

Evaluation of UML to OWL Approaches and Implementation of a Transformation Tool for Visual Paradigm and MS Visio

Andreas Grünwald

Christian Doppler Laboratory CDL-Flex, Institute of Software Technology and
Interactive Systems, Vienna University of Technology
Favoritenstrasse 9-11/188, AT 1040 Vienna, Austria
a.gruenw@gmail.com

Table of Contents

Evaluation of UML to OWL Approaches and Implementation of a Transformation Tool for Visual Paradigm and MS Visio.....	1
<i>Andreas Grünwald</i>	
1 Introduction.....	1
2 Evaluation of Existing Transformation Tools	2
2.1 Overview	2
2.2 Purebred Transformation Tools	3
2.3 Semantic Modelling Tools and Hyprid Tools	6
2.4 Further Transformation Literature.....	8
2.5 Evaluation Results	10
2.6 A Word about XML Metadata Interchange	12
2.7 Eclipse ATL Transformation Approach	14
2.8 Evaluation Conclusions	15
3 UML to OWL Transformation Tool by Vienna University of Technology	16
3.1 Functionality and Obtainability	16
3.2 Software Architecture	17
3.3 Time Complexity	26
3.4 Automated Testing	28
3.5 Use Case: Transformation of CDL-Flex Tank Model	30
3.6 Future Work	32
4 Conclusion	33
Appendix	37
A Reference Models.....	37
B Protégé Screenshot	39

1 Introduction

As the CDL-Flex at Vienna University of Technology¹ cooperates with partners in the areas of software technology, automation engineering and industrial automation systems, one of the basic requirements is to extract knowledge from cross domains into OWL ontologies. These ontologies are integrated into the Engineering Knowledge Base (EKB), enabling intercommunication of heterogeneous tools based on a semantically level within CDL-Flex's Engineering Service Bus (EngSB)². As the laboratory experienced, most of the engineers are familiar with UML class diagram notation. Hence, Visual Paradigm's UML editor³ is established to collect and share domain knowledge between project partners using UML's logical data model notation, primarily.⁴ Up to now, an expert of CDL-Flex has to reconstruct an equivalent OWL ontology using Protégé OWL editor⁵ to enable knowledge integration into the EKB framework.

To automatize the UML to OWL transformation process, research and evaluation of existing UML to OWL approaches have been done. The following chapter gives an overview of existing tools and summarizes evaluation results, pointing out some interesting solutions, e.g. Eclipse's ATL. The chapter concludes, why transformation of Visual Paradigm's UML data models failed.

Afterward, an own UML2OWL transformation tool, that has been developed during this thesis, and which meets the requirement of Visual Paradigm XMI 2.1⁶ transformation, will be introduced. The tool comes with a well-defined functional range and rests on a solid software architecture. Latter will be detailed in section 3.2 and eases adoption of supplementary XMI converters through an automated test approach based on provided reference models. Time complexity of the tool has been examined and transformations of Visual Paradigm XMI 2.1, MS Visio XMI 1.0 and ArgoUML XMI 2.1 class diagrams have been carried out. Finally, the tool has been applied to a specific CDL-Flex use case, whereas some additional optimization effort originated.

¹ Christian Doppler Laboratory - Software Engineering Integration for Flexible Automation Systems

² <http://openengsb.org/>, last called 2011,06,07

³ <http://www.visual-paradigm.com/product/vpuml/>, last called 2011,06,07

⁴ <http://www.agiledata.org/essays/umlDataModelingProfile.html#TablesEntitiesViews>, last visit 2011-06-15

⁵ <http://protege.stanford.edu/>, last called 2011,06,07

⁶ <http://www.omg.org/spec/XMI/2.1>, last visit 2011-06-12

2 Evaluation of Existing Transformation Tools

The following chapter gives an overview of concrete, noteworthy UML to OWL transformation tools, followed by a brief description of (mostly) available tools. Technological approaches used for the transformation process are condensed and the most promising techniques are highlighted at the end of the chapter. Evaluation results will be presented in tabularized form. The chapter concludes, why transformation of XMI is difficult in practice and outlines ATL model transformation technology⁷.

2.1 Overview

Because large vendors tend to come up with new ontology modeling solutions currently, existing tools have been grouped into "purebred" UML to OWL transformation and hybrid semantic modeling tools during evaluation phase. Latter might be interesting for CDL-Flex in the long term.

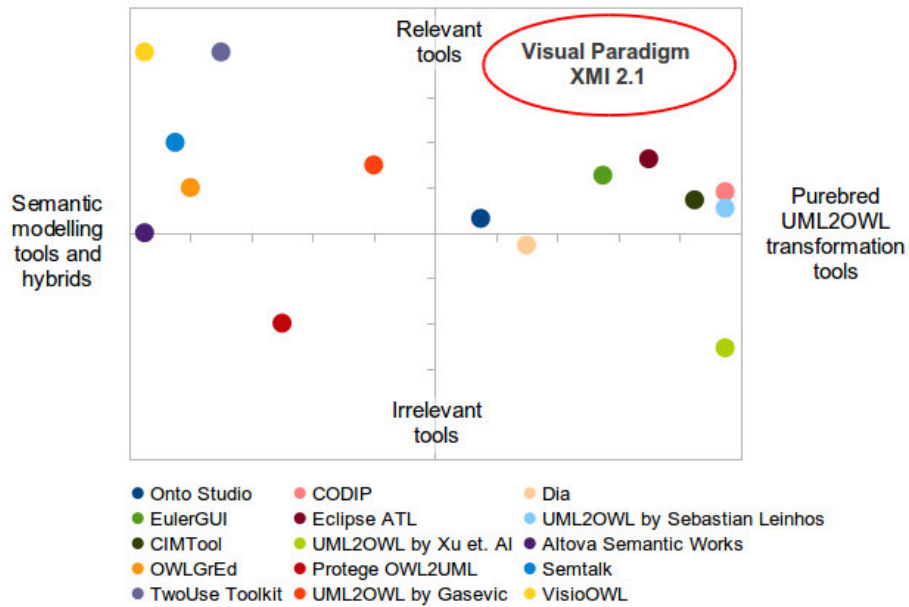


Figure 2.1: UML/OWL modeling tools classified by relevance and technology

Most information about transformation tools has been collected either by browsing scientific research papers (e.g. [14]) or by tracing Web forums and project

⁷ <http://www.eclipse.org/atl/>, last visit 2011-06-10

websites. For some tools, the author successfully has been contacted for information exchange, (e.g. [11]) while in other cases no answer has been received⁸. A dozen of purebred transformation tools have been esteemed as further interesting. One prerequisite for evaluation was, that the tool must be executable and claims to transform UML into OWL ontologies. During research, some modern, graphically sophisticated tools have been discovered, which often do not support transformation of UML diagrams directly, but offer modeling of ontologies using an advanced UML syntax (therefore, they are called "hybrids").

Figure 2.1 depicts, that none of the tools is suitable for transformation of Visual Paradigm XMI 1.2 or 2.1. The most relevant solution may be Eclipse's ATL approach, which will be discussed later. Although Xu et al. describe a very interesting transformation solution [14], their tool is not available, hence it has been classified as irrelevant. Rating of purebred transformation tools is based upon evaluation results, which will be described consecutively. Hybrids have not been evaluated as detailed as purebred evaluation tools, but were inspected and classified by relevance. The following section gives a short overview of pictured tools and their evaluation, to develop an intuition of which tools are available at present (in alphabetical order).

2.2 Purebred Transformation Tools

CODIP

Hartl and Emery provide a collection of Java based modules and systems for the processing, creation and management of OWL ontologies in a project named CODIP⁹. They host a command line tool called Duet, that claims to support Object Management Group (OMG)'s¹⁰ UML/XMI 1.2 standards. It is based upon Netbeans MDR¹¹, a metadata repository which is an extended implementation of MOF¹², XMI¹³ and JMI¹⁴ [12, pg. 1]. MDR can be used to load any MOF meta model and store instances of that model, for instance UML classes, either using Java API or XMI import. For instance, Java developers might define a meta model consisting of UML classes, attributes and associations and load concrete UML diagram instances (metadata) into the repository. Thus, transformation into MDR is an intermediate step of model to model transformation, i.e. to generate Java source code or OWL from UML. [12, pg. 5,8]

⁸ <http://www.umlowlgen.com/>, last visit 2011-06-11

⁹ <http://projects.semwebcentral.org/projects/codip/>

¹⁰ <http://www.omg.org/spec/>, last visit 2011-03-22

¹¹ <http://netbeans.org/about/press/8.html>, last visit 2011-06-10

¹² <http://www.omg.org/mof/>, last visit 2011-06-11

¹³ http://www.omg.org/technology/documents/modeling_spec_catalog.htm#XMI, last visit 2011-06-10

¹⁴ <http://java.sun.com/products/jmi/>, last called 2011-06-10

Dia

Surprisingly, the drawing/modeling tool Dia¹⁵ (at least the Debian version) contains an UML to OWL converter. On Ubuntu, the XSLT script, that is used for transformation, is located at `/usr/share/dia/xslt/dia-uml2owl.xsl`. The OWL converter can be chosen during UML export by selecting "XSLT transformation filter" and specifying OWL as the output format. Even Dan Connolly, the author of the tool, seemed to be surprised that his script has been integrated into Dia.¹⁶

Dia supports import of MS Visio¹⁷ diagrams (in vdx format¹⁸) and therefore seemed to be relevant for CDL-Flex. Unfortunately, export functionality failed during evaluation. Neither UML class diagrams designed in Dia, nor imported MS Visio class diagrams could be exported successfully (Dia generated an empty file and throw an error message). Furthermore, user interface and usability have been rated low, hence Dia is not a candidate for the CDL-Flex EKB framework.

EulerGUI

EulerGUI is a reasoning engine and lightweight Artificial Intelligence IDE, that internally is based on N3 (turtle) syntax.¹⁹ Amongst other formats, EulerGUI supports OWL, UML (XMI 2.1) and SPARQL queries as source formats. It provides a diligent manual, containing an UML to OWL transformation guide.²⁰ The graphical interface is based on Java AWT and straightforward, although n3 graph visualization did not work. Transformation of EulerGUI UML sample files performed well, but transformation of Visual Paradigm XMI 2.1 failed. Note, that EulerGUI returns transformation results in N3²¹ format, which have to be translated into OWL RDF/XML before they can be imported into Protégé. Therefore, online converters, such as <http://www.mindswap.org/2002/rdfconvert/> are helpful.

ODM Implementation (Bridging UML and OWL) based on ATL

One of the transformation use cases for ATL is the ODM implementation (Ontology Definition Metamodel) for bridging UML and OWL.²² ATL (ATLAS Transformation Language) is a model transformation language and Eclipse toolkit that provides ways to produce a set of target models from a set of source models.²³ It originally has been initiated by OMG and is part of Eclipse's M2M

¹⁵ <http://projects.gnome.org/dia/>, last visit 2011-06-10

¹⁶ <http://www.mail-archive.com/www-archive@w3.org/msg00195.html>, last visit 2011-06-10

¹⁷ <http://office.microsoft.com/de-de/visio/>, last visit 2011-06-10

¹⁸ <http://msdn.microsoft.com/en-us/library/aa218409.aspx>, last visit 2011-06-11

¹⁹ <http://eulergui.sourceforge.net/>, last visit 2011-06-10

²⁰ <http://eulergui.sourceforge.net/documentation.html#Data>, last visit 2011-06-10

²¹ <http://www.w3.org/DesignIssues/Notation3>, last visit 2011-06-11

²² <http://www.eclipse.org/m2m/atl/usecases/ODMImplementation>, last visit 2011-06-11

²³ <http://www.eclipse.org/atl>, last visit 2011-06-11

(Model to Model transformation) project.²⁴ Hillairet, the author of the tool, provides the most current project files (including samples) on his homepage.²⁵ Also the source files for reverse transformation (OWL2UML) are available there. Both comprise an *ant*²⁶ file, which must be executed inside Eclipse. Evaluation showed, that the tool transforms XMI 2.1 UML models, that have been designed with Eclipse UML2 plugin²⁷ or Papyrus²⁸, which both implement OMG's XMI 2.1 standard, accurately. Transformation of Visual Paradigm XMI 2.1 caused exceptions and resulted in an empty OWL file.

ATL UML2OWL project requires Eclipse EMF (Modeling Framework), UML2 and AM3²⁹. Some of those components are still in incubation phase and incompatible either among themselves or with some versions of Eclipse. Thus, the environment is hard to install. AM3 plugins are only available via SVN and have to be integrated into Eclipse using plugin perspective. A well-oiled bundle of the required tools can be downloaded from http://www.eclipse.org/downloads/download.php?file=/modeling/m2m/at1/bundle/ATL_Bundle_2.0.0RC2_Windows.zip.

OntoStudio

OntoStudio is a commercial modeling environment for the creation and maintenance of ontologies.³⁰ It supports OWL, RDF(S), RIF, ObjectLogic and can import UML2.0, database schema's (MS-SQL, DB2, MySQL), Excel spreadsheets, MS Outlook mails and directory structures of filesystems. A three month free trial is available for download. OntoStudio only supports transformation of XMI 2.1 into ObjectLogic ontologies³¹, but does not support OWL ontology transformation. It is distributed by Ontoprise GMBH (Germany), which also develops OntoBroker³² - a middleware solution for Semantic Web.

UML2OWL Ontologieextraktion und -modellierung by Leinhos

In [11] S. Leinhos introduces two XSLT scripts: One for transforming UML to OWL and one to modell ontologies using UML notation. Both scripts are available for download.³³ The OWLfromUML script supports transformation of XMI 1.2 as implemented by Poseidon CE 4.1 [11, p. 44]. A very detailed description of how UML elements are transformed into their OWL counterparts can be found at [11, p. 65 ff]. Even transformation of association classes is supported.

Although the transformation script is based on a substantial written master thesis, the script fails for newer versions of Poseidon (e.g. CE 6.0) and does not support any XMI formats of Visual Paradigm and MS Visio UML.

²⁴ <http://www.eclipse.org/m2m>, last visit 2011-06-07

²⁵ <http://perso.univ-lr.fr/ghillair/projects.html>, last visit 2011-06-07

²⁶ <http://ant.apache.org/>, last visit 2011-06-11

²⁷ <http://www.eclipse.org/uml2/>, last visit 2011-06-11

²⁸ <http://www.eclipse.org/modeling/mdt/papyrus/>, last visit 2011-06-11

²⁹ <http://www.eclipse.org/gmt/am3/>, last called 2011-06-11

³⁰ <http://www.ontoprise.de/en/products/ontostudio/>, last visit 2011-06-11

³¹ <http://help.ontoprise.de/example.htm>, last visit 2011-06-11

³² <http://www.ontoprise.de/en/products/ontobroker/>, last visit 2011-06-07

³³ <http://diplom.ooyoo.de/index.php?page=download>, last visit 2011-06-11

CIMTool

During the project "Integration in the Electric Power Industry", Pacific Northwest National Laboratory developed an UML to OWL transformator, that is part of "CIM-Tool". CIM (Common Information Model) is a modeling standard in the electric power industry. As described in "Ontology and the Age of Integration in the Electric Power Industry" [6], the transformation is based on Simple API for XML (SAX). One main problem the project team encountered during development were the varying dialects of XMI, even more than one per UML tool. CIMTool has been implemented for Eclipse and is available for download on Langdale Consultants website.³⁴ Even the source code is available. It is well documented and provides a robust Eclipse workspace environment, but is intended to be used with CIM XMI, and therefore failed to import Visual Paradigm XMI 2.1.

UML-OWL Generator

UML-OWL Generator³⁵ is a machinery that claims to support the transformation of any UML data model in standard XMI format (class diagram) into OWL-DL ontologies. The UML-OWL Generator has been patented (patent pending) with the United States Patent and Trademark Office (USPTO). The only way to try it, is to email the UML file to info@umlowlgen.com and receive the corresponding OWL-DL ontologies within 24 hours. However, the sent transformation request has never been answered.

Xu et al. - UML2OWL

In [14] Xu et al. present a transformation algorithm called *U2OTrans* [14, p. 128-130], and its prototype implementation tool based on J2SE 1.5 and the DOM API. Xu et al. tested the tool against ten UML diagrams (XMI 2.0) with diagrams' range between 50 and 500 [14, p. 131-133]. The result was, that all created ontologies were syntactically correct, feasible and the tool was working efficient. The paper also states time complexity, which is $O(N)$, where N is the number of all UML elements (classes, attributes, generalizations, associations, roles) within the XMI structure. Unfortunately, the authors' UML2OWL tool is not available for download and a request to the author did not lead to a response.

2.3 Semantic Modelling Tools and Hybrid Tools

Altova Semantic Works

Altova's Semantic Works calls itself the *ground-breaking visual RDF and OWL editor for the Semantic Web*.³⁶ It supports graphically design for RDF instance documents, RDFS vocabularies, and OWL ontologies (different profiles) within code generation and functions for organizing Ontology instances, properties, classes and offers creation of Semantic Web applications. A 30 day trial version is available for free.

OWLGrEd

OWLGrEd (OWL Graphical Editor) is derived from the Protégé "OWL2UML" plugin.³⁷ Even though OWLGrEd is not an UML to OWL nor a OWL to UML transformation tool, it might be worth having a look at, because OWLGrEd uses an own, extended

³⁴ <http://wiki.cimtool.org/index.html>, last visit 2011-06-11

³⁵ <http://www.umlowlgen.com/>, last visit 2011-06-11

³⁶ <http://www.altova.com/semanticworks.html>, last visit 2011-06-11

³⁷ <http://protegewiki.stanford.edu/wiki/OWL2UML>

UML syntax approach and therefore can transform existing OWL ontologies into an ODM (Ontology Definition Metamodel) based, extended UML syntax. Description, installation guide, Protégé plug-in and standalone version can be downloaded from the project's website.³⁸

Protégé

Protégé 4.0 offers an OWLtoUML plug-in which empowers OWL visualization using an UML profile model (not vice versa).³⁹ While the plugin claims to be compatible with Protégé 4.0, installation of the current version showed that UML export is neither supported by Protégé 4.0, nor by Protégé 4.1 beta. Furthermore, as described above, the OWL2UML plugin has grown and has been transferred into the "OWLGrEd" project.

Semtalk OWL Edition

SemTalk is an extension for Microsoft Visio. It does not support UML to OWL transformation, but offers ontology modeling using syntax built on UML 2.0. Besides, also semantic business process modeling (EPK, BPMN, SAP processes and services) is offered.⁴⁰

TwoUse Toolkit

TwoUse Toolkit is a project that is sponsored by the European Commission. It implements current OMG and W3C standards for software design, code generation and OWL ontology engineering. All components are implemented as Eclipse plugins. The aim of TwoUse toolkit is to enable Semantic Web Software Engineering and Model Driven Semantic Web. Thus, besides extensive browsing and querying support (e.g. SPARQL), it offers different graphically design tools for modelling of OWL ontologies:

- UML editor for modelling and transformation of UML profiles into OWL ontology functional syntax.
- TwoUse Graphical Editor for directly creating ontologies (some design elements have been adopted from Protégé).

Examples, a demonstration video and Eclipse update URL are available on the project website.⁴¹ The ontology in figure 2.2 has been modeled using Eclipse Helios and TwoUse toolkit plugin and contains a SWRL - Rule (UncleRule).

UMLtoOWL: Converter from UML to OWL

In "Converting UML to OWL ontologies" [8] Gasevic et al. describe UML transformation based on XSLT. Gasevic provides an UMLtoOWL converter on his homepage⁴². The tool converts extended Ontology UML Profile (OUP) models from XMI into OWL ontologies. It can not convert UML models without OUP stereotypes. For conversion, a XSLT processor like Xalan (Java) is required. The tool supports UML object diagrams, whoms elements are transformed into OWL individuals.

³⁸ <http://owlgred.lumii.lv/>, last visit 2011-06-11

³⁹ <http://protegewiki.stanford.edu/wiki/OWL2UML>, last visit 2011-06-11

⁴⁰ <http://www.semtalk.com/semntation/semtalk.pdf>, last visit 2011-06-11

⁴¹ <http://code.google.com/p/twouse/>, last visit 2011-06-11

⁴² <http://www.sfu.ca/dgasevic/projects/UMLtoOWL>, last visit 2011-05-10

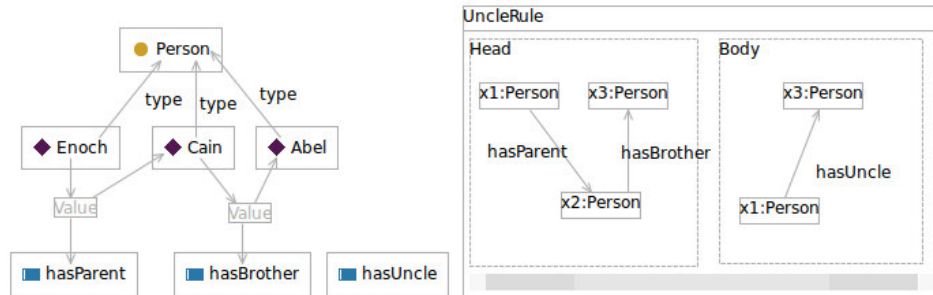


Figure 2.2: Demonstration of TwoUse toolkit model elements

VisioOWL

VisioOWL is a Microsoft Visio application to support the use of Visio for creating graphical representations of OWL ontologies. The implementation is intended to provide, as close as possible, a direct one-to-one mapping between the OWL language constructs and their graphical representation. The graphical representation of an OWL ontology may provide, for some developers and users, a more comprehensive insight into overall class and property relationships than could be garnered from the OWL markup alone.⁴³

2.4 Further Transformation Literature

There are some elder approaches that describe meta models for transformation of UML models into RDF and DAML (compare [1]). Although transformation approaches for DAML ontologies are useless for CDL-Flex purposes, some of those works contain interesting solution statements that build the base of existing transformation tools and are worth reading. For instance, Baclawski, et al. [1, p. 143–146] list a mapping between UML and DAML terminology.

Cranefield describes in [5], how to map UML to Java classes or RDF/XML schemas and discusses XSLT transformation approach in detail.

In *A Detailed Comparison of UML and OWL* [10, p. 38ff] Kiko and Atkinson illustrate mapping between UML elements and OWL constructs. Enumerations, disjoint classes, quantifiers and property relations (e.g. functional, inverse, transitive, symmetric) are covered as well as UML and OWL meta level hierarchies (UML four level OMF meta model vs. semantic layer model) and differences in closed (UML) and open world assumption (OWL) models. [10, p. 5ff]

Falkovych, Sabou and Stuckenschmidt in [7] outline two transformation approaches for DAML and OIL ontologies. Both approaches suggest different solutions for the mapping between UML attributes, UML associations and ontology properties. [7, p. 104] They deal with problem, that in UML, associations are related to exactly two endpoints, while ontology properties can have various domain and range classes. That implies,

⁴³ <http://mysite.verizon.net/jflynn12/>, last visit 2011-06-11

that one has to define unique naming conventions for attributes and associations, and furthermore has to decide, if different UML associations/attributes with the same label should be merged into a single ontology property or if for each of them a new property has to be defined.

In Baclawski's approach, UML stereotypes are proposed to map between UML attributes, UML associations and properties, which the designer has to add to the UML model. [7, p. 104] Falkovych's approach assumes, that attributes are only linked to data types (primitives), while associations link objects. Depending on the association type (binary, unidirectional, aggregation and composition), the associations are renamed into ontology properties. [7, p.99,100]

Brockmans et al. introduce further UML and SWRL rule modeling approaches based on UML profiles in in [4] and [3]. TwoUse toolkit uses the same syntax for rule modeling as proposed in [3, p. 311].

2.5 Evaluation Results

Table 1 summarizes evaluation results of the purebred transformation tools, that already have been addressed in section 2.2. The tools have been compared and evaluated based on portability, their transformation approach, up-to-dateness, availability of documentation, support, usability and surplus value for CDL-Flex.

	OntoStudio	CODIP	Dia U2O	EulerGUI	Eclipse ATL	U2O Leinhos	CIMTool	Xu et al.
Platform	Windows, Linux	indep.	Windows, Linux, Mac	indep.	indep.	indep.	Windows	indep.
Software technology	Eclipse based	Java	GUI (GTK)	Java AWT, C#, Python, JS, Prolog	Eclipse based	XSLT	Eclipse based	Java 1.5
Add. required components	-	-	-	n3 to XML converter, e.g. mindwap.org	EMF, UML2.0 plugins, ATL, AM3	SAX Parser	-	-
Transf. approach	Closed source	Netbeans MDR	XSLT plugin	N3 rule based	ATL	XSLT	Java SAX	Java DOM
Source code	Closed source	Java project as ZIP-file	C/XSLT	only parts available	ATL (and ant)	XSLT	Java repository (github)	-
Licence	royalty	BSD	GPL2	W3C Software Licence	EPL/CPL	not specified	GPL2.1	-
Current version	V3	Duet 0.5.3	0.97.1	1.8	1.0	1.1	1.8.3	-
Last update	2011	2004	2011	2011	2007	2006	2011	-
XML/XMI formats as specified	XMI 2.*	modified XMI 1.2	MS Visio vdx	XMI 2.*	XMI 2.* (Papyrus, Eclipse UML2)	Poseidon XMI 1.2	XMI 2.* based on CIM standards	XMI 2.0 (Magic-Draw, Poseidon)
Support	hotline	none	UNIX community (?)	less	Eclipse community	EU supported project	author	-
Documentation	online help	Duet GUIDE (doc-file)	poor (for dia-uml.xml)	excellent guide	scattered	How-to; Master thesis on request	Excellent wiki page	Paper (excellent)
Fulfills UML2OWL requirements	no	yes	partly	yes	yes	yes	only CIM models	yes
Passed reference model test	not tested further	yes	no	yes	yes	partly	yes	-
Passed VP XMI 1.2/2.1 test	no	no	no	no	no	no	no	-

Table 1: Comparison of UML2OWL transformation tools

None of the tools transformed Visual Paradigm XMI 1.2 or 2.1 into valid OWL DL. All scripts completed within a reasonable time (in other words: none of the programs hang up), but also none of the tools correctly transformed a single UML element into its valid OWL counterpart. The tools either terminated before an OWL file was created

or the root element of the resulting OWL XML/RDF file was empty.

Table 1 illustrates, that some of the transformation tools have not been updated since years. Most of the tools passed the reference model test, which means that they successfully translated their own examples (XMI files) into valid, Protégé compatible OWL ontologies.

Transformation of Leinhos' UML2OWL reference models failed when the models were opened in Poseidon for UML SE 6.0.2 and re-exported into XMI 1.2. Dia successfully imported MS Visio XSD files, but crashed during OWL export, because the Visio file could not be saved as a valid Dia file (tested with Ubuntu 11.04). OntoStudio seemed to be interesting and the graphical interface (Eclipse based) is quite nice, but evaluation showed, that import of UML 2 is only available for ObjectLogics ontologies. Eclipse offers an entire UML modeling environment (EMF; ATL; UML2 or Papyrus project) and ATL transformation performed well for the distributed XMI 2.1 reference examples, but the flip side of the coin is, that most of the Eclipse modeling tools⁴⁴ are still in incubation and therefore, setup of AM3⁴⁵ prototype, which is necessary for XML serialization, is very hard.

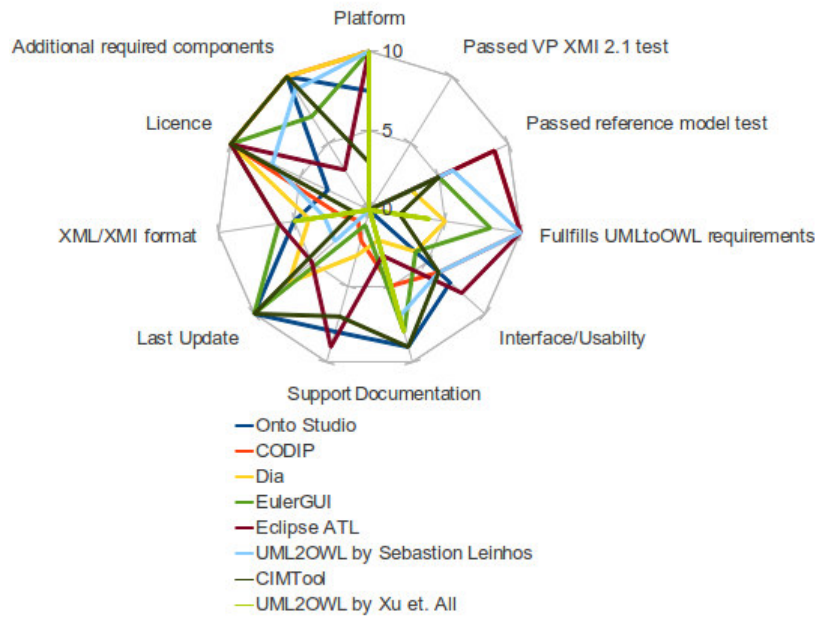


Figure 2.3: Result of evaluation. Rating of tools in categories between 0 and 10.

Evaluation results are illustrated in figure 2.3. Tools have been rated in categories (see

⁴⁴ <http://www.eclipse.org/downloads/>, last visit 2011-06-11

⁴⁵ <http://www.eclipse.org/gmt/am3/>, last visit 2011-06-11

table 1) between 0 and 10, where 0 is worst and 10 is *excellent*. For instance, OntoStudio comes with license costs (not published), and therefore *license* has been rated with 3, while CimTool (GPL 2.1) has scored high (10).

The net chart states, that most tools can be executed on several OS and are available for free. All tools provide documentation (some of them installation guides), but only three tools provide adequate support (hotline or community). Four tools fulfilled the UML2OWL transformation requirements, as specified by CDL-Flex, which means that they can transform UML 2.0 data diagrams, without additional UML profile notation, into OWL ontologies. Most of the tools support either XMI 1.2 or XMI 2.1 of a single UML vendor. UML2OWL solutions are rated higher in this category, if they define, which UML tools' syntax they support and which formats have been used for testing. Interface and usability rating depended on both, usability and type of interface, e.g. "Eclipse based" or "command line tool". GUI interfaces usually are rated higher, but only if the interface is straight forward to use and pliable.

2.6 A Word about XML Metadata Interchange

Evaluation showed, that the transformation process heavily depends on the format of the provided XMI (XML Metadata Interchange) file. Therefore, the central problems concerning XMI in general, and Visual Paradigm XMI in detail, will be summarized next.

XMI has been proposed by OMG⁴⁶ with the intend to provide a standard way to exchange information about metadata and is based on XML, UML and OMG's MOF (Meta Object Facility).⁴⁷ Today, XMI has been implemented by most of the UML tool market leaders. Even Microsoft offers downloadable plugins and macros for Visio 2010 to export XMI 1.2 version and above.⁴⁸

On www.uml-forum.com/FAQ.htm the question why interchange of UML models between modeling tools in practice often fail, and why it is difficult to parse XMI in a uniform manner, is put in a nutshell: While the XMI (XML Metadata Interchange) standard purports to facilitate the interchange of UML models, it has been largely ineffective in practice. There are at least two technical reasons for this. First and foremost, XMI attempts to solve a technical problem far more difficult than exchanging UML models; it attempts to provide a mechanism for facilitating the exchange of any language defined by the OMG's Metamodel Object Facility (MOF). Secondly, the UML 2.* Diagram Interchange specification lacks sufficient detail to facilitate reliable interchange of UML 2.* notations between modeling tools. Since UML is a visual modeling language, this shortcoming is a showstopper for many modelers who don't want to redraw their diagrams.⁴⁹

Although introduction of XMI 2.* standard tackled some issues regarding visualization of UML elements, vendor formats still are highly incompatible. To help establishing

⁴⁶ <http://www.comp.nus.edu.sg/~yangfei/HYP/XMI.pdf>, last visit 2011-06-12

⁴⁷ <http://searchsoa.techtarget.com/definition/XMI>, last visit 2011-06-07

⁴⁸ <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3DD3F3BE-656D-4830-A868-D0044406F57D>, last visit 2011-06-12

⁴⁹ http://www.uml-forum.com/FAQ.htm#Why_cant_I_easily_interchange_UML_models, last visit 2011-06-11

an unified exchange format in industry, one proposal might be to define a new XML schema that is tailor-made for UML and offers well-defined sub schemes either for each of the 13 UML diagram types⁵⁰ or for the most popular ones (e.g. class diagram, activity diagram, sequence diagram, use case diagram). The presentation component should be redesigned and treated separately. Therefor an equal schema should be used for all UML diagram types and kept in a separate namespace. This component should handle visualization of UML elements for multiple vendor solutions highly accurate and reliably. To accelerate an unique exchange format, a new OMG project might be launched, that offers API's and implementation proposals to ease integration for vendors.

The following list attempts to explain, why transformation of Visual Paradigm (consecutively called "VP") XMI 2.1 format into valid OWL failed using existing UML2OWL tools.

1. VP takes use of different XML namespaces. For instance, UML element types like **class**, **association** or **package** are prefixed with **uml**, while UML elements themselves, are located in the default namespace. Attributes again, e.g. ID, are located in **xmi** namespace. Existing tools might have problems to extract UML elements regarding different namespaces.
2. VP XMI structure is nested. XSLT scripts are likely to have problems, if XMI hierarchies are slightly different. Furthermore, to extract the whole information about an element, one has to extract different XML references and grab various XML elements across the whole XMI document, which in some cases can be a challenging task.
3. Structure of VP XMI depends on the UML editor view during XMI export process: If there are multiple diagrams, which VP organizes in *packages* (VP diagram and VP package are synonyms), UML **classes** and **associations** are nested inside the XMI package element they belong to, except the diagram that is processed in the UML editor currently. This diagram is not exported as a XMI package and its elements (**classes**, **associations**) are added as XML nodes directly at the same level as **package** elements.
4. VP XMI syntax even varies between Visual Paradigm versions. For instance in Visual Paradigm for UML 7.2, navigable association endpoints are organized in another way, than in Visual Paradigm for UML 8.2.

The conclusion is, that an UML2OWL transformation process always must be tailored to a specific vendor's UML solution. UML2OWL tool developers are required to

- specify vendor, product name and version of the UML editor they support
- clearly specify supported XMI versions and instructions, how to export that XMI version from the respective UML editor
- define, which UML elements they support and which they do not
- provide adequate reference models, test cases (automated approaches?) and documentation of test scenarios.

A concrete proposition can be found in chapter 3, in which a new open source UML2OWL transformation tool (including transformation of Visual Paradigm XMI 2.1) will be introduced. Firstly, however, we will point out Eclipse ATL transformation approach, which seems to be the most expandable and promising one.

⁵⁰ http://www.omg.org/gettingstarted/what_is_uml.htm, last visit 2011-07-01

2.7 Eclipse ATL Transformation Approach

As was stated above, the ODM UML2OWL implementation is based on ATL (ATLAS transformation language). The complete transformation scenario is illustrated below.

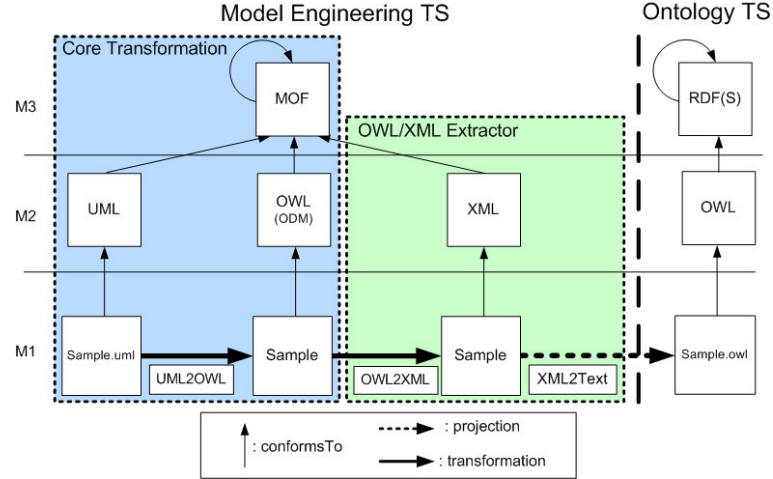


Figure 2.4: ATL UML2OWL transformation scenario copied from Eclipse ATL web page

Figure 2.4 illustrates transformation steps on different meta levels, as defined by OMG (compare [13]). ATL is used to transform UML 2.0 (XMI 2.1) input files into OWL metadata, based on ecore. Eclipse ecore is a metalanguage inspired by MOF 1.4 and is used to define platform independent models. [2, p. 8 ff] Once the XMI input file has been transformed into OWL ecore format, AM3 plugin⁵¹ serializes OWL resulting in valid OWL RDF/XML output.

```
rule UMLClass2OWLClass {
  from
    c : UML!uml::Class (
      c.ocIsTypeOf(UML!uml::Class) and
      not thisModule.sequenceOfUnionClass.includes(c)
    )
  to
    oc : OWL!OWLClass (
      subClassOf <- c.general,
      uriRef <- u,
      label <- label
    ),
    label : OWL!PlainLiteral ( lexicalForm <- c.name ),
    u : OWL!URIReference ( fragmentIdentifier <- 1, uri <- uri
  ),
  l : OWL!LocalName ( name <- c.name ),
```

⁵¹ <http://www.eclipse.org/gmt/am3/>, last visit 2011-06-11

```
uri : OWL!UniformResourceIdentifier ( name <- c.name )
}
```

Listing 1.1: ATL rule for UML class to OWL class transformation

The demonstrated rule is part of the UML2OWL ODM implementation and has been extracted from `UML2OWL.atl`. Each ATL rule consists of a **from** and **to** part. In listing 1.1, the rule selects all UML classes from the XMI input file, but only if the respective **class** element does not already exist in the module (therefore, a helper class called `sequenceOfUnionClass` is used). All selected UML classes are transformed into their OWL class counterpart within the **to** - phase, whereas labels, identifiers and generalizations are extracted from the UML model. `OWL!PlainLiteral`, `OWL!URIReference` and `OWL!UniformResourceIdentifier` are OWL data type declarations. Various ATL tutorials are available online.⁵²

The ATL approach is noble, though transformation rules are defined in a single, transparent file. It also supports syntax highlighting and debugging. Transformation rules are intuitive to interpret and may be easier to read than XSLT, because the rules do not contain any XML. Helper functions can be defined and give more freedom to the developer. ATL is easy adaptable.

Drawbacks are, that the Eclipse environment is hard to install (compare section 2.2). In particular, AM3 plugins have to be built directly from SVN repository and did not work in some cases. Furthermore, ecore/ATL training effort is high, because the rule based transformation language has to be learned. Developers have to familiarize with the EMF environment to gain a deeper understanding of EMF concepts, which can be time consuming.

2.8 Evaluation Conclusions

Referring to evaluation results, existing UML to OWL transformation tools provide insufficient functionality to be integrated into the CDL-Flex EKB framework. Central problems of transforming Visual Paradigm XMI 2.1 have been discussed in section 2.6. Although Eclipse ATL transformation approach in section 2.7 sounds promising, it has major drawbacks concerning interoperability of diverse EMF components and high training effort. Due to this, during this work a new UML to OWL transformation tool has been developed, which will be introduced in the next chapter.

⁵² http://wiki.eclipse.org/ATL/Tutorials_-_Create_a_simple_ATL_transformation , last visit 2011-06-12

3 UML to OWL Transformation Tool by Vienna University of Technology

UML to OWL transformation tool by Vienna University of Technology, consecutively called *umlTUowl*, is a new transformation approach, that has been developed during this thesis, because none of the evaluated tools transformed Visual Paradigm XMI into valid OWL ontologies successfully.

umlTUowl is optimized to transform UML class data models, as used by partners of CDL-Flex, into Protégé compatible OWL ontologies. Thus, it only supports the most common UML 2.0 components, but transforms Visual Paradigm XMI 2.1 into OWL DL⁵³ reliably. Further objectives during development have been, to make the *umlTUowl* available as an open source project, to support additional UML tools (e.g. MS Visio XMI) and to ease integration of new transformation scripts by third parties.

The tool also should be seen as a recommendation. It attempts to suggest, which information should be provided when publishing an UML transformation tool, based on the experiences collected during evaluation phase (see chapter 2).

3.1 Functionality and Obtainability

umlTUowl has been realized as a command line tool, based on Java 1.6 (or higher). Currently, it supports transformation of

- Visual Paradigm for UML, versions 7.2 and 8.2: XMI 2.1
- Microsoft Visio 2010: XMI 1.0
- ArgoUML 0.32.2 (available for free): XMI 2.1

The project is hosted at <http://uml2owl.sourceforge.net/> and is available under the BSD license. Java source code (Eclipse project) and executable jar file can be downloaded from there. Installation guides for users and developers are located at <http://sourceforge.net/apps/mediawiki/uml2owl/>. Developers are welcome to join the project.

Supported UML elements	Not supported UML elements
– classes – abstract classes – interfaces – generalization – multiple packages (diagrams) – attributes – visibility of attributes (meta model) – attributes with primitive data types including XML built-in data types – associations – navigable associations – multiplicity of associations	– multiplicity of attribute value – attribute values other than primitives – package elements inside a diagram – data constraints – class operations – association classes – n-air associations – overlapping/disjoint class annotation – roles (VP, ArgoUML) – XOR annotation – redefinition of derived attributes – subset annotation

⁵³ <http://www.w3.org/TR/owl-guide/>, last visit 2011-06-13

– aggregations – compositions – labeled endpoints (MS Visio) – comments / note elements	– ordering and uniqueness annotations for attributes – data type meta annotation – enumerations – stereotypes
--	--

Table 2: Comparison of supported and not supported UML elements

Visual Paradigm uses the term *package* for *diagram* synonymously. *umlTUowl* allows to define, if all packages should be merged into a single ontology, or if for each package an own ontology file should be created.

3.2 Software Architecture

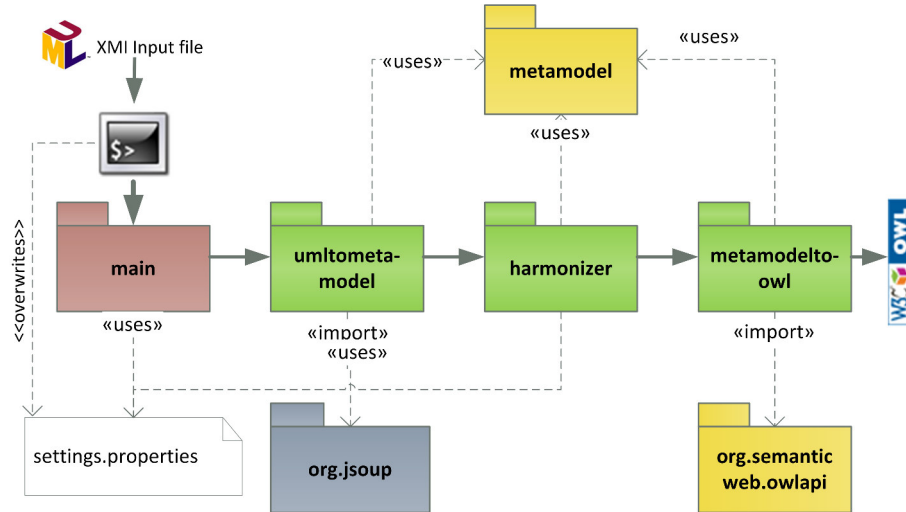
Figure 3.1: Workflow and software architecture (package structure) of *umlTUowl*

Figure 3.1 illustrates workflow and software architecture of *umlTUowl*. The tool can be executed as a **jar** file from console and accepts UML diagrams in the specified XMI format. Package **main** loads the XML file into the memory, considering file encoding. **settings.properties** is an internal properties file, that configures transformation, especially renaming process of UML element names. All settings can be overwritten, either by specifying an external **settings.properties** file or by defining Java input parameters. Package **uml2metamodel** contains the converters, that load the XMI file into a simplified meta model. Converters exist for each supported input format. For instance, Visual Paradigm XMI 2.1 models are transformed into Java meta models using `at.tuvienna.VisualParadigmConverterXMI2.java`. XMI elements are parsed

using jsoup⁵⁴, which provides CSS selector syntax.⁵⁵ Afterward, the **harmonizer** package ensures, that all classes, attributes and associations are labeled with a unique and Protégé compatible name. For instance, space characters are removed from class names, duplicate attributes are renamed and associations are labeled with a unique name. Non unique attribute names appear, when packages are merged into a single ontology or when the same attribute name is contained in different UML classes (e.g. age might exist for a class named "Student", but also for a class named "Building"). Associations often occur without an association label. Even when associations are named, the label might be useless, if the association is bidirectional navigable. Therefore, the **harmonizer** component offers naming strategies, which can be customized. For instance, "Building" contains "Rooms" results in two OWL object properties. The first one is named "buildingHasRoomAssociation", the second (inverse) object property is named "roomHasBuildingAssociation". If the association is bidirectional navigable, there is no way, to find out, if "Building" is contained in "Room" or vice versa. After the **harmonizer** renamed all elements, **metamodeltoowl** transformation is carried out. Therefore OWL API⁵⁶ by University of Manchester is used. The API guarantees, that the model is serialized into valid OWL2, resulting in a Protégé compatible OWL output file. The main packages are described in detail below.

Metamodel

Analogue to Eclipse MOF (as described in section 2.7) and Netbeans MDR (section 2.2), a simplified meta model has been defined, which is well suited for the *umlTUowl* transformation purpose. Meta data is held in-memory and the meta model is used as a buffer during harmonizing of element names (to gain name uniqueness). If one decides to create a new UML converter (e.g. for a not yet implemented UML editor tool), the meta model has to be populated. Basically, it consists of the following element types:

Element type	Usage
MetaModel	Container of all existing UML packages. Facilitates operations to manage them and provides methods to save global diagram name and descriptions.
MetaPackage	Represents a single UML diagram inside an UML model with an unique name. Provides methods to manage and find (UML) classes, either by their name or by XMI reference ID. Offers methods to retrieve classes with non-unique names, which may be useful, if various packages are merged into a single ontology.
MetaClass	Represents a UML class or an interface. A class can be defined as concrete or abstract. Each class contains generalization hierarchies, which mean, that it is linked to a set of super- and subclasses. Furthermore, each class can have attributes and associations. The Java class provides methods to select attributes and associations by their name.
MetaAttribute	UML attributes have a non-unique name, visibility and a data type. If no data type is given, "void" is assumed. Data type can be a built in XSD data types (compare ⁵⁷).

⁵⁴ <http://www.jsoup.org>, last visit 2011-06-13

⁵⁵ <http://www.w3.org/TR/2009/PR-css3-selectors-20091215/>, last visit 2011-06-13

⁵⁶ <http://owlapi.sourceforge.net/>, last visit 2011-06-13

MetaAssociation	Associations are either unidirectional or bidirectional. Some associations are aggregations or compositions. Each association can have a name, but has at least two endpoints (classes), it is connected with. Each association has its inverse association, except the association is navigable into only one direction.
MetaElement	All other meta elements are derived from this abstract class type. It contains handling of comments (also called "notes" in UML). Comments result in OWL annotations and can either be used to transform UML notes into ontologies or to add information, that otherwise will get lost during transformation. Examples: Information, that a class had been defined as abstract in UML or that an association had been renamed.

Table 3: umlTUowl meta model elements and their usage

Employment of a meta model has numerous benefits. On the one hand, decoupling of XMI parsing process and OWL serialization eases adoption of additional converters (e.g. for further UML model editors like Poseidon⁵⁸) and enables automated testing approaches. Even non-UML based models, like Entity-Relation diagrams can be converted into meta data and then are automatically transformed into OWL. On the other hand, a meta model can facilitate access to specific meta data elements, which implies that searching, manipulating and referencing of entities is much more comfortable than directly accessing XML. In the case of *umlTUowl*, the usage of a Java meta model provides a flexible architecture through object-oriented deriving, replacing and overriding mechanisms. For instance, the **harmonizer** component applies renaming strategies to class names, attributes and associations. Therefor strategies can easily be replaced or extended by developers goal-dependent, without touching the XMI conversion process (despite, some of the behavior can also be injected customizing **settings.properties**). Decoupling of input (parsing) and output (serialization) process furthermore has the advantage, that not only the parser might be replaced, but also the OWL serialization component. For instance, transformation into further ontology dialects like DAML⁵⁹, WSMML⁶⁰ or UML into Excel transformations may be implemented, just to name some possibilities.

Uml2metamodel

uml2metamodel is one of the tool's core components. It transforms XMI, that has been parsed into *jsoup*⁶¹ document format, into an internal meta model. Currently, the following converters exist:

1. **VisualParadigmConverterXMI2.java**: transformation of class diagrams in XMI 2.1 format, exported by Visual Paradigm for UML, versions 7.2, 8.2. All UML elements, as specified in table 2 are supported.

⁵⁷ <http://www.w3.org/TR/xmlschema-2/#built-in-datatypes>, last visit 2011-06-16

⁵⁸ <http://www.gentleware.com/>, last visit 2011-07-06

⁵⁹ www.daml.org, last visit 2011-07-07

⁶⁰ www.daml.org, last visit 2011-07-07

⁶¹ <http://www.jsoup.org>, last visit 2011-06-13

2. `MSVisio2010ConverterXMI.java`: transformation of class diagrams designed in Microsoft Visio 2011 and exported using Microsoft's *UML Background Addon*⁶² The supported format is XMI 1.0, which hardly has anything in common with XMI 2.1. Unlike Visual Paradigm, Visio does not export UML note elements. A workaround is, to add notes inside the documentation field for the particular UML class or association. For a proper XMI export, one has to ensure, that all UML elements are assigned to their specified package, using Visio's *model explorer*. Visio takes use of some language dependent terms inside the XMI file, which seems to be bad practice for automated approaches. For instance, in the German edition, compositions are labeled "zusammengesetzt" (engl.: composed). Therefore, only the English and the German version of Visio 2010 are supported currently.

MS Visio 2010 does not support labeling of associations directly, but labeling of its endpoints. Therefore, if only one endpoint is labeled, the converter interprets it as the only association name. If both endpoints are labeled and both directions are navigable, then each of the endpoint labels is used to name one of the resulting links. This seems to be a very convenient way to solve the problem of naming inverse object properties in OWL.

3. `ArgoUMLConverterXMI2.java`: To also support an open source UML modeling tool, ArgoUML (version 0.32.2)⁶³, a XMI 2.1 converter for ArgoUML is shipped within *umlTUowl*. While ArgoUML itself supports modeling of multiple packages, this information is lost during XMI export. Hence, elements are provided without package structure in the resulting XMI file. ArgoUML export also does not consider comments, hence they are not supported in *umlTUowl*.

Primitive data types get lost in ArgoUML during export. For instance, attribute data ranges, like `integer` or `string` disappear. A workaround solution is to define primitive data types inside the model, explicitly and use them instead of the original ones. Removal of unused associations and classes before export (using ArgoUML explorer) is essential to prevent errors.

umlTUowl utilizes `jsoup`, which, as already mentioned, provides CSS selector syntax, thus make it easy to parse XML. `jsoup` avoids circuitous code and make parsing of XML more transparent. Listing 1.2 provides some examples:

```
//Extraction of all UML class elements
//Each XML element contains an attribute named "xmi:type"
//and its value is "uml:Class"
Elements classes = xmi.select("[xmi:type=uml:Class]");

//Extract associations from XMI file: all elements with tags named
//"packagedElement", that have an attribute named "xmi:type", whoms
//value is "uml:Association"
Elements associations=
    xmi.select("packagedElement[xmi:type=uml:Association]");

//Check, if a class is surrounded by a package element and extract
//package name from XML attribute "name"
if (c.parent().tag().getName().equals("packagedElement"))
```

⁶² <http://social.technet.microsoft.com/Forums/en-CA/visiogeneral/thread/606f77b2-da70-4edf-a5a3-1ad55e8246f2>, last visit 2011-06-13

⁶³ <http://argouml.tigris.org/>, last visit 2011-06-13

```
&& c.parent().attr("xmi:type").equals("uml:Package"))
packageName = c.parent().attr("name");
```

Listing 1.2: Demonstration of jsoup selector syntax as used in umlTUowl

Listing 1.3 depicts, how XMI into meta model transformation is carried out in `VisualParadigmConverterXMI2`. The pseudo code is very simplified, but a good starting point for everyone, who intends to code his own converter.

```
//predefined
xmi      := Jsoup.parse(xmlFile)
metaModel := create a new MetaModel
//step 1)
xmiClasses := extract all elements with attribute "xmi:type"
              equal to "uml:Class" or "uml:Interface"
Iterate over xmiClasses as xmiClass {
  metaClass := create a new meta class
  extract abstract flag, name and internal XMI id
  from xmiClass and put it into metaClass
//step 2)
attrs := extract all attribute tags from xmiClass
Iterate over attrs as a {
  extract name, visibility and rangeID from a
  extract datatype from XMI using rangeID
  create a new MetaAttribute, fill it with name, visibility
  and datatype and add it to metaClass
}
//step 3)
packageName := extract from xmi (either its xmiClass parent
                      element or all uml:Diagram elements have to
                      be traversed).
metaPackage := lookup for packageName in metamodel and create a
                  new package if it does not exist yet.
add metaClass to metaPackage.
}
//step 4) all classes have been loaded into metamodel. Thus,
//generalization and adding of associations can be carried out
iterate over xmiClasses and extract tags named "generalization" {
  extract class IDs of found superclasses
  add found superclasses to corresp. subclasses
  in metamodel and reverse
}
//step 5) handle associations (VP 8.2 version - very simplified)
associations := extract all elements with tagname "packagedElement"
              and xmi:type equals "uml:Association";
Iterate over associations as ass {
  ep := extract endpoints from ass (exactly 2).
  ass[1] := create a new MetaAssociation between ep[1] and ep[2].
  ass[2] := create a new MetaAssociation between ep[2] and ep[1].
  extract association name and add it to both, ass1 and ass2.
  for all ep using counter i {
    if ep[i].attribute("aggregation") is not empty then
      mark ass[i] either as "composition" or "aggregation"
    end-if
  }
  if all ep contain attribute "isnavigable" then
    ass1.setInverse(ass2);
    ass2.setInverse(ass1);
    add both associations to corresponding meta classes
  else
    only add association that contains a navigable endpoint to
```

```

        corresponding meta class
    end-if
}
//step 6
extract comments and associate them to metaClasses
and metaAssociations

```

Listing 1.3: Abstraction of Visual Paradigm XMI 2.1 meta model converter. Extraction steps are accented.

Harmonizer

The harmonizer component ensures uniqueness of element names and prepares them for the OWL export. Depending on `settings.properties`, a strategy is selected and all entity names in the meta model are unified, so that each name only occurs once. Furthermore, attribute and association names are converted into common OWL style.

UML element	Example	D	Harmonizing result	Relevant settings
class	Austrian Student	X	Austrian.Student	Concrete-prefix= Concrete-postfix=
class	Two classes named <i>Student</i> exist within two different packages.	X	Student Package2 .Student	merge-packages=true merge-disable-fixing=true merge-class-prefix={package}_
class	Two classes named <i>Student</i> exist within two different packages.		Package1 .Student Package2 .Student	merge-packages=true merge-disable-fixing=false merge-class-prefix={package}_
class	Two classes named <i>Student</i> exist within two different packages.		Student 1 in out. Package1 .owl Student 2 in out. Package1 .owl	merge-packages=false
abstract class	{ <i>abstract</i> } <i>Student</i>	X	Abstract .Student	abstract-prefix=Abstract_ abstract-postfix=
interface class	<i>Student</i>	X	Interface .Student	interface-prefix=Interface_ interface-postfix=
attribute	Class <i>Student</i> has attribute <i>age</i> (integer).	X	hasAge	data-property-prefix=has
boolean attribute	Class <i>Student</i> has attribute <i>male</i> : Boolean.	X	isMale	data-property-prefix-boolean=is
attribute	Two classes, namely <i>Student</i> and <i>Building</i> exist within the same package. Both contain attribute <i>name</i> .	X	Building: hasName Student: has Package1 .StudentName	merge-packages=true data-property-prefix=has merge-attribute-strategy=duplicates merge-attribute-prefix=package-class_
attribute	Two classes, namely <i>Student</i> and <i>Building</i> exist within the same package. Both contain attribute <i>name</i> .		Building: hasName Student: has Student Name	merge-packages=true data-property-prefix=has merge-attribute-strategy=duplicates merge-attribute-prefix={class}
attribute	Two classes, namely <i>Student</i> and <i>Building</i> exist within the same package. Both contain attribute <i>name</i> .		Student: has Student Name Building: has Building Name	merge-packages=true data-property-prefix=has merge-attribute-strategy=all merge-attribute-prefix={class}
attribute	<i>Person</i> is an abstract super-class of <i>Student</i> . Both have the attribute <i>name</i> (<i>Student</i> overrides <i>Person</i> 's name).	S	Abstract .Person: hasName Person: has Student Name Reason: Abstract classes and interfaces are treated first.	merge-packages=false data-property-prefix=has default-attribute-strategy=duplicates default-attribute-prefix={class} default-attribute-postfix=
association	Unidirectional association named <i>based on</i> between <i>Tree</i> and <i>Trunk</i> .	X	hasTreeTrunkRelation hasTrunkTreeRelation (inverse). Comment "Relation originally named 'based on' will be added.	relation-strategy-relation=Relation relation-strategie-1=has{from}{to}{dependingRelation}

UML element	Example	D	Harmonizing result	Relevant settings
composition	Each <i>Tire</i> belongs to exactly 1 <i>Car</i> at one time.	X	hasCarTireComposition isTireOfCar (inverse - part of)	relation-strategie-1=has{from}{to}{dependingRelation} relation-strategie-composition-part-1=is{from}Of{to} relation-strategie-composition=Composition
aggregation	<i>University</i> has <i>Researchers</i> .		hasUniversityResearcher IsResearcherOfUniversity (inverse)	relation-strategie-1=has{from}{to}{dependingRelation} relation-strategie-aggregation-part-1=is{from}Of{to} relation-strategie-aggregation=
association	Three associations: A <i>Tank</i> contains <i>Temperature Sensors</i> , <i>Level Sensors</i> and <i>Heaters</i> (all class names). All associations are labeled with <i>contains</i> .	S	All associations are still labeled with <i>contains</i> . In OWL, one object property named contains with domain Tank and range Temperature.Sensor or Level.Sensor or Heater will be created.	allow-multiple-labeled-names=true relation-strategie-1={name} relation-strategie-2=has{from}{to}{dependingRelation}

Table 4: Examples of harmonizing techniques, depending on settings

Table 4 illustrates the most common harmonizing techniques and how they are applied. Column "D" contains "X", if harmonizing has been carried out using default settings, "S" if it is a special case and hence required configuration changes, and is empty, if configuration parameters must be changed, to achieve the illustrated result, but is not a special case yet.

Two strategies have been implemented to handle packages: Depending on configuration settings in `settings.properties`, either all meta packages are merged into a single meta model within the harmonizing phase, or all packages remain and are serialized into separated ontologies, subsequently. If all classes are merged into a single ontology, then more harmonizing effort is required, because it can happen, that semantically nonequivalent classes with same class name exist in different packages. These classes have to be prefixed with their package name.

Duplicate attributes and association names frequently occur even within a package. Thus, attribute names are always renamed (pre- or suffixing class and/or package name), if they occur more than once, although different strategies exist therefore. Association labels are useless, except the association is navigable. Hence, all associations are renamed, depending on the chosen strategy (`settings.properties`). For instance, as illustrated in table 4, aggregations and compositions can be renamed using supplementary patterns.

Another point concerning attribute transformation is, that it is necessary to traverse attributes of abstract classes and interfaces first and afterward continue with attributes of concrete classes. The reason is, that abstract classes are more general, so they should be altered as less as possible. If an attribute (e.g. age) exists for an abstract class and three concrete classes (even they are no subclasses of the abstract class), it is desirable that the abstract class keeps its original attribute name, while the concrete class's attributes may be converted into, lets say studentAge, buildingAge, animalAge, without any further problems.

Metamodeltoowl

Finally, OWL API by the University of Manchester has been used, to serialize *umlTUowl*'s meta model into valid OWL2 DL. OWL API project, including documentation and examples, is located at <http://owlapi.sourceforge.net/>. The API's design is outlined in [9, p. 2 ff].

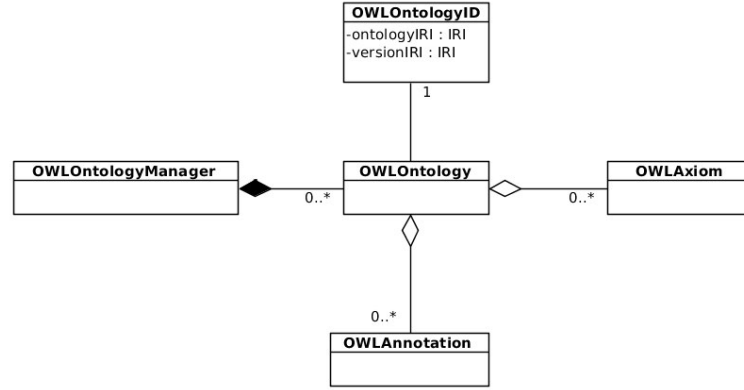


Figure 3.2: Ontology management in OWL API, as modeled in [9, p. 3]

Basically, the API (as illustrated in figure 3.2), is composed of the following components:

- **OWLManager**: facilitates management of ontologies. Instantiates OWL ontologies and is used to add new axioms to a specify ontology.
- **IRI**: Document resource identifier handling. All entities inside the ontology must be prefixed with a valid identifier, e.g. <http://www.acin.tuwien.ac.at/model.owl>.
- **OWLOntology**: single ontology, managed by OWLManager.
- **OWLDataFactory**: OWLManager references to a factory, that is used to create new OWL resources, e.g. OWL classes, annotations, and axioms.
- **Axiom**: All relations and restrictions are added as OWL axioms to the ontology. Examples: `OWLSubClassOfAxiom`, `OWLDeclarationAxiom`, `OWLDataPropertyDomainAxiom`, `OWLInverseObjectPropertiesAxiom`, `OWLObjectPropertyRangeAxiom`, `OWLObjectPropertyDomainAxiom`.
- **Expression**: Expressions are used to restrict cardinality. They can be combined, using set operations. Example: `getOWLObjectUnionOf` is used to set range and domain of object properties if an *umlTUowl* user wants to combine multiple, equally named associations to a single OWL object property in the way **Tank contains (Water OR Sensor OR Pump)**.

Figure 3.3 illustrates the most common mapping mechanisms of UML elements and OWL resources. For instance, an abstract UML class is transformed into an `OWL:Class` entity and an annotation, that the resulting OWL resource has been derived from an abstract UML class, is added. Depending on the chosen transformation strategy (`settings.properties`), either all UML packages are merged into a single file, or each

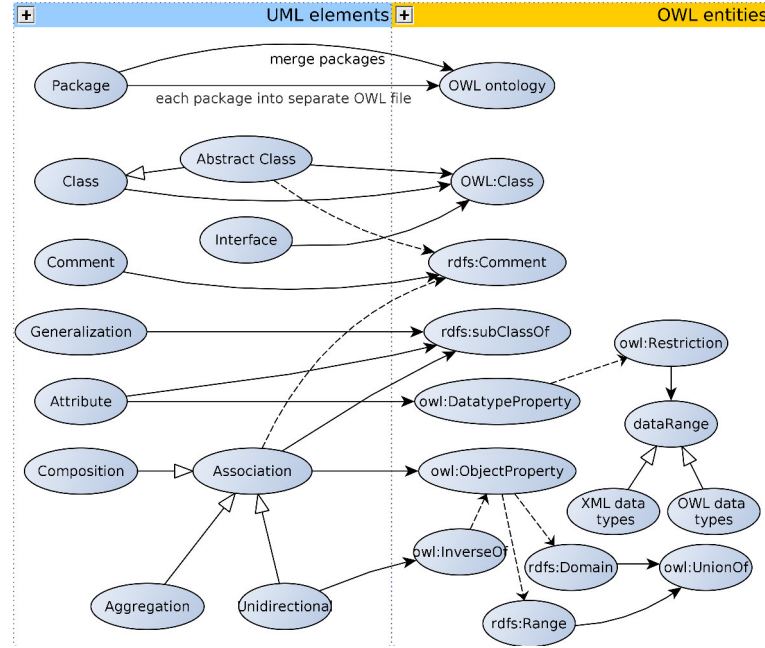


Figure 3.3: Mapping of UML elements and OWL entities

package is separated into a single OWL ontology. The attributes are transformed into data properties. XML built in data types⁶⁴ are supported as well as OWL built in data types (e.g. `anyType`).

UML associations result in OWL object properties, which are restricted by domain and range through their corresponding OWL classes (formally UML classes or interfaces). Equally named associations are supported. Thus, domain/range expressions contain concatenated OWL entities, separated by **OR**. Example: UML associations **Professor holds Lecture** and **Professor holds Seminar** will be transformed into an object property **holds** with domain **Professor** and range **Lecture OR Seminar**. If an association is navigable in both directions, object properties will be linked as *Inverse properties* one another. All object properties are added as subclass axioms to their related OWL classes, concerning cardinality. Assuming, that a Professor can hold between 0 and 500 lectures, the example above would result in two subclass axioms: **holds min 0 Lectures and max 500 Lectures** and **holds some Seminar**.

All OWL files that have been gained from transformation of the used UML reference models (see section 3.4 and appendix A for more details), have been validated in Protégé and were reasoned with FaCT++⁶⁵. The resulting OWL format has been approved, using <http://owl.cs.manchester.ac.uk/validator> (compare [9, p. 5,6]), and is valid OWL2 DL. A Protégé screenshot of exploring a transformed UML class model can be found in appendix B.

⁶⁴ <http://www.w3.org/TR/xmlschema-2/#built-in-data-types>, last visit 2011-06-14

⁶⁵ <http://owl.man.ac.uk/factplusplus/>, last visit 2011-06-14

3.3 Time Complexity

This section treats time complexity of *umlTUowl*. The results are discussed afterward.

We define:

#N... number of XML elements (depending on XML serialization)
#C... number of classes
#I... number of interfaces
#G... number of generalizations
#P... number of packages
#Attr... number of attributes
#Assoc... number of associations
#Com... number of comments
#ComC... number of connections between comments and classes/associations
#Strategies... number of renaming strategies as def. in settings

According to listing 1.3, time complexity for UML to meta model transformation is calculated as follows:

$$\begin{aligned} T_1 = & O(\#N) + [O(\#C + \#I) + O(\#P)] + O(\#G) * O(\#C + \#I) \\ & + O(\#Attr) * O(\#N) + O(\#Assoc) * O(\#C/\#P) \\ & + O(\#Com) * O(\#ComC) * [O(\#C + \#I) + O(\#Assoc)] \end{aligned} \quad (1)$$

Furthermore

$$\begin{aligned} O(\#C + \#I) &< O(\#N), \\ O(\#C + \#I) + O(\#P) &< O(\#N), \\ O(\#C + \#I) + O(\#Assoc) &< O(\#N), \\ O(\#C/\#P) &< O(\#N), \\ O(\#N) &< O(\#Attr) * O(\#N). \end{aligned} \quad (2)$$

(1) and (2) lead to

$$\begin{aligned} T_1 = & O(\#G) * O(\#N) + O(\#Attr) * O(\#N) + O(\#Assoc) * O(\#N) \\ & + O(\#Com) * O(\#ComC) * O(\#N) \\ = & O(\#N) * [O(\#G) + O(\#Assoc) + O(\#Attr) + (\#Com) * O(\#ComC)] \end{aligned} \quad (3)$$

UML to meta model transformation is the most time-consuming part. Harmonizing of meta model elements is limited by (4).

$$T_2 = O(\#Attr) * O(1) + O(\#Assoc) * [O(\#Strategy) + O(1)] \quad (4)$$

Additional effort (5) is necessary, if merging strategy is used.

$$T_{2M} = (\#C + \#I) * O(1) \quad (5)$$

$O(1)$ occurs in formula (4) and (5). The reason is, that a `HashSet` is engaged to avoid duplicates. Therefore, `contains` - operation of `hasSet` is used, whose time complexity

is constant.⁶⁶ The number of harmonizing strategies is constant and in practice below five. Time complexity for OWL serialization is (simplified) limited by (6).

$$T_3 = O(\#C + \#I + \#Attr + \#Assoc + \#Com) \quad (6)$$

The time complexity of the entire transformation process is formulated in (7).

$$T_{ENTIRE} = T_1 + T_2 + T_3 \quad (7)$$

Because

$$\begin{aligned} T_2 &= O(\#Attr) * O(1) + O(\#Assoc) * [O(\#Strategy) + O(1)] \\ &= O(\#Attr + \#Assoc) < O(\#N) \end{aligned} \quad (8)$$

and

$$T_3 = O(\#C + \#I + \#Attr + \#Assoc + \#Com) < O(\#N) \quad (9)$$

, the entire time complexity is limited only by the UML to meta model transformation component (`uml2metamodel`), as formulated in (3). The result shows, that execution time will be determined by the number of XML elements inside the XMI file (dependent on number of diagram elements and XML serialization) and the number of connections between UML classes (plus interfaces and comments), namely generalizations, associations, comment connections and attributes.

Comparison with Xu et al.'s solution [14, p. 131] (see 2.2) shows, that time complexity is a bit higher than $O(\#N)$. The main reason is, that the DOM structure of Visual Paradigm's XMI 2.1 in some cases has to be traversed, e.g. to map between attributes and their data range, or to link comments to their corresponding entity. The algorithm may be optimized (e.g. extracting data types into an array before processing attributes), but the current algorithm is broadly satisfactory.

Comparing the DOM based parsing approach with a SAX based solution, one can conclude, that SAX cannot unfold its merits. Firstly, the used meta model approach decouples XMI parsing from OWL serialization phase and attempts to ease adaption of new converters. Therefore, the model's meta data has to be stored in-memory, making SAX's storage-size advantage dispensable. Secondly, the XMI file contains loads of cross references between elements, hence a SAX based approach probably may result in unnecessarily complicated code. Furthermore, the SAX based approach has to walk over the whole XML document's elements at least once, so also irrelevant elements are passed through, even the sections of the XMI document, that only contain layout and arrangement information of the UML elements within Visual Paradigm.⁶⁷ Thus, the DOM based approach not only cuts better regarding clarity and maintainability, but also is in no way inferior concerning time- and space complexity.

Tests of the tool showed, that execution time for UML diagrams that contain between 30 and 50 classes or interfaces, is very low. The transformation process terminated within 1 second, even on a Intel Core Duo Processor with 1.6GHz and 1024 MB RAM.

⁶⁶ <http://download.oracle.com/javase/tutorial/collections/implementations/set.html>, last visit 2011-06-14

⁶⁷ <http://www.w3.org/DOM/faq.html#SAXandDOM>, last visit 2011-07-04

3.4 Automated Testing

umlTUowl uses a very convenient testing approach, that is based on jUnit ⁶⁸. This approach has not been found in any existing UML to OWL transformation literature so far, hence it will be introduced.

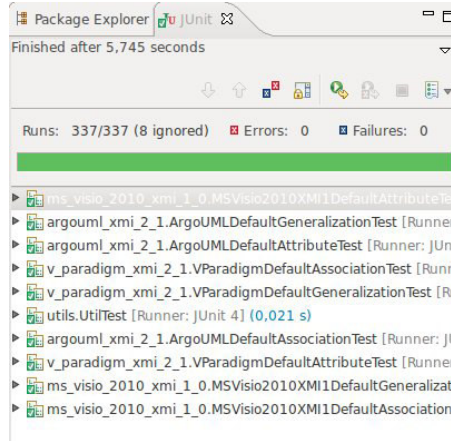


Figure 3.4: Screenshot of passed jUnit tests (Eclipse)

umlTUowl provides a wide variety of jUnit test cases, combined with a UML reference model. The model consists of three packages, which cover the most common UML class diagram constructs. In particular, UML classes, interfaces, attributes (including data types), inheritance structures (generalization), associations (bi- and unidirectional, compositions, aggregations) concerning cardinality and comments (also called notes) occur. The model's packages are illustrated in appendix A.

To support a new UML modeling editor's XMI format, one just has to write a new converter, based on the existing examples (e.g. `VisualParadigmConverterXMI2.java`) and add it to the `uml2metamodel` package inside of *umlTUowl*. Afterward, the provided UML reference models have to be recreated in the UML editor of choice. The *umlTUowl* Java project already provides an adequate file structure (sub-directories of `test/reference-models/{name-of-the-new-converter}`), in which the new modeled examples can be placed:

- `models/`: images of the reference model's packages, e.g. in png-format.
- `screenshots/`: can contain additional screenshots, to demonstrate the UML editor modeling environment and make XMI export process reproducible.
- `working/`: must contain the UML editor's project files.
- `input-file.xmi`: the resulting XMI file of the modeled diagram.
- source package `name-of-the-new-converter/`: copy an existing reference model test package, e.g. `ms_visio_2010_xmi_1_0`, rename jUnit test classes and adjust the relative file path's in the code to support your model.

⁶⁸ <http://www.junit.org>

All reference test cases will be derived from `DefaultAssociationTest.java`, `DefaultAttributeTest.java` and `DefaultGeneralizationTest.java` automatically. If a UML modeling tool does not support a specific XMI element (e.g. comments are not supported in ArgoUML), the specific test case can be removed, using JUnit's `@Ignore` annotation, although removal of test cases should be avoided. Note, that the test cases only cover UML to meta model transformation, because harmonizing and transformation of meta model into OWL are standardized and hence are not relevant when adding new XMI converters. Developers just have to code the XMI converter, parsing XML elements into the meta model using `JSoup` and don't have to care about the rest of transformation process!

The automated testing approach on the one hand guarantees, that a standard level of transformation tool quality can be reached, and on the other hand reduces development costs. Additional reference models can be placed to support additional features for a specific UML editor's XMI file format. Adaption of not yet implemented transformation features can be carried out across all existing converters, hence help to gain high-quality converters. Even UML-related models, such as MS Visio ER (Entity Relationship) diagram, can be implemented in adequate time.

3.5 Use Case: Transformation of CDL-Flex Tank Model

To certify, that *umlTUowl* can be integrated into the CDL-Flex EKB framework and to dig into a use case, the tool has been validated against an existing, domain specific tank model, that had been used in a project by CDL-Flex, previously. Both, Visual Paradigm class data diagram and OWL ontology, that had been created in manual work by CDL-Flex, are available.

The UML diagram models a piped tank, that is connected to several control elements, which ensure that the tank provides enough water (or oil) and the content is heated, dependent on the measures of different sensors. The existing, manually created on-

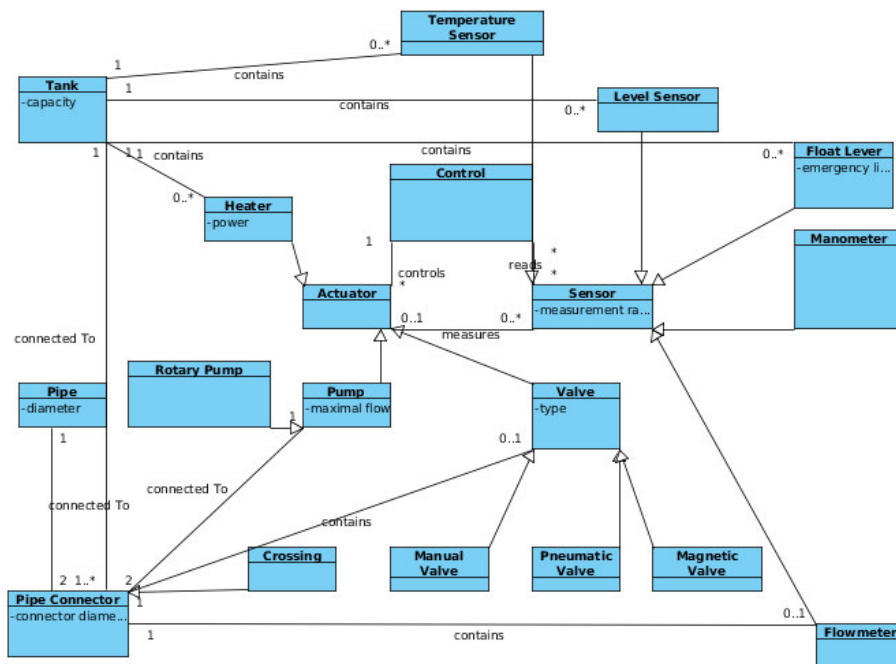


Figure 3.5: Tank modeled in Visual Paradigm for UML 8.2

tology and the *umlTUowl* transformation result have been compared in Protégé. The results are outlined next.

Class Transformation and Disjointness

The tool transformed all 19 classes and considered all hierarchy levels correctly. For instance, actuator is the superclass of heater, pump and valves, while valve itself has been subdivided into magnetic, manual and pneumatic classes. Unlike the manually created ontology, disjointness of classes is not defined in the *umlTUowl* created on-

tology. By default, UML classes are defined as overlapping ⁶⁹, hence *umlTUowl* does not assume disjointness. The ontology engineer has to explicitly define disjoint classes, using Protégé.

Equivalent Classes

Of course the automated *umlTUowl* transformation approach cannot distinguish equivalent and defined classes. While the CDL-Flex ontology engineer defined some equivalent classes in OWL, the tool created defined classes with subclass axioms, instead. Thus, object properties appeared in Protégé's superclass section.

Attributes

Because for none of the attributes a data type had been specified in the UML diagram, the tool assigned data range **anyType**. The ontology engineer has to add missing ranges to all data properties manually, e.g. set the data range of **hasCapacity** (Domain = Tank) to **float**.

Associations

In the first test run, the tool created 24 object properties, while the CDL-Flex ontology contained 7 object properties. The reason was, that the transformation tool created an object property and its inverse for each UML association. For instance, one sensor measures an actuator. The resulting object properties were **hasActuatorSensorRelation** and **hasSensorActuatorRelation**, which is the inverse object property (compare table 4). For avoidance of inverse relations, either the object properties have to be removed in Protégé, or the UML designer has to define unidirectional associations (add navigation arrows).

After the UML diagram had been reworked, 24/2=12 object properties remained. The CDL-Flex ontology still contained 7 object properties. The reason is, that in the UML model 6 associations have been labeled with **contains** and two are named **connectedTo**. The UML tool can not estimate, if all associations with same label are semantically equal and therefore, creates a new object property for each association.

umlTUowl had been extended, to support transformation of equally named associations into a single object property (compare 4).

One can modify **settings.properties** as followed:

```
whitespace-replacement-char=empty
allow-multiple-labelled-names=true
relation-strategy-1=name
```

The first line specifies, that blanks will be removed from UML element names, e.g. **contains To** will be converted into **containsTo** instead of **contains_To**). The second and the third line define, that equally named associations are allowed and that their name should be used to label OWL object properties.

After specifying the modified **settings.properties** as an input parameter in *umlTUowl*, the number of object properties in the result was reduced to a number of 6

⁶⁹ <http://www.essentialstrategies.com/publications/modeling/uml.htm>, last visit 2011-06-15

(the manually created ontology contained an additional property, that is not modeled in the UML file). The result has been equivalent to the manually created ontology. Domains, ranges and cardinality of object properties have been transformed properly.

3.6 Future Work

umlTUowl provides both, a simple user console environment and a mature Java API, which are ready to be integrated into the CDL Flex's EKB framework. It already supports most of the common UML elements, even though rarer elements are not yet supported. One issue concerns handling of disjointness within OWL. Although UML supports notation of disjoint classes, they are hardly used in practice. Obviously, an implementation within *umlTUowl* might not be a silver bullet. From the CDL Flex's point of view, it seems the best to either review the entities in Protégé and mark disjoint classes manually or to add an input parameter, so that *umlTUowl* assumes, that all classes in the UML model are disjoint. Latter might be relevant, because in practice engineers often assume disjointness of UML classes but do not denote this property explicitly. Other relevant extensions concern support of attribute multiplicity, detection of explicitly defined attribute data types and recognition of additional UML constructs, such as enumerations, n-ary associations and association classes. *umlTUowl* has been designed for logical UML class data diagrams originally, but transformation of constraints, such as **primary key** or **not null** does not seem to be expedient because constraints are assigned to data properties and not to entities directly. Instead, introduction of additional UML notations for specifying rules (e.g. SWRL), as already implemented in Two Use Toolkit, might be worth thinking about. A helpful extension, that has been introduced by [8] is the support of UML object diagrams to enable the transformation of class instances into OWL individuals. Therefore *umlTUowl*'s meta model has to be extended slightly.

Although back-transformation of OWL into UML might be useful to apply ontology changes to the UML diagram, the architecture of the tool therefor is unsuitable currently and would need refactoring. It is also unlikely, that this approach would work sufficiently in practice, because UML model editors often have problems when importing XMI files and OWL entities cannot be aligned and positioned automatically.

The UML to OWL transformation process also might be used for code generation. For instance Java classes with Hibernate⁷⁰ annotations could be created from the internal meta model, using the UML model as the initial point of model driven architecture (MDA) and knowledge externalization.

As mentioned, *umlTUowl* is hosted as an open source project. Thus, a desirable goal might be to support a broader range of existing UML and Entity- Relationship modeling tools, using *umlTUowl*'s automated test approach. The next candidates therefor are Eclipse Papyrus and Poseidon UML.

⁷⁰ www.hibernate.org/, last visit 2011-07-04

4 Conclusion

As regards evaluation, none of the existing transformation tools satisfies requirements of transforming Visual Paradigm UML 2.0 into OWL. Evaluation showed, that most of the purebred transformation tools only understand vendor (or even UML-editor) specific XMI formats (e.g. Poseidon CE 4.1) and fail in processing other formats. Furthermore, most of the transformation tools are not up-to-date. Although Eclipse's ATL framework, which has been introduced in chapter 2.7, offers a transparent transformation approach, it is - in combination with OWL - still half-baked.

umlTUowl, which has been developed during this work and has been introduced in chapter 3, is tailor-made for the CDL-Flex. The tool supports *Visual Paradigm for UML 8.2*, *MS Visio 2010* and *ArgoUML 0.32.2*. It transforms UML class data models into valid OWL 2 DL. *umlTUowl* has been realized as an open source project and is available at <http://uml2owl.sourceforge.net>. It rests on a well-thought-out Java design, and provokes developers to participate. An automated test approach helps to integrate new XML/XMI converters into *umlTUowl* rapidly.

Although *umlTUowl* supports all of the most common UML elements, some rarer elements are not yet supported (see table 2). Concerning ontologies, the support of disjoint UML class annotations, might be useful. Even though it is not required by the CDL-Flex currently, transformation of attribute data types other than primitives, multiplicity of data properties and support of constraints could make sense longer-term.

Depending on future trends, hybrid modeling tools might substitute traditional UML editors. TwoUse toolkit, which has been introduced in chapter 2.3, seems to be promising, since it also supports modeling of rules and is worth a look.

References

1. BACLAWSKI, K., KOKAR, M. M., KOGUT, P. A., HART, L., SMITH, J., LETKOWSKI, J., AND EMERY, P. Extending the Unified Modeling Language for Ontology Development. *Software and Systems Modeling* 1, 2 (2002), 142–156.
2. BACVANSKI, V., AND GRAFF, P. Mastering eclipse modeling framework. *EclipseCon. Eclipse Foundation* (2005).
3. BROCKMANS, S., HAASE, P., HITZLER, P., AND STUDER, R. A metamodel and uml profile for rule-extended owl dl ontologies. *The Semantic Web: Research and Applications* (2006), 303–316.
4. BROCKMANS, S., VOLZ, R., EBERHART, A., AND LÖFFLER, P. Visual Modeling of OWL DL Ontologies Using UML. In *The Semantic Web – ISWC 2004*, S. A. McIlraith, D. Plexousakis, and F. van Harmelen, Eds., vol. 3298 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2004, ch. 15, pp. 198–213.
5. CRANFIELD, S. Uml and the semantic web. *Proceedings of the International Semantic Web Working Symposium* (2001).
6. DEVOS, W. Ontology and the Age of Integration in the Electric Power Industry. *Semantic Technology Conference* (2006), 38 ff.
7. FALKOVYCH, K., SABOU, M., AND STUCKENSCHMIDT, H. Uml for the semantic web: Transformation-based approaches. *Knowledge Transformation for the Semantic Web 95* (2003), 92–107.
8. GASEVIC, D., DJURIC, D., DEVEDZIC, V., AND DAMJANOVI, V. Converting UML to OWL ontologies. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (New York, NY, USA, 2004), WWW Alt. '04, ACM, pp. 488–489.
9. HORRIDGE, M., AND BECHHOFFER, S. The OWL API: a Java API for working with OWL 2 ontologies. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2009)*, *CEUR Workshop Proceedings*, Chantilly, VA, United States, October, Citeseer.
10. KIKO, A. A Detailed Comparison of UML and OWL. *REIHE INFORMATIK TR-2008-004* (2008).
11. LEINHOS, S. OWL Ontologieextraktion und -modellierung auf der Basis von UML Klassendiagrammen. Master’s thesis, University of the Federal Armed Forces Munich, 2006.
12. MATULA, M. Netbeans Metadata Repository (Whitepaper), 2003.
13. NYTUN, J., AND PRINNZ, A. Metalevel representation and philosophical ontology. In *Workshop on Philosophy, Ontology, and Information Systems (held as part of the Eighteenth European Conference on Object Oriented Programming, ECOOP-04)*, Oslo (2004), p. 2.
14. XU, Z., NI, Y., LIN, L., AND GU, H. A Semantics-Preserving Approach for Extracting OWL Ontologies from UML Class Diagrams. *Database Theory and Application* (2009), 122–136.

List of Figures

2.1 UML/OWL modeling tools classified by relevance and technology	2
2.2 Demonstration of TwoUse toolkit model elements	8
2.3 Result of evaluation. Rating of tools in categories between 0 and 10.....	11
2.4 ATL UML2OWL transformation scenario copied from Eclipse ATL web page	14
3.1 Workflow and software architecture (package structure) of <i>umlTUowl</i>	17
3.2 Ontology management in OWL API, as modeled in [9, p. 3]	24
3.3 Mapping of UML elements and OWL entities	25
3.4 Screenshot of passed junit tests (Eclipse)	28
3.5 Tank modeled in Visual Paradigm for UML 8.2	30
A1 Reference model for evaluation of attributes	37
A2 Reference model for evaluation of inheritance structures	37
A3 Reference model for evaluation of associations	38
B4 Example of transformed UML class model exploring in Protégé	39

Declaration of Authorship

I hereby confirm that I have authored this thesis independently and without use of other than the indicated resources. All passages, which are literally or in general manner taken out of publications or other sources, are marked as such.

Andreas Grünwald

Appendix

A Reference Models

The following section lists the reference models, that have been used for automated testing during umlTUowl development. Those models have been designed in each UML model editor, that is supported (Visual Paradigm, MS Visio 2010, ArgoUML) currently. However, the following models have been created in Visual Paradigm UML 8.2 and were used for XMI 2.1 transformation evaluation.

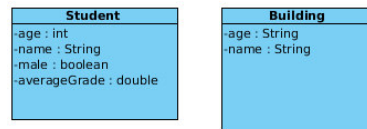


Figure A1: Reference model for evaluation of attributes

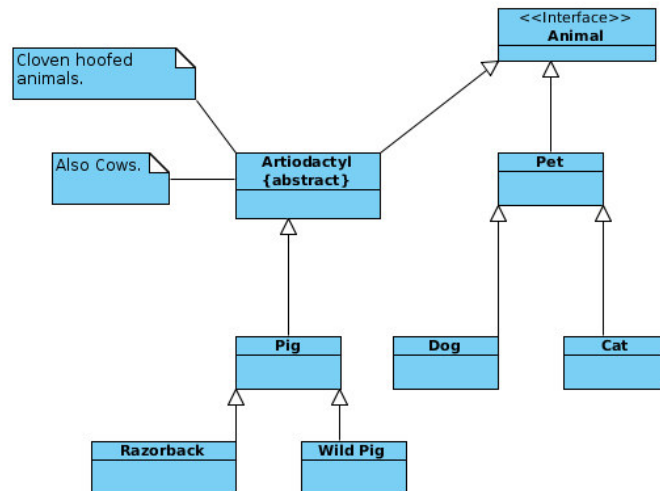


Figure A2: Reference model for evaluation of inheritance structures

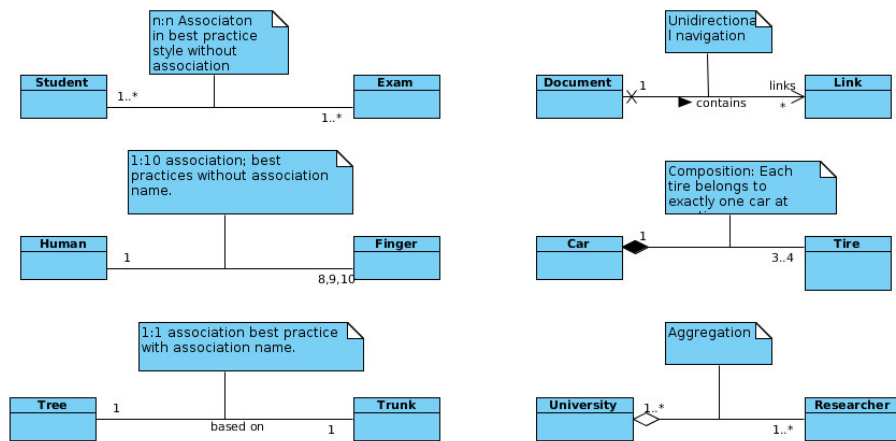


Figure A3: Reference model for evaluation of associations

B Protégé Screenshot

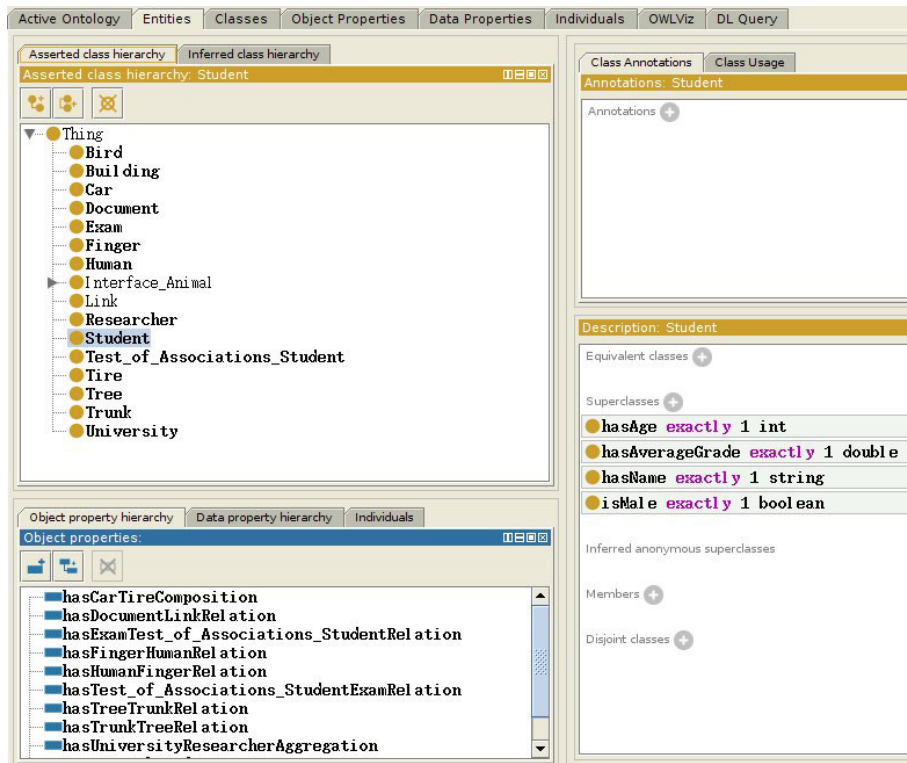


Figure B4: Example of transformed UML class model exploring in Protégé