um zu überprüfen ob die Generierung tatsächlich erfolgreich war, kann man bspw. in der generierten Klasse `AbstractBuch` nachschauen, ob der benutzerdefinierte Validator `BVA_Isbn` wie folgt generiert wurde:

```
@com.tsystems.bva.buchkatalog.Isbn(message = "Invalid ISBN")
private java.lang.String isbn = null;
```

Dasselbe gilt für die `BVA_Pattern` Validatoren in den Klassen `AbstractAutor` und `AbstractVerlag` die wie folgt aussehen müssen:

```
@javax.validation.constraints.Pattern(
    message = "Invalid PLZ", regexp = "[0-9]{5,5}")
private java.lang.String plz = null;
```

3.6 Anmerkung zum Beispielmodell

Das zweimalige definieren der identischen Validierungsregel für die *plz* Attribute ist im Beispiel etwas unschön gewählt.

Man stelle sich vor, dass zu einem späteren Zeitpunkt die PLZ Validierung angepasst werden soll, da eine vorher nicht bekannte Einschränkung berücksichtigt werden muss, z.B. dass Feldpost mit einer vierstelligen PLZ ausgezeichnet wird.

Um die Konsistenz der Validierungsregeln für Postleitzahlen bei späteren Änderungen sicherstellen zu können, sollte also besser ein benutzerdefinierter Validator namens *BVA_Plz* verwendet werden.

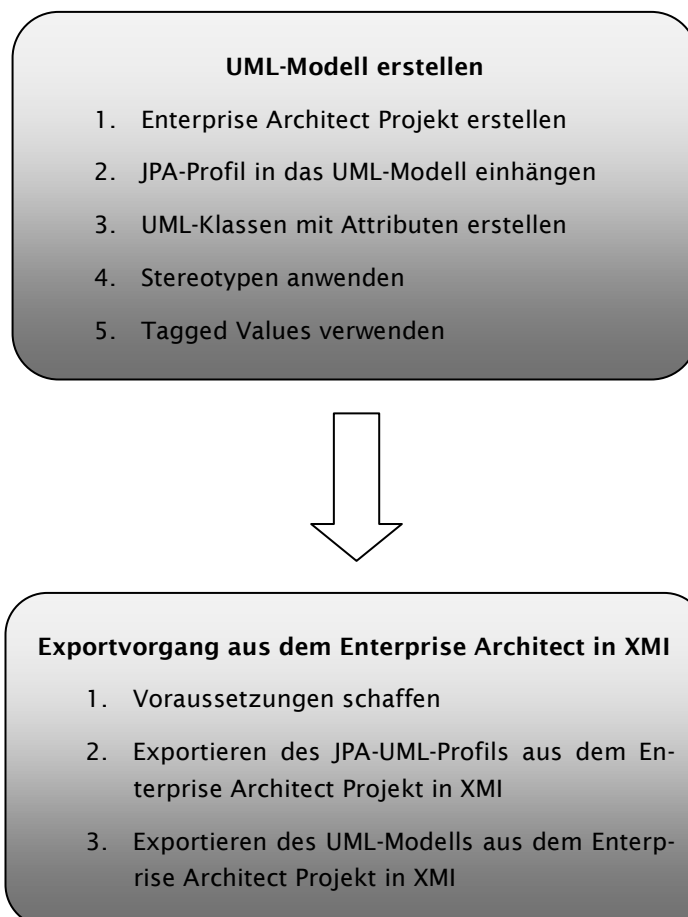
Dieses Vorgehen hätte mehrere Vorteile: Zum einen kann der Validator problemlos in späteren Projekten wiederverwendet werden und zum anderen ist bei einer Änderung sichergestellt, dass diese an allen Stellen an denen ein *BVA_Plz* Validator eingesetzt wird auch tatsächlich zum Tragen kommt.



4 Anwendung des JPA-Generators anhand eines Beispiels

Dieses Kapitel ist ein Tutorial und dient als Einarbeitung in den MDA-Generator. Anhand eines anschaulichen Beispiels lernt man schnell und einfach die erforderlichen Schritte für die Codegenerierung. Von der Erstellung des UML-Modells mit dem Werkzeug Enterprise Architect über vollständig generierten Javacode bis hin zur Generierung einer DDL-Datei.

4.1 Überblick



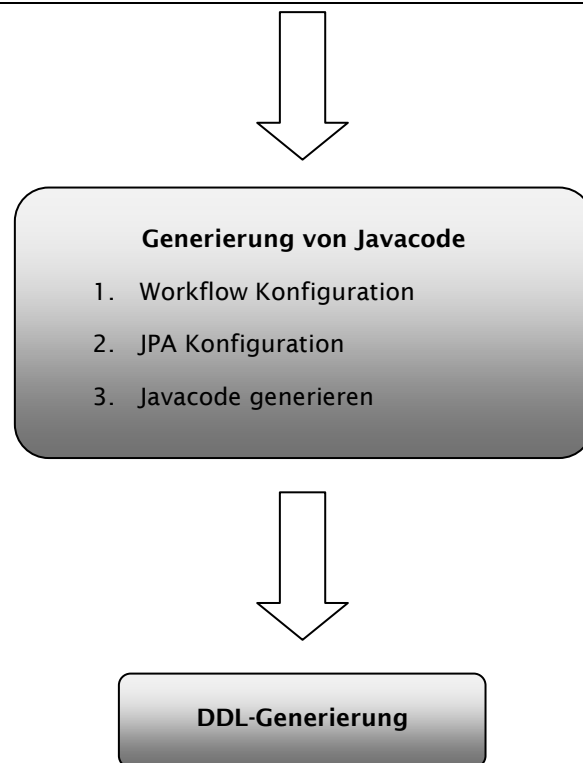


Abbildung 26: Überblick über das Tutorial

4.2 Erstellen des Beispiel Modells

1. Enterprise Architect Projekt erstellen

Damit ein UML-Modell erstellt werden kann, ist es notwendig, ein neues Enterprise Architect Projekt mit dem Namen Tutorial zu erstellen. Die dafür erforderlichen Schritte sind im Kapitel 4.1 beschrieben. Ablageverzeichnis der Projektdatei sollte das Verzeichnis */Ihr Workspace/jpa.exporter/Model/* sein.

Die Klassen Class1 bis Class3 sowie das Interface1 in der Resources View sollte man entfernen. Sie werden beim Anlegen des Enterprise Architect Projekts automatisch erzeugt. Das Package System sowie das Diagramm System bitte in JpaTutorial umbenennen.

2. JPA-Profil in das UML-Modell einhängen

Um das JPA-UML-Profil im Modell verwenden zu können, kann dies wie in Kapitel 4.6 eingehängt werden. Die dafür benötigte XML-Datei SimpleJSR220.xml ist im droMDAry Package im JPA Profil Ordner zu finden.

3. UML-Klassen mit Attributen erstellen

Nun kann das Modell, welches im Beispiel ein reines Entitäten-Modell ist, erstellt werden. Als erstes ist die Klasse „Autor“ mit den Attributen „vorname“, „nachname“, „strasse“ vom Typ String sowie „alter“ und „plz“ von Typ Integer zu erstellen. Die Schritte für die Erstellung und Bearbeitung von Klassen und Attributen sind im 4.2 und Kapitel 4.3 zu finden.

Weitere Klassen mit Attributen (alle haben den Scope „private“), die man erstellen muss:

- Klasse: Buch
 - Attribut: buchId vom Typ int
 - Attribut: auflage vom Typ int
 - Attribut: titel vom Typ String
 - Attribut: isbn vom Typ int
- Klasse: Verlag
 - Attribut: verlagName vom Typ String
 - Attribut: strasse vom Typ String
 - Attribut: plz vom Typ int
- Klasse: Taschenbuch
 - Attribut: format vom Typ String

Somit sind alle Klassen für das Beispielmmodell erstellt. Jede Klasse wird später 1:1 mit allen Attributen auf die Datenbank gemappt. Dies bedeutet, jede Klasse entspricht einer Tabelle in einer Datenbank.

4. Stereotypen anwenden

Nun kann man die entsprechenden Stereotypen aus dem JPA-UML-Profil auf die UML-Elemente anwenden. Da der MDA-Generator nach dem KISS-Prinzip (Keep it simple and stupid) arbeitet, sind nur wenige Angaben nötig, um eine lauffähige Java-Persistenzschicht zu generieren.

Der Stereotyp „JPA_Entity“ muss zwingend auf jede Klasse angewendet werden, damit diese Klasse später auf eine Datenbank gemappt werden kann. Natürlich sind auch Klassen, welche keine Entitäten sind, im Modell modellierbar und werden somit nicht auf die Datenbank abgebildet. In diesem Fall muss man den Stereotyp „JPA_Entity“ nicht auf eine Klasse anwenden.

Alle anderen Stereotypen sind optional. D.h. wenn man keine weiteren Stereotypen anwendet, generiert der MDA-Generator die für die Persistenzschicht notwendigen Annotationen⁵ und Id's implizit (wird als **Standardfall** bezeichnet). Dadurch wird der Modellierungsvorgang stark vereinfacht und dem Entwickler viel Arbeit abgenommen.

Im Standardfall wird der Javacode für Entity-Klassen und Attribute wie folgt generiert:

- Der Klassenname im Modell entspricht dem Tabellennamen in der Datenbank
- Das Datenbankschema entspricht dem im „jpa.properties“-File angegebenen Schema
- Jedes Attribut im Modell wird als Attribut für die Datenbank gemappt (Annotation „JPA_Column“)
- Der Attributname im Modell entspricht auch dem Namen in der Datenbank (Annotation „JPA_Column“ ohne Tagged Value)
- Für jede Klasse wird ein Attribut mit dem Namen „id“ vom Typ Integer inkl. der Annotation „@Id“ erzeugt (Primärschlüssel)

Falls man nicht den Standardfall verwenden möchte, sind optionale Stereotypen aus dem JPA-UML-Profil anzuwenden. Wenn noch nicht geschehen, muss der Stereotyp „JPA_Entity“ nun auf alle Klassen im Beispielmmodell angewendet werden. Wie dies mit dem Enterprise Architect funktioniert, kann man im Kapitel 4.7 nachlesen.

Im Beispielmmodell enthält nur die Klasse „Buch“ eine explizit definierte Id und somit den optionalen Stereotyp „JPA_Id“. Für die anderen Klassen wird implizit eine Id generiert, da diese für die eindeutige Identifikation eines Datensatzes zwingend notwendig ist. Hier ist es notwendig die Id explizit zu definieren, da diese Klasse im Beispiel einen fachlich zusammengesetzten Primärschlüssel enthalten soll. Um das im Beispielmmodell zu realisieren, muss auf die Attribute „buchId“ und „auflage“ der Stereotyp „JPA_Id“ angewendet werden. Sobald es mehr als ein angewendeten „JPA_Id“ Stereotyp in einer Klasse gibt, erkennt der MDA-Generator automatisch, dass eine Primärschlüsselklasse generiert werden muss. Die Primärschlüsselklasse ist zur korrekten Identifizierung eines Datensatzes, der nur durch mehrere Primärschlüssel identifiziert werden kann, da.

⁵ Von Stereotypen spricht man im UML-Modell und in Java von Annotationen.

Als Beispiel für eine weitere optionale Angabe wird der Stereotyp „JPA_Column“ auf das Attribut „strasse“ der Klasse „Autor“ angewendet. Damit ist explizit definiert, dass dieses Attribut auch als Attribut in der Datenbank verwendet wird. Wenn bei dieser Annotation nichts mehr geändert wird, wäre sie unnötig. Denn der Standardfall deckt diesen, wie oben erwähnt, mit ab. Damit diese Angabe nicht unnötig ist, ist dafür ein Tagged Value zu verwenden (siehe Punkt 5.). Sobald in einer Klasse auf ein Attribut die „JPA_Column“ Annotation angewendet wird, muss für alle anderen Attribute, welche in einer Datenbank auch als Attribute erscheinen sollen, die Annotation ebenfalls gesetzt werden. Ansonsten werden diese Attribute in einer Datenbank ignoriert.

5. Tagged Values verwenden

Der Datenbankname des Attributs „strasse“ der Klasse „Autor“ soll im Beispiel nicht dem Namen des Attributs im Modell entsprechen, sondern dem Namen „str“. Jetzt kommen die Tagged Values ins Spiel. Manche Stereotypen aus einem Profil haben Tagged Values. Zurück zum Name des Attributs „strasse“. Der Tagged Value „name“ muss nun in der Tagged Values View im Enterprise Architect auf „str“ gesetzt werden. Wie dieser Schritt funktioniert, ist im Kapitel 4.8 erklärt.

6. Beziehungen modellieren

Der letzte Schritt für die Erstellung des UML-Modells ist, die Beziehungen zwischen den einzelnen Klassen im Modell einzutragen. Wie das mit dem Enterprise Architect realisiert wird, kann im Kapitel 4.5 nachgelesen werden. Die Klassen stehen in folgenden Beziehungen zueinander:

- Autor:Verlag / *: * (ManyToMany) / Typ: Association
- Autor:Buch / 1: * (OneToMany)/ Typ: Aggregation
- Buch:Taschenbuch / Typ: Ableitung (Taschenbuch ist von Buch abgeleitet)

Für die Beziehung „Autor:Verlag“ ist es nicht notwendig, die Multiplicity anzugeben, da der MDA-Generator ohne eine Angabe der Multiplicity implizit eine „*: *“ (ManyToMany) Beziehung annimmt.

Das nun fertig modellierte UML-Modell sollte wie folgt aussehen:

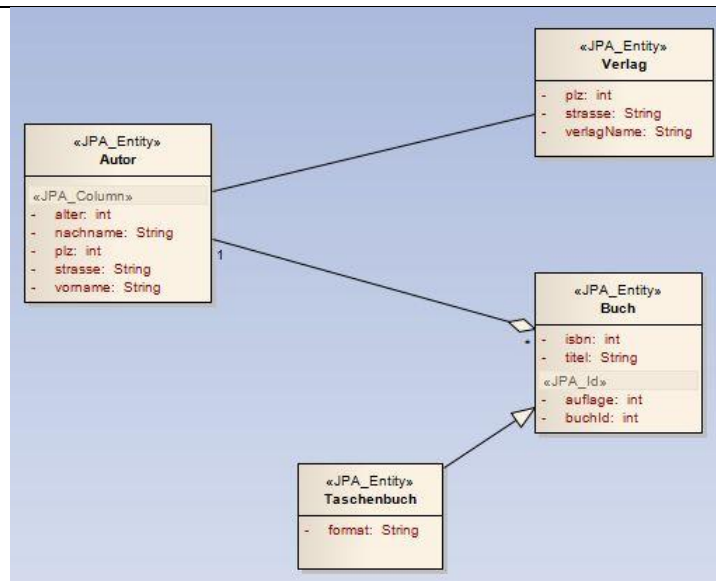


Abbildung 27: UML-Beispielmodell

4.2.1 Exportvorgang aus dem Enterprise Architect in XMI

Der Exportvorgang findet in der oAW-Umgebung im Projekt `jpa.exporter` statt. Um eine optimale Unterstützung in der oAW-Umgebung (Eclipse) zu haben, wechselt man am besten in die `openArchitectureWare` Perspektive. In diese gelangt man über das Menü *Window – Open Perspective – Other... – openArchitectureWare*. Desweiteren ist es zu empfehlen, über die Navigator Ansicht im Workspace zu navigieren. Diese kann über das Menü *Window – Show View – Navigator* geöffnet werden.

1. Voraussetzungen schaffen

Im Auslieferungszustand ist die Datei „SimpleJSR220-profile.eap“ im Verzeichnis „Model“ vorhanden. Falls diese Datei nicht in diesem Verzeichnis liegt, ist es notwendig, sie aus dem Profil Ordner des droMDAry Packages, in das Verzeichnis `/Ihr Workspace/jpa-exporter/Model` zu kopieren. Diese Datei enthält das komplette JPA-UML-Profil und wird nachher in das XMI-Format transformiert. Man wundert sich vielleicht, warum das Profil nochmals in einer separaten Enterprise Architect Projektdatei enthalten ist, da es schon im UML-Modell eingehängt ist. Da aber das eingehängte Profil im UML-Modell nur als UML-Erweiterung dient sowie bei der Transformation in XMI nicht als separates Profil transformiert und somit aus oAW nicht wiederverwendet werden kann, muss es in einer einzelnen Datei zur Verfügung stehen. Desweiteren soll es möglich sein, mehrere Profile zu verwenden. Um dafür eine saubere und übersichtliche

Strategie einzubringen, ist für jedes UML-Profil eine separate Enterprise Architect Projektdatei notwendig.

2. Exportieren des JPA-UML-Profiles aus dem Enterprise Architect Projekt in XMI

oAW bietet so genannte Workflow-Dateien, welche den Ablauf einer Generierung oder einer Transformation bestimmt. Um den Generierungsprozess zu starten, muss nur die entsprechende Workflow-Datei ausgeführt werden. Die entsprechende Workflow-Datei für den Export- / Transformationsvorgang ist im Verzeichnis */Ihr Workspace/jpa.exporter/oaw/profile.oaw* zu finden.

Jedes oAW-Projekt enthält desweiteren eine Properties-Datei, die eine Konfiguration des Generierungsprozesses beinhaltet. Im „jpa.exporter“ Projekt dient die Datei */Ihr Workspace/jpa.exporter/oaw/workflow.properties* dazu. Für den Exportvorgang des UML-JPA-Profiles, sind folgende Einstellungen notwendig:

1

```
##### Konfiguration des UML-Profiles #####

# Datei des Enterprise Architect Projekts.
profile_ea_file = Model/SimpleJSR220-Profile.eap

# Datei, in welche das in XMI transformierte UML-Profil abgelegt wird.
# ACHTUNG: Diese Datei muss immer existieren (es darf sich auch um eine leere Datei handeln).
profile_file    = Profile/SimpleJSR220.profile.uml

# Pfad zum Package, in dem das Diagramm abgelegt ist.
# Der Pfad ist im Enterprise Architect im Project Browser zu erkennen.
profile_pkg     = Models/Profile Model/SimpleJSR220

# Name des Enterprise Architect Diagramms, welches sich in profile_pkg befindet.
profile_name    = SimpleJSR220

# Verzeichnis, in dem sich das Profil befindet.
profile_dir     = Profile

# Source Ordner für generierte Artefakte
gen_dir        = src-gen
```

Abbildung 28: workflow.properties – Konfiguration des Profil Exportvorgangs

Desweiteren sollte die Property „project_root“ (Basisverzeichnis) leer sein.

Diese Beispielkonfiguration sowie auch die Konfiguration für das Modell ist bereits in der Datei „workflow.properties“ im Auslieferungszustand enthalten. Weitere Informationen dazu findet man im Kapitel.

Die Datei „SimpleJSR220-Profile.eap“ ist im Auslieferungszustand des Projekts „jpa.exporter“ vorhanden. Durch diesen Exportvorgang wird die Datei „SimpleJSR220.profile.uml“, die das transformierte XMI-Format hat, generiert.

Per *Rechtsklick auf die Datei profile.oaw – Run As – oAW Workflow* startet man den Export- / Transformationsvorgang. Falls die Trial Version auf dem PC installiert ist, erscheint ein Dialog mit den Namen „Evaluation Version of Enterprise Architect“. Dieser Dialog kann mit dem Button *Continue Trial* bestätigt werden. Bei diesem Vorgang wird auf die Datei „SimpleJSR220-Profile.eap“ zugegriffen um es zu transformieren. Voraussetzung ist dadurch ein lauffähiger Enterprise Architect mit einer gültigen Lizenz. In der Konsolen View des Eclipse wurden während des Vorgangs Informationen ausgegeben. Ein erfolgreicher Exportvorgang wird mit folgender Ausgabe bestätigt:

```
... INFO WorkflowRunner      - workflow completed in ... ms!
```

3. Exportieren des UML-Modells aus dem Enterprise Architect Projekt in XMI

Für den Exportvorgang des UML-Modells, sind folgende Einstellungen in der Datei „workflow.properties“ notwendig:

```
# Basisverzeichnis
project_root =

##### Konfiguration des UML-Modells #####

# Datei des Enterprise Architect Projekts.
model_ea_file = Model/Tutorial.eap

# Datei, in welche das in XMI transformierte UML-Modell abgelegt wird.
# ACHTUNG: Diese Datei muss immer existieren (es darf sich auch um eine leere Datei handeln).
model_file = Model/model.uml

# Pfad zum Package, in dem das Diagramm abgelegt ist.
# Der Pfad ist im Enterprise Architect im Project Browser zu erkennen.
model_pkg = Model/Class Model/JpaTutorial

# Name des Enterprise Architect Diagramms, welches sich in model_pkg befindet.
model_name = JpaTutorial

# Interner Slot für oAW, in welchem sich das Modell im EMF-Format befindet.
output_slot = model
```

Abbildung 29: workflow.properties – Konfiguration des Modell Exportvorgangs

Das Beispielmmodell wurde in Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** erstellt und sollte sich bereits im Verzeichnis befinden. Diese Datei wird in XMI transformiert und in der Datei „model.uml“ abgelegt.

Per *Rechtsklick auf die Datei model.oaw – Run As – oAW Workflow* startet man wieder den Export- / Transformationsvorgang. Anschließend bitte wieder den Enterprise Architect Dialog mit dem Button *Continue Trial* bestätigen. Der Exportvorgang war erfolgreich, wenn in der Eclipse Konsolen View folgende Ausgabe erscheint:

... INFO WorkflowRunner - workflow completed in ... ms!

4.2.2 Generierung von Javacode

Ein Generierungsprozess wird im Projekt „jpa.generator“ durchgeführt. Wie auch im „jpa.exporter“ Projekt gibt es Properties Dateien. Diese sowie auch die Workflow Datei, mit der später der Generierungsprozess gestartet wird, befinden sich im Verzeichnis */Ihr Workspace/jpa.generator/src/main/resources*. Im Auslieferungszustand sind die Properties Dateien schon entsprechend konfiguriert. Um schnell einen Einblick zu bekommen, sind die wichtigsten Einstellungsmöglichkeiten trotz der fertigen Konfiguration in diesem Abschnitt erklärt. Wie auch im Exportvorgang sind alle Einstellungen in Properties Dateien zu pflegen.

Im Generator gibt es zwei Properties Dateien:

- workflow.properties -> Konfiguration des Workflows
- jpa.properties -> JPA-Einstellungen

1. Workflow Konfiguration

In der „workflow.properties“ Datei gibt es zwei Abschnitte. Der erste Abschnitt betrifft die Konfiguration des JPA-Generators wie z.B. die Einstellung der Ausgabepfade und des UML-Modells sowie des Profils, welche im Exportvorgang erstellt wurden und als Input für den JPA-Generator dienen (siehe Abbildung 30: Konfiguration der JPA-Cartridge in der Datei „workflow.properties“).


```
##### Konfiguration der JPA-Cartridge #####
# Basisverzeichnis
basedir = .

# Encoding Einstellung für das Einlesen von Daten.
file.encoding = ISO-8859-1

# Encoding Einstellung für das Ausgeben von Daten.
file.encoding.output = ISO-8859-1

# Ordner für Artefakte, die generiert und nicht modifiziert oder gelöscht werden dürfen.
outlet.src.dir = ${basedir}/src/generated/java

# Ordner des Artefakts persistence.xml.
outlet.persistenceXML = ${basedir}/META-INF/persistence.xml

# Ordner für Interface Artefakte.
outlet.src.interfaces.dir = ${basedir}/src/generated/java

# Ordner für Konfigurationsdateien.
outlet.res.dir = ${basedir}/src/generated/resources
outlet.res.once.dir = ${basedir}/src/main/resources

# Ordner für Artefakte, die später durch die Entwickler angepasst werden
# können (Implementierungsklassen).
outlet.src.once.dir = ${basedir}/src/main/java

# Datei des UML-Modells
model.file = ${basedir}/../jpa.exporter/Model/model.uml
```

Abbildung 30: Konfiguration der JPA-Cartridge in der Datei „workflow.properties“

Der zweite Abschnitt dient zur Konfiguration der mitbenutzen Javabasic-Cartridge. Mehr Informationen zur Konfiguration der Javabasic Cartridge sind unter

[http://fornax.itemis.de/confluence/display/fornax/JavaBasic+\(CJB\)](http://fornax.itemis.de/confluence/display/fornax/JavaBasic+(CJB))
zu finden.

```
##### Konfiguration der mitverwendeten Javabasic-Cartridge #####

# Mehr Informationen zu diesen Properties findet man unter
# http://fornax.itemis.de/confluence/display/fornax/JavaBasic+(CJB)

# Inhalt des Headers für generierte Artefakte
type.header.text = '/* (c) 2009 - Matthias Ziegler \n\
 * E-Mail: matthias.ziegler@t-systems.com\n\
 * \n\
 * Diese Datei wurde generiert und darf weder modifiziert, noch gelöscht werden!\n\
 * Änderungen bitte stets im Modell oder Profil umsetzen!\n\
 */\n'

# Inhalt des Footers für generierte Artefakte
type.footer.text = ''

# Diese Property wird von der Javabasic Cartridge benötigt.
annotation.source.key = ''

# Generierung der Methoden equals, hashCode und toString in
# abstrakten Entity-Klassen.
# 'true' - Methoden werden generiert.
# 'false' - Methoden werden nicht generiert.
use.overridden.equals.hashCode.toString = 'true'

# Java Version (Generic Support 5 oder höher)
java.version = 5

# Generierung von abstrakten Methoden in einer abstrakten Klasse.
# 'use_abstract_method' - Abstrakte Methode in abstrakter Klasse generieren.
# 'none' - Keine abstrakte Methode generieren.
interface.operation.implementation.strategy = 'none'

# Generierung der serialVersionUID.
# 'true' - serialVersionUID wird generiert.
# 'false' - serialVersionUID wird nicht generiert.
javabasic.generateSerialVersionUID = 'true'

# Entscheidung ob eine Liste als 'set' oder als 'list' generiert wird.
list.set.property = 'unique'
```

Abbildung 31: Konfiguration der Javabasic-Cartridge in der Datei „workflow.properties“

2. JPA Konfiguration

Die Properties Datei „jpa.properties“ ist zur JPA Konfiguration da. Sie definiert den Pfad der „persistence.xml“, weitere Einstellungen zum Mapping der Objekte, das Datenbankschema der Tabellen sowie den Packagename der zu generierenden DAO-Files (siehe Abbildung 32: JPA Konfiguration in der Datei „jpa.properties“). Mehr technische Details dazu sind im Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** aufgeführt.

```

##### JPA Konfiguration #####

# Pfad der persistence.xml ausgehend von src/generated/java.
persistence.xml.path = ../../main/java/META-INF/persistence.xml

# Properties für persistence.xml.
persistence.xml.properties = <!-- Scan for annotated classes and Hibernate mapping XML files -->\n\
    <property name=\"hibernate.archive.autodetection\" value=\"class, hbm\"/>\n\
    <!-- SQL stdout logging\n\
    <property name=\"hibernate.show_sql\" value=\"true\"/>\n\
    <property name=\"hibernate.format_sql\" value=\"true\"/>\n\
    <property name=\"use_sql_comments\" value=\"true\"/>\n\
    -->\n\
    <property name=\"hibernate.connection.driver_class\" \n\
        value=\"org.apache.derby.jdbc.EmbeddedDriver\"/>\n\
    <property name=\"hibernate.connection.url\" \n\
        value=\"jdbc:derby:miadb;create=true\"/> \n\
    <property name=\"hibernate.connection.username\" \n\
        value=\"sa\"/>\n\
    <property name=\"hibernate.c3p0.min_size\" \n\
        value=\"5\"/>\n\
    <property name=\"hibernate.c3p0.max_size\" \n\
        value=\"20\"/>\n\
    <property name=\"hibernate.c3p0.timeout\" \n\
        value=\"300\"/>\n\
    <property name=\"hibernate.c3p0.max_statements\" \n\
        value=\"50\"/>\n\
    <property name=\"hibernate.c3p0.idle_test_period\" \n\
        value=\"3000\"/>\n\
    <property name=\"hibernate.dialect\" \n\
        value=\"org.hibernate.dialect.HSQLDialect\"/>\n\
    <property name=\"hibernate.hbm2ddl.auto\" value=\"create-drop\"/>

# Datenbankschema
table.schema = APP

# Paketname für DAO relevante Java Klassen.
dao.package.name = dao

```

Abbildung 32: JPA Konfiguration in der Datei „jpa-properties“

Wenn beide Properties Dateien fertig konfiguriert sind und das Enterprise Architect Modell sowie das Profil in XML transformiert wurde, kann der Generierungsprozess gestartet werden. Das Modell und Profil muss man nicht in das Generator Projekt verschieben, es wird direkt aus dem Exporter Projekt gelesen.

3. Javacode generieren

Die Datei „workflow.oaw“ per *Rechtsklick – Run As – oAW Workflow* ausführen. In der Eclipse Konsolen View kann nun kontrolliert werden, ob der Workflow korrekt ausgeführt wurde. Falls dies der Fall ist, wurde für jede Entität ein Interface, eine abstrakte Klasse sowie eine Implementierungsklasse für eigene Methoden generiert. Hierbei gilt es zwischen den generierten Dateien, welche durch einen Entwickler modifiziert werden dürfen und welche nicht, zu unterscheiden. Dateien die nicht modifiziert werden dürfen, befinden sich im Verzeichnis *src/generated/java/jpatutorial*. Darin sind alle Interfaces und abstrakte Klassen abgelegt. Wenn in diesen Dateien eine Änderung nötig ist, muss

dies im UML-Modell geändert werden. Anschließend ist es notwendig, den kompletten Generierungsprozess (inkl. Exportvorgang) wieder durchzuführen. Im Verzeichnis *src/main/java/jpatutorial* befinden sich alle Implementierungsklassen. Diese werden nur einmalig generiert. Falls ein weiterer Generierungsprozess durchgeführt wird, können die Implementierungsklassen nicht überschrieben werden. So gehen keine Anpassungen eines Entwicklers in der Implementierungsklasse verloren. Falls es dennoch gewünscht ist, die Implementierungsklassen neu zu generieren, muss das Package „JpaTutorial“ einfach gelöscht werden. Anschließend erzeugt der Generator diese Dateien neu.

Um zu verstehen, wie der Generator das UML-Modell in Javacode umsetzt, wird anhand der abstrakten Klassen „AbstractAutor“ und „AbstractBuch“ erklärt (siehe Abbildung 33: Generierte Klasse „AbstractAutor“).

```
@Entity
@Table(name = "Autor", schema = "APP")
public abstract class AbstractAutor
    implements Autor {

    @Id
    private int id;

    @Column()
    private java.lang.String nachname = null;

    @Column()
    private java.lang.Integer alter = null;

    @Column()
    private java.lang.Integer plz = null;

    @Column(name = "str")
    private java.lang.String strasse = null;

    @Column()
    private java.lang.String vorname = null;

    @OneToMany(targetEntity = JpaTutorial.AbstractBuch.class)
    @JoinColumn(name = "Autor_id", nullable = false)
    public java.util.Set<JpaTutorial.Buch> buch = null;

    @ManyToMany(targetEntity = JpaTutorial.AbstractVerlag.class)
    @JoinColumn(name = "Autor_id", nullable = false)
    public java.util.Set<JpaTutorial.Verlag> verlag = null;
```

Abbildung 33: Generierte Klasse „AbstractAutor“

Um alle JPA-Annotationen zu unterstützen, ist der Import von „*javax.persistence.**“ generiert worden.

Im UML-Modell ist die Klasse Autor mit dem Stereotyp „JPA_Entity“ versehen worden. Dadurch generiert der Generator die Annotation „@Entity“ und damit diese Klasse später auch richtig auf eine Datenbank gemappt werden kann,

erzeugt er die Annotation „@Table“ mit den Tagged Values „name“, welcher den Tabellennamen beschreibt sowie „schema“, welcher das Schema aus der „jpa.properties“ Datei ausliefert.

Für die Klasse Autor ist im UML-Modell keine Id definiert worden. Aus diesem Grund generiert der JPA-Generator automatisch ein Attribut vom Typ Integer und zeichnet diese mit der Annotation „@Id“ aus.

Die Attribute, welche im Modell mit dem Stereotyp „JPA_Column“ ausgezeichnet sind, wurden im Java-Code mit der Annotation „@Column“ versehen. Damit sind die Attribute später auch in der Datenbank verfügbar. Speziell für das Attribut „strasse“ ist im UML-Modell der Tagged Value „name“ auf „str“ gesetzt worden, um den Namen des Attribut speziell in der Datenbank auf „str“ zu setzen. Aus diesem Grund ist für das Attribut „strasse“ im Java-Code auch die „@Column“ Annotation mit dem Tagged Value „name“ generiert worden.

Die Attribute „buch“ und „verlag“ sind dazu da, um die modellierten Beziehungen in Java abzubilden. Die Beziehung „Autor zu Buch“ wurde als „1:*“ Beziehung modelliert. Deswegen wird durch den Generator die Annotation „@OneToMany“ mit dem nötigen Tagged Value „targetEntity“, welche der Zielklasse in der Beziehung entspricht, versehen. Die Annotation „@JoinColumn“ wird für ein korrektes Mapping der Beziehung in die Datenbank generiert. Gleiches Schema gilt für die Beziehung „Autor zu Verlag“, aber eben nur für eine „*:*” (ManyToMany) Beziehung.

Anschließend wurden die Getter- und Setter sowie die „Equals“ und „Hash Code“ zum Vergleichen der Objekte und die „To String“ Methode generiert.

```
@IdClass(BuchPk.class)
@Entity
@Table(name = "Buch", schema = "APP")
public abstract class AbstractBuch
implements Buch {

    @Id
    @Column
    private java.lang.Integer buchId = null;

    @Id
    @Column
    private java.lang.Integer auflage = null;

    @Column
    private java.lang.String titel = null;

    @Column
    private java.lang.Integer isbn = null;
```

Abbildung 34: Generierte Klasse „AbstractBuch“

In der generierten Klasse „AbstractBuch“ ist an der Annotation „@IdClass“ zu erkennen, dass es einen zusammengesetzten fachlichen Primärschlüssel gibt und eine Primärschlüssel Klasse „BuchPk“ generiert wurde (siehe Abbildung 34: Generierte Klasse „AbstractBuch“). Der zusammengesetzte Primärschlüssel besteht aus den Attributen „buchId“ und „auflage“, zu erkennen an der Annotation „@Id“. Ebenfalls erkennbar ist, dass jedes Attribut mit der Annotation „@Column“ annotiert ist, obwohl im Modell dies nicht definiert wurde (Standardfall).

```
/**
 *
 * Primary Key Class
 */
public class BuchPk implements Serializable {

    private final static long serialVersionUID = -1171125664L;

    @Id
    @Column
    public Integer buchId;

    @Id
    @Column
    public Integer auflage;
```

Abbildung 35: Generierte Primärschlüsselklasse „BuchPk.java“

In der Primärschlüsselklasse werden die Attribute nochmals mit den Annotationen „@Id“ und „@Column“ aufgeführt. Dadurch wird der zusammengesetzte fachliche Primärschlüssel richtig in die Datenbank gemappt. Diese Strategie stammt aus einer der vorgeschlagenen Mapping-Strategien des Hibernate⁶ Buches.

Im Package DAO sind die benötigten Klassen für die Standard CRUD-Operationen generiert worden. Die Klasse „GenericJpaDAO“ enthält dabei alle wichtigen Methoden dafür. Alle anderen Klassen benötigen die Klasse „GenericJpaDAO“ wiederum, um die Operationen ausführen zu können. Das DAO Package ist, wie von Hibernate⁷ beschrieben, aufgebaut. Das gesamte Package wird automatisch generiert und wird vom UML-Modell nicht beeinflusst.

Die durch den zusammengesetzten Primärschlüssel in der Klasse „Buch“ benötigte Primärschlüsselklasse „BuchPk“ wird im selben Package „JpaTutorial“ generiert, in dem auch die abstrakten Klassen und Interfaces liegen. Die Primär-

⁶ Siehe im Literaturverzeichnis den Eintrag „JAVA Persistence with Hibernate“

⁷ Siehe im Literaturverzeichnis den Eintrag „JAVA Persistence with Hibernate“

schlüsselklasse enthält alle wichtigen Angaben, die für ein korrektes Mapping benötigt werden. Sie ist wie von Hibernate⁸ empfohlen, aufgebaut.

Weitere Informationen zu DAO und Primärschlüsselklasse findet man auch im Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden..**

4.2.3 DDL-Generierung

Die generierte DDL-Datei kann zur Persistierung der Entitäten in eine beliebige Datenbank verwendet werden. Mit diesem Schritt sind alle notwendigen Schritte, von der Modellierung eines UML-Modells, über die Generierung einer Persistenzschicht in Java, bis hin zur Generierung der DDL-Datei, durchgeführt. Wie die DDL-Datei in eine Datenbank persistiert wird, ist nicht Gegenstand dieser Arbeit und wird hier nicht erklärt. Je nach Datenbanktyp ist dieser Schritt unterschiedlich durchzuführen. Klar ist aber, dass die Persistierung ebenfalls mit wenigen einfachen Schritten durchzuführen ist und der dafür zeitliche Aufwand sehr gering ausfällt.

Für die Generierung werden die Hibernate-Tools eingesetzt. Benötigte Bibliotheken befinden sich bereits im *lib* Verzeichnis und sind im Projekt eingebunden.

Die zur Generierung der DDL-Datei benötigten „persistence.xml“ ist im vorigen Generierungsprozess bereits automatisch erstellt worden. Sie befindet sich im Verzeichnis */Ihr Workspace/jpa.generator/src/main/java/META-INF*.

Um die Generierung der DDL-Datei durchzuführen, muss man die Datei „build.xml“, welche direkt im Basisverzeichnis des Projekts „jpa.generator“ liegt, mit einem *Rechtsklick – Run As – Ant Build* ausführen. Hierbei bitte aufpassen, es gibt auch die Möglichkeit *Ant Build...* anzuklicken. Bitte nur *Ant Build* ausführen.

Daraufhin wird die Datei „sql.ddl“ im Verzeichnis */Ihr Workspace/jpa.generator/ddl* generiert. Diese Datei sollte folgenden Inhalt haben:

⁸ Siehe im Literaturverzeichnis den Eintrag „JAVA Persistence with Hibernate“

```
create table APP.Autor (id integer not null, alter integer, nachname varchar(255), plz integer, str varchar(255), primary key (id));
create table APP.Buch (DTYPE varchar(31) not null, buchPk varbinary(255) not null, auflage integer not null,
    buchId integer not null, isbn integer, titel varchar(255), format varchar(255), Autor_id integer not null,
    primary key (buchPk, auflage, buchId));
create table APP.Verlag (id integer not null, plz integer, strasse varchar(255), verlagName varchar(255), primary key (id));
create table Autor_Verlag (Autor_id integer not null, verlag_id integer not null, primary key (Autor_id, verlag_id));
alter table APP.Buch add constraint FK1FC4183D6B4AC3 foreign key (Autor_id) references APP.Autor;
alter table Autor_Verlag add constraint FK77CF8A6B3D6B4AC3 foreign key (Autor_id) references APP.Autor;
alter table Autor_Verlag add constraint FK77CF8A6BCEA4A091 foreign key (verlag_id) references APP.Verlag;
```

Abbildung 36: Generierte DDL-Datei

Somit ist das Ende des Tutorials erreicht. Wenn man bedenkt, in welcher kurzen Zeit eine komplette Persistenzschicht in Java anhand eines UML-Modells generiert wurde, kann man schon grob einschätzen, was für einen großen Vorteil komplexere Softwareprojekte mit diesem Generator haben können.

5 Tutorial für die gemeinsame Verwendung des JPA- und BVA-Generators

In diesem Kapitel wird anhand eines Beispiels beschrieben, wie der JPA- und BVA-Generator gemeinsam verwendet werden kann. Für diesen Fall ist im Workspace ein weiterer Generator, mit dem Namen Common-Generator entwickelt worden.

Dieses Tutorial beschreibt nur die Schritte genauer, die im vorigen Beispiel noch nicht erklärt wurden.

Äquivalent zu den Projekten „*jpa.exporter*“, „*jpa.generator*“, „*bva.exporter*“, „*bva.generator*“, gibt es für den Common-Generator ein Projekt „*common.exporter*“ sowie „*common.generator*“. Beide Projekte müssen bereits im Workspace importiert sein.

3.6.1 UML-Modell

Da im vorherigen Tutorial bereits ein UML-Modell erstellt wurde, ist das jetzt benötigte UML-Modell mitgeliefert. Es befindet sich im droMDAry Package im Verzeichnis *Model* des Projektes „*common.exporter*“.

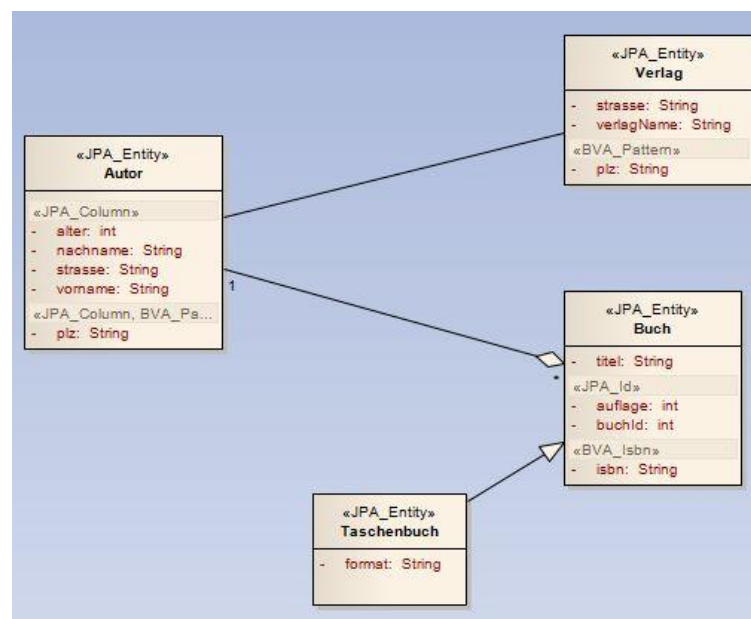


Abbildung 37: UML-Beispiel für den Common Generator

Wie man mit dem Enterprise Architect umgeht sowie weitere Informationen, kann man unter Kapitel 4 nachlesen. Detailliertere Informationen zum Thema Bean Validation API (BVA), findet man in der Diplomarbeit von Herrn Renz.

3.6.2 Exportvorgang aus dem Enterprise Architect in XMI

Der Exportvorgang wird im Projekt „common.exporter“ durchgeführt.

Die entsprechende Konfiguration der „workflow.properties“ Datei ist bereits im Auslieferungszustand des „common.exporter“ Projekts enthalten. Im Prinzip funktioniert die Konfiguration genau wie in Kapitel 4.2.1. Die Besonderheit ist, dass darin nun drei Profile konfiguriert sind. Ein Profil für den JPA-Generator, eines für den BVA-Generator sowie eines für den benutzerdefinierten Validator „Isbn“.

Wenn eine Trial Version des Enterprise Architect im Einsatz ist, erscheint der Evaluierungsdialog des Enterprise Architect für jedes Profil, also drei Mal.

Nachdem die Dateien „profile.oaw“ und „workflow.oaw“ erfolgreich ausgeführt wurden, ist das Profil und Modell aus dem Enterprise Architect in das XMI-Format vollständig exportiert.

3.6.3 Generierung von Javacode

Da nun der JPA- und der BVA-Generator eingesetzt werden kann, gibt es die Möglichkeit, im „common.generator“ Projekt in der „workflow.properties“ Datei, die Generatoren ein- oder aus zuschalten (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**).

```
# JPA-Cartridge benutzen -> generate.jpa = true
# JPA-Cartridge nicht benutzen -> generate.jpa = false
generate.jpa = true

# BVA-Cartridge benutzen -> generate.bva = true
# BVA-Cartridge nicht benutzen -> generate.bva = false
generate.bva = true
```

Abbildung 38: Konfigurationsmöglichkeit, welcher Generator im „common.generator“ Projekt zur Generierung verwendet werden soll

Hinweis: Der Common Generator ist vorerst nur zur Verwendung beider Generatoren gleichzeitig implementiert worden. Die Funktion, nur eine der beiden Generatoren einzusetzen, dient als Vorbereitung für später und ist zur Weiter-

entwicklung eingeplant. Erfolgreich getestet wurde aus Zeitgründen nur der gleichzeitige Einsatz beider Generatoren.

Alle weiteren benötigten Einstellungen sind bereits im Auslieferungszustand enthalten. Weitere Informationen dazu, sind unter Kapitel 4.2.2 bereits äquivalent für den „jpa.generator“ erklärt.

Die „jpa.properties“ müssen **immer**, auch aus dem „common.generator“ Projekt heraus, im „jpa.generator“ Projekt konfiguriert werden (siehe Kapitel 4.2.2).

Wie auch im „jpa.generator“ Projekt kann man über die Build Datei, welche direkt unter dem Verzeichnis *common.generator* liegt, die DDL-Datei generieren. In Kapitel 4.2.3 der Diplomarbeit erfährt man mehr darüber.

Nachdem die Datei „workflow.oaw“ erfolgreich ausgeführt wurde und man den JPA- sowie der BVA-Generator zur Generierung verwendet hat, ist in der generierten Datei „AbstractAutor“ folgendes zu sehen:

```
@javax.validation.constraints.Pattern(message = "Invalid PLZ", regexp = "[0-9]{5,5}")
@Column()
private java.lang.String plz = null;
```

Abbildung 39: Generiertes Attribut in der Klasse „AbstractAutor“

Für das Attribut „plz“ sind, wie im Modell angegeben, eine Annotation aus dem BVA-Generator sowie eine Annotation aus dem JPA-Generator erzeugt worden.

Die Implementierung des benutzerdefinierten BVA-Validators „Isbn“, findet man im Package „com.tsystems.bva.buchkatalog“ des BVA-Generators.

4 Wichtige Hinweise zur Modellierung mit dem Enterprise Architect

Dieses Kapitel beschreibt die wichtigsten Schritte, um in kürzester Zeit ein mit dem MDA-Generator kompatibles UML-Modell zu erstellen. Es gibt fast immer mehrere Wege die zum gewünschten Ziel führen. Im Kapitel wird jedoch nur ein Weg, der meiner Ansicht nach einfachste, erklärt. Für weitere Informationen ist eine Dokumentation vom Hersteller Sparx in englischer Sprache unter

http://www.sparxsystems.com/uml_tool_guide/index.html

verfügbar.

4.1 Neues Enterprise Architect Projekt erstellen

Um ein neues UML-Modell zu erstellen, muss zuerst ein *Enterprise Architect* Projekt angelegt werden. Hierzu bitte den Menüpunkt *File – New Project...* anklicken. Es öffnet sich ein Dialog zur Speicherung der Projektdatei. Wenn der Speicherort ausgewählt wurde, öffnet sich ein weiterer Dialog (siehe Abbildung 40: Screenshot – Auswahl des UML-Modell Typs) zur Auswahl des Modelltyps.

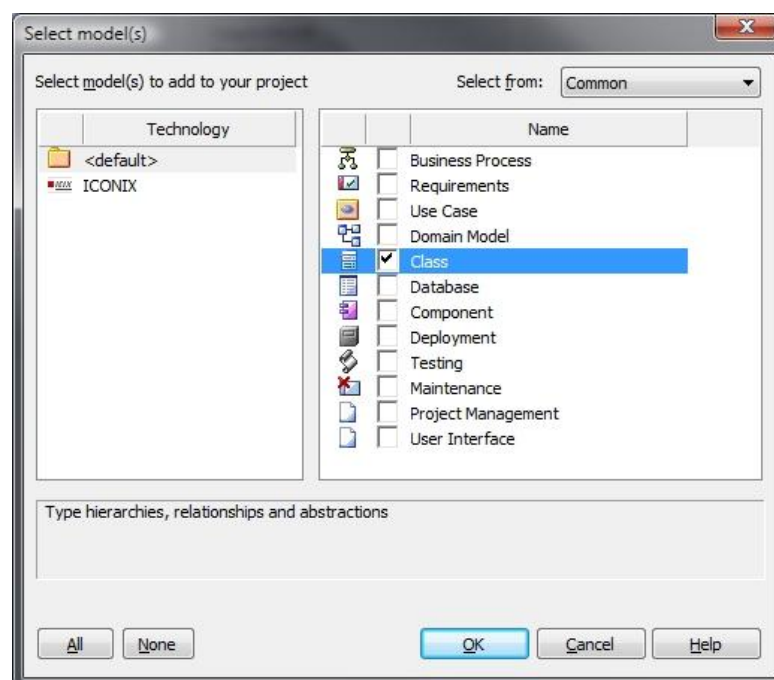


Abbildung 40: Screenshot – Auswahl des UML-Modell Typs

Hier ist *Class* auszuwählen, um ein kompatibles UML-Modell für den MDA-Generator zu entwerfen. Nachdem dieser Dialog mit *OK* bestätigt wurde, erscheint rechts in der View „Project Browser“ ein Package mit dem Namen „Class Model“. Dieses Package sowie das darunter liegende „System“ Package muss aufgeklappt werden, damit das Diagramm mit dem Namen „System“ geöffnet und geändert werden kann.

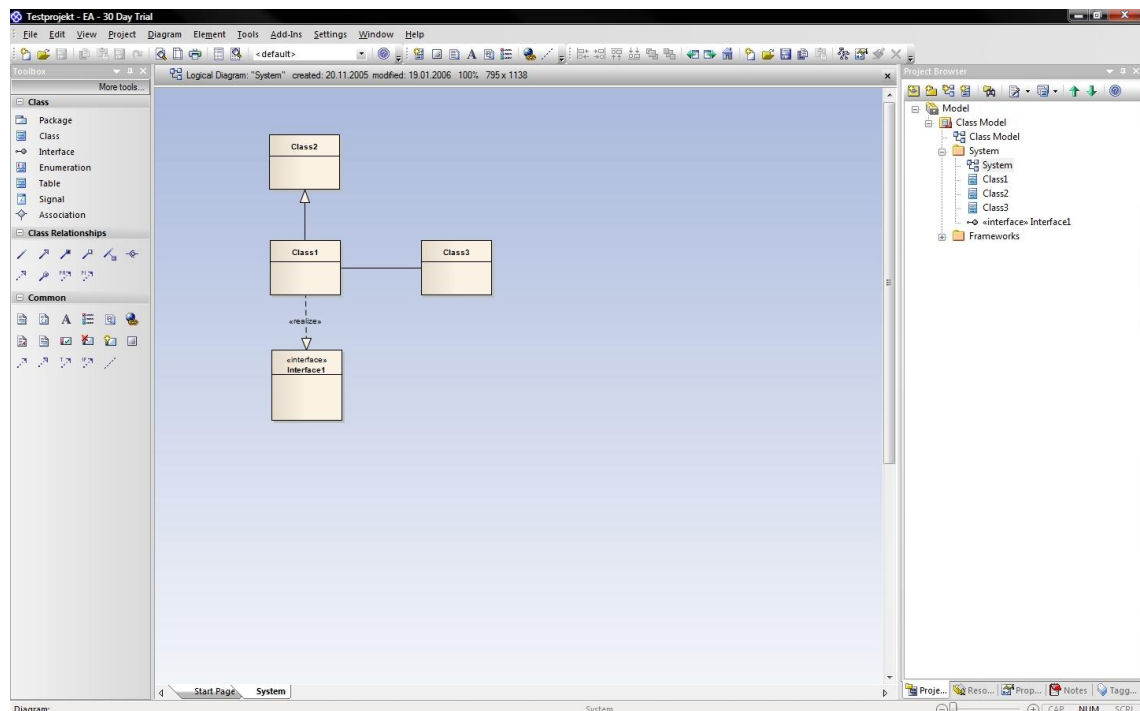




Abbildung 41: Screenshot - Neu erstelltes Enterprise Architect Projekt

Die durch den *Enterprise Architect* erzeugten Packages, Diagramme und Klassen können nun geändert und umbenannt werden. Um die Namen zu ändern, bitte im „Project Browser“ in den „Properties“, in die man per *Rechtsklick auf Package- oder Diagrammname – Properties* gelangt, das Feld Name entsprechend ändern. Das Symbol  steht für Package und das Symbol  für Diagramm.

4.2 Neue UML-Klasse erstellen

Eine neue UML-Klasse wird durch das Klicken auf *Class*, links in der „Toolbox View“ und dass anschließende Ziehen per Drag’n’Drop auf das Diagramm, erzeugt. Es öffnet sich anschließend der Klassen Dialog. In diesem Dialog können

sämtliche Einstellungen für die Klasse vorgenommen werden. Die Stereotypen kann man allerdings erst nach dem Einhängen des Profils angeben. Dieser Vorgang wird in Kapitel 4.6 erklärt.

4.3 Attribute erstellen und bearbeiten

Durch doppeltes Klicken auf die UML-Klasse im Diagramm öffnet sich der Klassen Dialog. Im Register *Details* gibt es die Möglichkeit den Button *Attributes...* anzuklicken (siehe Abbildung 42: Screenshot – Details Ansicht des Klassen Dialogs)

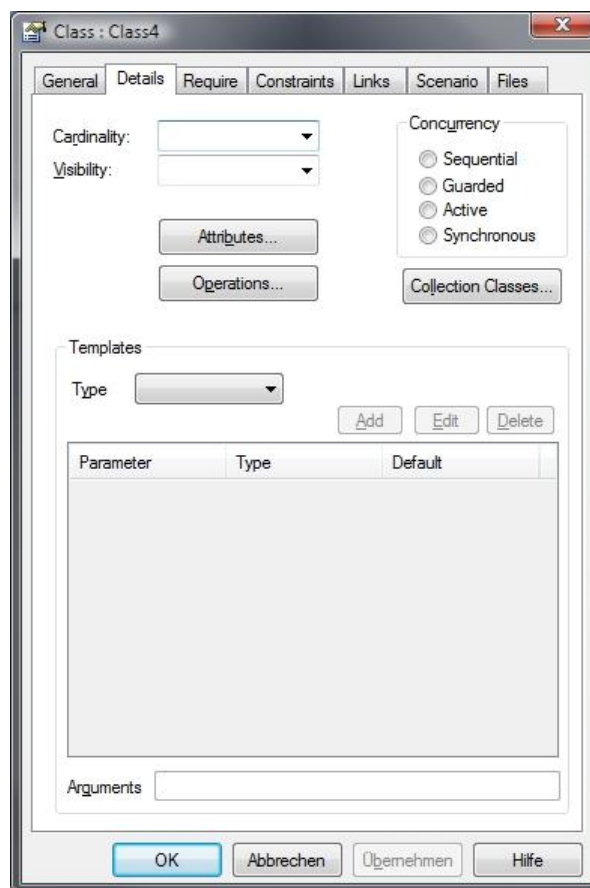


Abbildung 42: Screenshot – Details Ansicht des Klassen Dialogs

Daraufhin erscheint ein weiterer Dialog, um die Attribute anzulegen und zu bearbeiten. Nun müssen die Felder „Name“, „Type“ und „Scope“ ausgefüllt werden. Die Stereotypen können allerdings erst nach dem Einhängen des Profils angegeben werden. Dieser Vorgang wird in Kapitel 4.6 erklärt. Falls der Typ des Attributs nicht im Drop Down Menü zu finden ist, muss man diesen manuell in das Feld eintragen. Wichtig ist beim Anlegen eines Attributs, dass der

Save Button geklickt wird sobald die Felder fertig ausgefüllt wurden. Anschließend erscheint in der Attributliste im unteren Bereich des Dialogs das neu erzeugte Attribut (siehe Abbildung 43: Screenshot – Neu erstelltes Attribut in einer UML-Klasse).

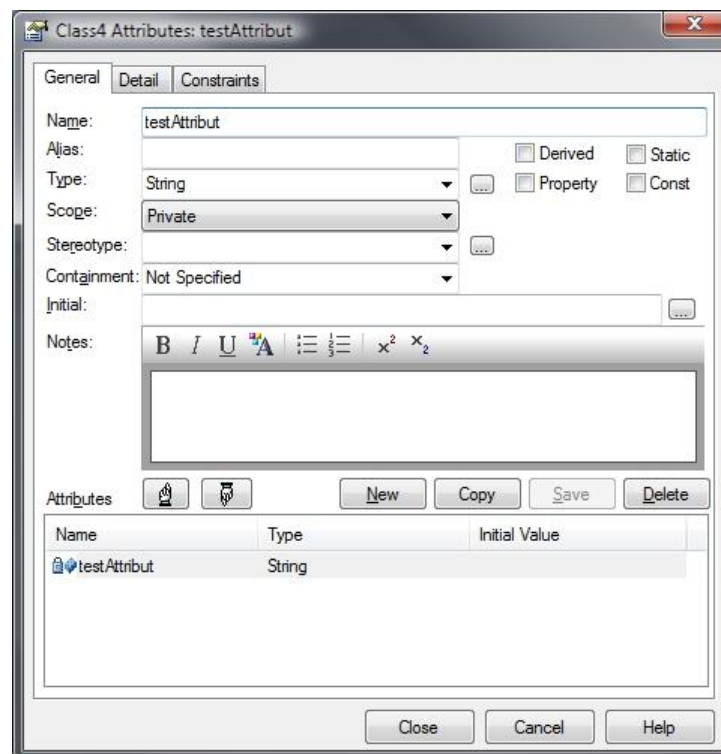


Abbildung 43: Screenshot – Neu erstelltes Attribut in einer UML-Klasse

Um ein Attribut zu bearbeiten, muss man dies in der Attributliste markieren. Nachdem das Attribut fertig bearbeitet wurde, sind mit dem Klick auf den *Save* Button die Änderungen abgespeichert. Wenn ein Attribut gelöscht werden soll, muss man dies in der Attributliste auswählen und durch den Klick auf den *Delete* Button löschen.

4.4 Methoden erstellen und bearbeiten

Wie auch für Attribute gibt es für Methoden einen eigenen Dialog. In diesen Dialog gelangt man durch doppeltes Klicken auf eine UML-Klasse im Diagramm, dann über den Register *Details* (siehe Abbildung 42: Screenshot – Details Ansicht des Klassen Dialogs) und über den *Operations...* Button in diesem Register. Dieser Dialog funktioniert ebenso wie der Dialog für Attribute (siehe Kapitel 4.3). Die Felder „Name“, „Return Type“ und falls es Übergabeparameter

gibt, das Feld „Parameters“, müssen angegeben werden. Die Stereotypen kann man allerdings erst nach dem Einhängen des Profils angeben. Dieser Vorgang wird in Kapitel 4.6 erklärt.

4.5 Beziehungen erstellen

Der einfachste Weg eine Beziehung zwischen zwei Klassen zu erstellen ist der, indem man eine Klasse durch einen einfachen Linksklick markiert. Es erscheint rechts oben an der markierten Klasse ein Pfeil (siehe Abbildung 44: Screenshot – Markierte UML-Klasse).

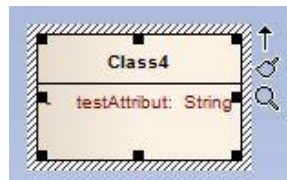


Abbildung 44: Screenshot – Markierte UML-Klasse

Durch einen einfachen Linksklick kann dieser Pfeil auf eine weitere UML-Klasse gezogen werden. Daraufhin erscheint ein Dialog zum Auswählen des Beziehungstyps. Wichtige Typen sind

- Association (Assoziation)
- Aggregation
- Composition (Komposition)
- Generalization (Ableitung).

Sobald der Beziehungstyp ausgewählt wurde, ist es notwendig die Kardinalität (im *Enterprise Architect* wird es Multiplicity genannt) anzugeben. Dazu klickt man doppelt auf die Beziehungsverbindung. Es erscheint ein Dialog mit mehreren Registern. Die beiden Register *Source Role* und *Target Role* sind für die Konfiguration interessant. Wie in Abbildung 45 gezeigt, ist die *Source Role* „TestKlasse1“ und die *Target Role* „TestKlasse2“. Man muss in beiden Registern die Multiplicity über ein Drop Down Menü angeben. Folgende Angaben sind möglich:

- **Source Role Multiplicity: 1, Target Role Multiplicity: 1** -> OneToOne
- **Source Role Multiplicity: 1, Target Role Multiplicity: *** -> OneToMany
- **Source Role Multiplicity: *, Target Role Multiplicity: 1** -> ManyToOne

- **Source Role Multiplicity: *, Target Role Multiplicity: * -> ManyToMany**

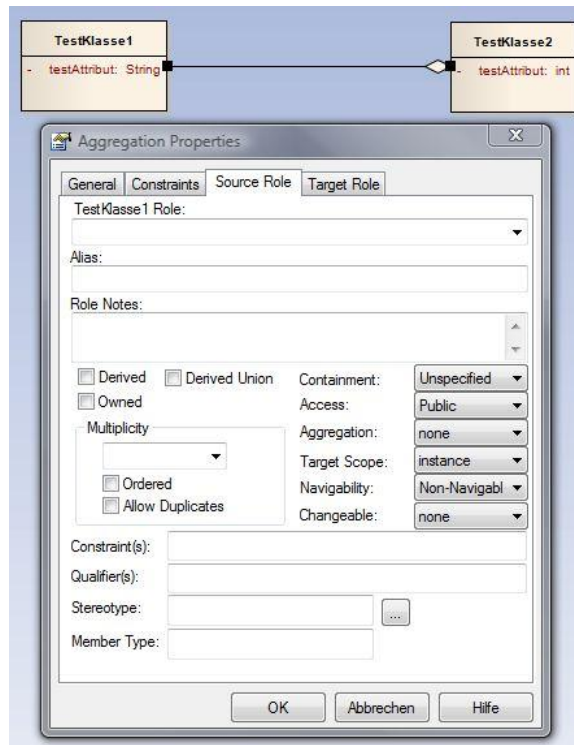


Abbildung 45: Screenshot – Kardinalität einer UML-Beziehung

4.6 UML-Profil einhängen

Falls die Resources View im *Enterprise Architect* noch nicht geöffnet ist, kann man diese über den Menüpunkt *View – Resources* öffnen. Um ein UML-Profil auszuwählen, kann per Rechtsklick auf *UML Profiles – Import Profile* und im sich daraufhin öffnenden Dialog der Button ... (der Button hat tatsächlich den Namen „...“) geklickt werden. Ein UML-Profil ist stets im XML⁹-Format angegeben. Sobald das UML-Profil über den Datei-Browser ausgewählt wurde, kann über den *Import* Button das UML-Profil in das *Enterprise Architect* Projekt eingehängt werden. Die Checkboxes bitte alle im Zustand deaktiviert lassen.

⁹ Die Extensible Markup Language (engl. für „erweiterbare Auszeichnungssprache“), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten. XML wird u. a. für den Austausch von Daten zwischen Computersystemen eingesetzt, speziell über das Internet.

4.7 Stereotypen auf ein UML-Element anwenden und löschen

Wenn ein Stereotyp aus einem bestimmten UML-Profil wie z.B. aus dem JPA oder BVA Profil auf ein UML-Element angewendet werden soll, ist das UML-Profil wie in Kapitel 4.6 beschrieben, in das *Enterprise Architect* Projekt einzuhängen. Zudem wird die *Resources* View im *Enterprise Architect* benötigt. Falls diese noch nicht geöffnet ist, kann man sie über den Menüpunkt *View – Resources* öffnen.

Unter *UML Profiles* sind die eingehängten Profile zu sehen. Die sich im Profil befindlichen Stereotypen werden darin angezeigt, wenn das Profil in der *Resources* View aufgeklappt ist (siehe Abbildung 46: Screenshot – UML-Profil in der Resources View).

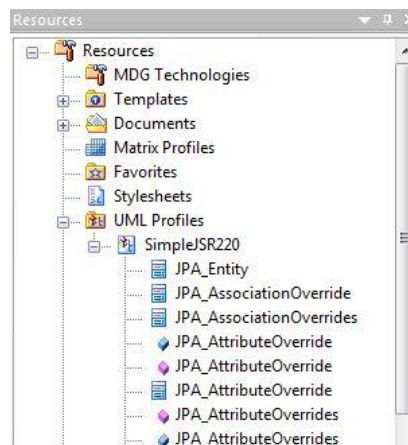





Abbildung 46: Screenshot – UML-Profil in der Resources View

Die Stereotypen werden mit drei verschiedenen Symbolen versehen:

-  Dieses Symbol bedeutet, der Stereotyp ist auf eine Klasse anwendbar
-  Dieses Symbol bedeutet, der Stereotyp ist auf ein Attribut anwendbar
-  Dieses Symbol bedeutet, der Stereotyp ist auf eine Methode anwendbar

Um einen Stereotyp auf ein UML-Element (Klasse, Attribut oder Methode) anzuwenden, ist zuerst die Klasse im Diagramm mit einem einfachen Linksklick und anschließend das gewünschte Element mit einem weiteren einfachen Linksklick zu markieren. Das markierte UML-Element ist am weißen Hinter-

grund zu erkennen (siehe Abbildung 47: Screenshot – Beispiel eines markierten UML-Elements).



Abbildung 47: Screenshot – Beispiel eines markierten UML-Elements (hier Attribut)

Sobald ein UML-Element markiert ist, kann in der *Resources* View der entsprechende Stereotyp ausgewählt und per Drag’n’Drop auf das UML-Element gezogen werden. Der Stereotyp erscheint anschließend über dem entsprechenden Element. Diese Vorgehensweise ist für alle UML-Elemente gleich.

Das Löschen von Stereotypen erfolgt je nachdem auf welches UML-Element er angewendet wurde, über den Klassen Dialog (siehe Kapitel 4.2), oder über den Dialog für das Bearbeiten von Attributen (siehe Kapitel 4.3) sowie über den Dialog für das Bearbeiten von Methoden (siehe Kapitel 4.4).

Vorsicht: Sobald ein Stereotyp Tagged Values enthält, löscht der *Enterprise Architect* die Tagged Values nicht mit. Hier ist es wichtig, die zu dem Stereotyp gehörigen Tagged Values in der Tagged Values View mit dem *Delete* Button zu löschen. Es ist jederzeit möglich, im mitgelieferten Profil nachzuschauen, welche Tagged Values zu einem Stereotyp gehören. Die Tagged Values sind als Attribute modelliert. In Abbildung 48 ist der Stereotyp „OneToMany“ mit seinen Tagged Values „cascade“, „fetch“, „mappedBy“ sowie „targetEntity“ abgebildet.

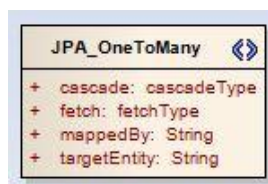


Abbildung 48: Screenshot – Stereotyp „OneToMany“ mit Tagged Values

4.8 Tagged Values bearbeiten

Tagged Values kann man erst bearbeiten, sobald ein Stereotyp auf ein UML-Element angewendet wurde (siehe Kapitel 4.7) und der Stereotyp Tagged Values enthält.

Um sich Tagged Values anzeigen zu lassen, muss die View Tagged Values geöffnet sein. Diese wird über das Menü *View – Tagged Values* geöffnet. Anschließend wählt man ein UML-Element (Klasse, Attribut oder Methode) aus. Dafür bitte zuerst die Klasse im Diagramm mit einem einfachen Linksklick und anschließend das gewünschte Element mit einem weiteren einfachen Linksklick markieren. Das markierte UML-Element ist am weißen Hintergrund zu erkennen (siehe Abbildung 47: Screenshot – Beispiel eines markierten UML-Elements). Wenn man in die Resources View wechselt, sind die entsprechenden Tagged Values zu sehen (siehe Abbildung 49: Screenshot – Beispiel Tagged Values View). Vorausgesetzt der Stereotyp beinhaltet Tagged Values.

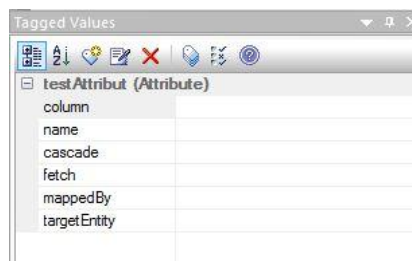


Abbildung 49: Screenshot – Beispiel Tagged Values View für den Stereotyp „OneToMany“

Die Inhalte der Tagged Values kann man nun im rechten Feld anpassen. Falls es sich um ein Tagged Value vom Typ ENUM handelt, wird im entsprechenden Feld ein Drop Down Menü angeboten (siehe Abbildung 50: Screenshot – Tagged Value vom Typ ENUM).

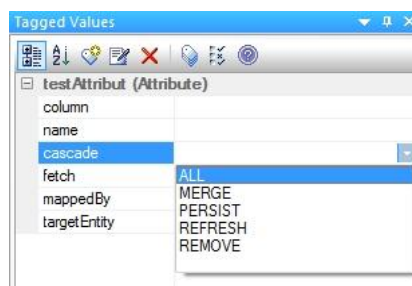


Abbildung 50: Screenshot – Tagged Value vom Typ ENUM

Falls mehrere Stereotypen auf das gleiche UML-Element angewendet wurden und diese Tagged Values enthalten, wird es sehr schnell unübersichtlich welcher Tagged Value zu welchem Stereotyp gehört. Dafür gibt es die Option *Show Fully Qualified Tags* (siehe Abbildung 51: Screenshot – Tagged Value View mit Option *Show Fully Qualified Tags*). **Vorsicht:** Wenn diese Option aktiviert

ist, dürfen die Tagged Values nicht verändert werden. Vorher muss die Option erst wieder deaktiviert werden (siehe Kapitel 4.9.1).

4.9 Bekannte Bugs

Es gibt leider einige bekannte Bugs im Zusammenhang mit dem *Enterprise Architect* und dem später erfolgenden Exportvorgang aus dem Projekt in das XMI-Format. Für diese Bugs wurden bisher keine Workarounds gefunden. Genau diese Bugs werden hier aufgeführt, da sie für die Modellierung ausschlaggebend sind. Um in den nächsten Versionen auf Bugfixes hoffen zu können, sind Tickets bei den entsprechenden Entwicklern eröffnet worden.

4.9.1 Tagged Values mit Option *Show Fully Qualified Tags*

Beschreibung:

Sobald in der Tagged Values View die Option *Show Fully Qualified Tags* ausgewählt ist und man in dieser View Tagged Values verändert, werden die Tagged Values im Generierungsprozess nicht berücksichtigt und fehlen somit im Javacode. Es ist aber dennoch möglich, sich über die Option *Show Fully Qualified Tags* kurz einen Überblick zu verschaffen, es dürfen lediglich keine Änderungen durchgeführt werden.

Eigentlich ist diese Funktion sehr hilfreich. Denn sobald mehrere Stereotypen auf ein UML-Element angewendet sind, erkennt man in der Tagged Values View leider nur noch an der Reihenfolge der angewendeten Stereotypen, welcher Tagged Value zu welchem Stereotyp gehört (siehe Abbildung 51: Screenshot – Tagged Value View mit Option *Show Fully Qualified Tags*).

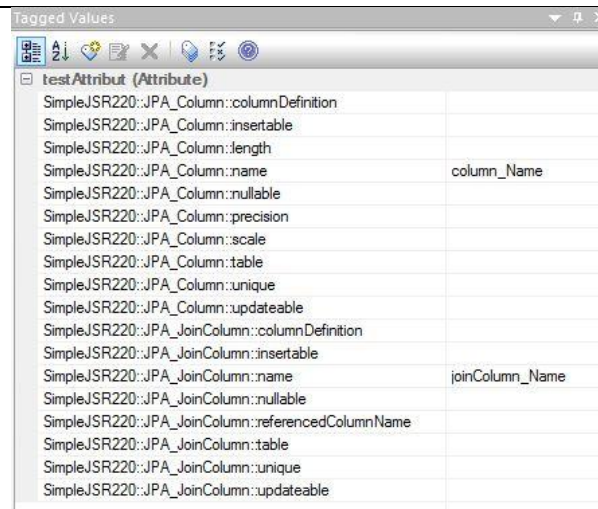


Abbildung 51: Screenshot –Tagged Value View mit Option *Show Fully Qualified Tags*

Fehlerbeseitigung:

Falls die Option *Show Fully Qualified Tags* aktiviert wurde, müssen folgende Schritte für jedes UML-Element auf das ein Stereotyp mit Tagged Values angewendet ist, durchgeführt werden:

1. UML-Element markieren (Hintergrund des Elements muss weiß sein)
2. In der Tagged Values View jeden Tagged Value mit dem *Delete* Button löschen
3. Den auf das UML-Element angewendeten Stereotyp entfernen

Anschließend ist es notwendig, in der Tagged Values View die Option *Show Fully Qualified Tags* wieder zu deaktivieren.

4.9.2 Mehrere gleiche Tagged Values für ein UML-Element

Beschreibung:

Wenn man auf dasselbe UML-Element mehrere Stereotypen anwendet, die eine gleiche Tagged Value Bezeichnung haben, wird der zuletzt gesetzte Tagged Value Wert für alle Tagged Values mit der gleichen Bezeichnung übernommen.

Beispiel:

In der Klasse „*TestKlasse1*“ ist auf das Attribut „*testAttribut*“ die Stereotypen „*JPA_Column*“ und „*JPA_JoinColumn*“ angewendet (siehe Abbildung 52:

Screenshot – Beispiel mit mehreren angewendeten Stereotypen auf ein Attribut.).

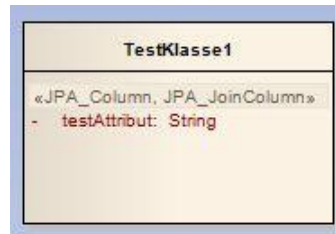
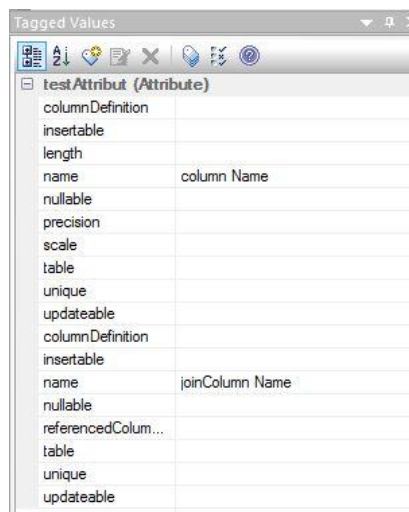


Abbildung 52: Screenshot – Beispiel mit mehreren angewendeten Stereotypen auf ein Attribut.

Beide Stereotypen haben ein Tagged Value „name“ (siehe Abbildung 53: Screenshot – Beispiel mit mehreren gleichen Tagged Values für ein UML-Element). Genau in so einem Fall haben im später generierten Code beide Tagged Values den Wert „joinColumnName“ und nicht wie erwartet der Tagged Value „JPA_Column::name“ den Wert „columnName“ und „JPA_JoinColumn::name“ den Wert „joinColumnName“.



Tagged Values	
testAttribut (Attribute)	
columnDefinition	
insertable	
length	
name	column Name
nullable	
precision	
scale	
table	
unique	
updateable	
columnDefinition	
insertable	
name	joinColumn Name
nullable	
referencedColumn...	
table	
unique	
updateable	

Abbildung 53: Screenshot – Beispiel mit mehreren gleichen Tagged Values für ein UML-Element

Fehlerbeseitigung:

Für diesen Fall gibt es derzeit keine Fehlerbeseitigung.

4.9.3 Beim Löschen von Stereotypen werden zugehörige Tagged Values nicht mit gelöscht

Beschreibung:

Sobald auf ein UML-Element angewendeter Stereotyp gelöscht wird und dieser Tagged Values enthält, werden die Tagged Values nicht mit gelöscht. Sie sind weiterhin in der „Tagged Values View“ zu sehen.

Fehlerbeseitigung:

In der „Tagged Values View“ muss jeder Tagged Value, der zu dem gelöschten Stereotyp gehört, über den *Delete* Button gelöscht werden.



6 Abbildungsverzeichnis

Abbildung 1: Screenshot vom Workspace Launcher	13
Abbildung 2: Screenshot vom Preferences Dialog des Java Compilers	14
Abbildung 3: Screenshot vom Eclipse Software Update Manager	15
Abbildung 4: Screenshot vom Preferences Dialog zum Verwalten der verfügbaren Metamodelle in <i>openArchitectureWare</i>	16
Abbildung 32: UML2 Profile in openArchitectureWare aktivieren	16
Abbildung 6: Screenshot des Importassistenten	17
Abbildung 7: Screenshot des Assistenten zum auswählen der Cartridges	18
Abbildung 8: Screenshot des Importvorgangs.....	19
Abbildung 9: Screenshot des Project Explorers nach dem erfolgreichen Import.....	19
Abbildung 10: Einfaches UML-Modell für das BVA Beispiel	23
Abbildung 11: Screenshot der Tagged Values des Stereotyp "BVA_Pattern"	24
Abbildung 12: Beispiel Profilstruktur für benutzerdefinierte Validatoren.....	25
Abbildung 13: Screenshot vom "New Model Package" Dialog im EA.....	26
Abbildung 14: Screenshot Projektstruktur ohne Stereotyp <<profile>>	26
Abbildung 15: Screenshot vom Package Dialog im EA.....	27
Abbildung 16: Screenshot der Profil Toolbox im EA.....	27
Abbildung 17: Screenshot des Create New Metaclass Dialogs im EA	28
Abbildung 18: Screenshot der BVA_Isbn Attribute.....	29
Abbildung 19: Stereotyp mit eingeblendetem Quick Link Symbol.....	30
Abbildung 20: Ableiten von einer Metaklasse	30
Abbildung 21: Screenshot vom Profil Export Dialog	31
Abbildung 22: Quellcode der Java Annotation @Isbn	32
Abbildung 23: Quellcodeausschnitt der Klasse IsbnValidator	33
Abbildung 24: Konfiguration des Modells mit der exporter.properties Datei ..	34
Abbildung 25: Konfiguration des benutzerdefinierten Profils mit der exporter.properties Datei	35
Abbildung 23: Überblick über das Tutorial	41
Abbildung 27: UML-Beispielmodell	45
Abbildung 28: workflow.properties – Konfiguration des Profil Exportvorgangs.....	46
Abbildung 29: workflow.properties – Konfiguration des Modell Exportvorgangs.....	47
Abbildung 30: Konfiguration der JPA-Cartridge in der Datei „workflow.properties“.....	49
Abbildung 31: Konfiguration der Javabasic-Cartridge in der Datei „workflow.properties“	50

Abbildung 32: JPA Konfiguration in der Datei „jpa-properties“	51
Abbildung 33: Generierte Klasse „AbstractAutor“	52
Abbildung 34: Generierte Klasse „AbstractBuch“	53
Abbildung 35: Generierte Primärschlüsselklasse „BuchPk.java“	54
Abbildung 36: Generierte DDL-Datei.....	56
Abbildung 37: UML-Beispiel für den Common Generator.....	57
Abbildung 38: Konfigurationsmöglichkeit, welcher Generator im „common.generator“ Projekt zur Generierung verwendet werden soll ..	58
Abbildung 39: Generiertes Attribut in der Klasse „AbstractAutor“	59
Abbildung 40: Screenshot – Auswahl des UML-Modell Typs	60
Abbildung 41: Screenshot - Neu erstelltes Enterprise Architect Projekt	61
Abbildung 42: Screenshot – Details Ansicht des Klassen Dialogs	62
Abbildung 43: Screenshot – Neu erstelltes Attribut in einer UML-Klasse	63
Abbildung 44: Screenshot – Markierte UML-Klasse.....	64
Abbildung 45: Screenshot – Kardinalität einer UML-Beziehung	65
Abbildung 46: Screenshot – UML-Profil in der Resources View.....	66
Abbildung 47: Screenshot – Beispiel eines markierten UML-Elements (hier Attribut)	67
Abbildung 48: Screenshot – Stereotyp „OneToMany“ mit Tagged Values	67
Abbildung 49: Screenshot – Beispiel Tagged Values View für den Stereotyp „OneToMany“	68
Abbildung 50: Screenshot – Tagged Value vom Typ ENUM.....	68
Abbildung 51: Screenshot –Tagged Value View mit Option <i>Show Fully Qualified Tags</i>	70
Abbildung 52: Screenshot – Beispiel mit mehreren angewendeten Stereotypen auf ein Attribut.	71
Abbildung 53: Screenshot – Beispiel mit mehreren gleichen Tagged Values für ein UML-Element	71

7 Tabellenverzeichnis

Tabelle 1: Tagged Values für die PLZ Validierung	24
Tabelle 2: Attribute des BVA_Isbn Stereotyps	29

8 Literaturverzeichnis

Brawand, U. (2009). *UML2Exporter*. Von <http://uml2ea.blogspot.com/> abgerufen

Eclipse Foundation and Others. (2009). *Eclipse Modeling Framework Project*. Von <http://www.eclipse.org/modeling/emf/> abgerufen

itemis AG. (2009). *JavaBasic-Cartridge*. Von [http://fornax.itemis.de/confluence/display/fornax/JavaBasic+\(CJB\)](http://fornax.itemis.de/confluence/display/fornax/JavaBasic+(CJB)) abgerufen

Java Community Process (verschiedene Firmen beteiligt). (16. März 2009). *JSR 303: Bean Validation 1.0 RC1 Proposed Final Draft*. Von <http://www.jcp.org/en/jsr/summary?id=303> abgerufen

No Magic Inc. (2009). *Magi Draw*. Von <http://www.magicdraw.com/> abgerufen

Object Management Group. (2009). *OMG Model Driven Architecture*. Von <http://www.omg.org/mda/> abgerufen

Object Management Group. (2009). *OMG Modeling and Metadata Specification*. Von http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML abgerufen

openArchitectureWare. (2009). *openArchitectureWare Projekt*. Von <http://www.openarchitectureware.org> abgerufen

Wikipedia Foundation and Others. (2009). *Wikipedia (English)*. Von <http://en.wikipedia.org/> abgerufen

Wikipedia Stiftung und Andere. (2009). *Wikipedia (deutsch)*. Von <http://de.wikipedia.org/> abgerufen

9 Stichwortverzeichnis

- Ableitung 38
- Aggregation 38, 59
- Annotation 27, 37, 38, 47, 48, 53
- Annotationen 46, 48
- Assoziation 59
- Attribut 36, 37, 47, 48, 57, 61
- Attribute 35, 37, 38, 48
- Beispiel Modell 18, 35
- benutzerdefinierter Stereotyp 23
- benutzerdefinierter Validator 21, 27
- Beziehung 59
- Beziehungen 38, 47
- Bug 64
- BVA 51, 53, 61
- bva.exporter 13
- bva.generator 13, 18, 32
- Cartridge 13
- common.exporter 13
- common.generator 13
- CRUD-Operationen 48
- DDL-Datei 34, 49, 53
- Eclipse 6, 7
- Eclipse Modeling Framework 10
- Enterprise Architect 7, 37, 52, 54
- exporter.properties 30
- Generalisierung 59
- Importieren 15
- Installation 7
- Java Compiler compliance Levels 9
- JDK 7
- jpa* 35, 40, 49
- JPA 61
- jpa.exporter 13
- jpa.generator 13
- Kardinalität 59
- Klasse 36, 45, 48, 61
- Komposition 59
- ManyToMany 59
- ManyToOne 59
- Mapping-Strategien 48
- Metaklasse 23, 26
- Methode 47, 61
- Methoden 48
- Multiplicity 38, 59
- oAW 39
- OneToMany 38, 47, 59
- OneToOne 59
- openArchitectureWare 7, 39
- Persistenzschicht 36, 49, 50
- Persistierung 49
- Primärschlüssel 37, 48
- Primärschlüsselklasse 37, 48
- profile 39
- Show Fully Qualified Tags* 64
- SimpleJSR303* 20
- Stereotyp 27, 37, 61
- Systemvoraussetzungen 6
- Tagged Value 20, 62, 64
- Tagged Values** 38
- UML-Klasse 56
- UML-Profil 26, 60
- workflow 52
- Workflow 40
- Workspace 8, 9
- XMI 64



XMI-Format 41