RC23795 (W0511-088) November 11, 2005
Computer Science

IBM Research Report

An MDA-Based System for Ontology Engineering

Yue Pan, Guotong Xie, Li Ma, Yang Yang, ZhaoMing Qiu
IBM Research Division
China Research Laboratory
HaoHai Building, No. 7, 5th Street
ShangDi, Beijing 100085
China

Juhnyoung Lee
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

An MDA-Based System for Ontology Engineering

Yue Pan, Guotong Xie, Li Ma,
Yang Yang, ZhaoMing Qiu

IBM China Research Lab
Beijing, China
{panyue, xieguot, malli, yangyy,
qiuzhaom}@cn.ibm.com

Juhnyoung Lee

IBM Watson Research Center
Hawthorne, New York
jyl@us.ibm.com

Abstract

W3C's Semantic Web provides a common framework that allows data to be shared and reused across application and enterprise. As the Semantic Web shapes the future of the Web, it becomes more and more important in software engineering and enterprise application development. While existing *ontology engineering* tools provide a stack of ontology management support and are used successfully in certain domains, there still remains a gap between the ontology engineering tools and the traditional software engineering. For several decades, software engineering has been established on different modeling languages and methodologies such as Unified Modeling Language (UML). The differences in modeling languages and methodologies cause difficulties in enterprise application development involving the Semantic Web technologies. The existing ontology engineering tools provide only an ad hoc approach to bridging this gap with limited functionality and performance. The primary objective of our work is to bridge this gap between two different, but complementary engineering disciplines with a systematic approach. Our approach leverages *Model-Driven Architecture* (MDA) and *Ontology Definition Metamodel* (ODM), which enable model transformation. This approach allows seamlessly supporting existing models in UML and other languages in Semantic Web-based software development. In addition, it allows exploiting the availability and features of UML tools for creation of vocabularies and ontologies. Furthermore, MDA enables code generation and facilitates software tool development. This paper presents an MDA-based system for ontology engineering. In addition, it presents the entire stack of individual components of the developed ontology engineering tool.

Keywords: Semantic Web, ontology engineering, model-driven architecture

1. Introduction

W3C's Semantic Web [1] provides a common framework that allows data to be shared and reused across application and enterprise. It is based on the Resource Description Framework (RDF), which describes various resources using XML for syntax and URIs for naming [9], and Web Ontology Language (OWL), which provides modeling constructs for specifying and inferring about knowledge [15]. As the Semantic Web shapes the future of the Web, it becomes more and more important in software engineering and enterprise application development. To meet the needs, a number of tools and systems for ontology development and management such as Protégé [26], Jena [4], Sesame [27], Pellet [24], KAON [14], RStar [13], and SnoBase [11], have been developed.

While these ontology engineering tools provide a relatively complete stack of ontology management support and are used successfully in certain domains, there still remains a gap between the ontology engineering tools and the traditional software engineering. For more than a decade, software engineering has been established on different modeling languages and methodologies such as OMG's Unified Modeling Language (UML). This difference in modeling languages and methodologies causes difficulties in large-scale enterprise application development involving the Semantic Web technologies. The existing ontology engineering tools provide only an ad hoc approach to bridging this gap with limited functionality and performance. The creation of ontologies and their use in software engineering projects is currently cumbersome and not seamless. The transformation of UML models to OWL ontologies and vice versa is conducted only in an ad hoc and incomplete way. Therefore, it is difficult to utilize the vast investment of enterprises in software engineering models, which are often accumulated over a decade, in ontology engineering. For the Semantic Web to have impact on enterprises and their business, and also to be widely accepted as a value-adding technology, bridging this gap in software and ontology engineering is critical.

The primary objective of our work is to bridge this gap between two different, but complementary engineering disciplines with a systematic approach. We leverage OMG's Model-Driven Architecture (MDA) [3] and Ontology Definition Metamodel (ODM) [23] to provide model transformation. This approach allows seamlessly supporting existing models in UML and other languages in Semantic Web-based software development. In addition, it allows exploiting the availability and features of UML tools for creation of vocabularies and ontologies. Furthermore, MDA enables code generation and facilitates tool development. This paper presents an MDA-based approach to ontology engineering. It describes the architecture of the ontology engineering system, and mappings between UML and OWL for model transformation. In addition, it presents the entire stack of individual components of the developed ontology engineering tool.

The rest of this paper is structured as follows: In Section 2, we describe a number of existing software tools for ontology development and management. It discusses a gap between these ontology engineering tools and the traditional software engineering tools. Sections 3 and 4 summarize technical background information on the Model-Driven Architecture and Ontology Definition Metamodel, respectively. In Section 5, we explain how EMF-based technologies for MDA and ODM are used to realize the proposed system for ontology engineering. Section 6 presents an implementation of the proposed system with the entire stack of components. Section 7 presents use scenarios illustrating how the features of the developed ontology engineering tool can be utilized in real-world applications. In Section 8, conclusions are drawn and future work is outlined.

2. Traditional Ontology Management Systems

In recent years, there has been a surge of interest in using ontological information for communicating knowledge among software systems. As a result, an increasing range of software systems engage in a variety of ontology management tasks, including the creation, storage, search, query, reuse, maintenance, and integration of ontologies. Recently, there have been efforts to externalize such ontology management burden from individual software systems and put them together in middleware known as an ontology management system. An ontology management system provides a mechanism to deal with ontological information at an appropriate level of abstraction. By using programming interfaces and query languages the ontology management system provides, application programs can manipulate and query ontologies without the need to know their details or to re-implement the semantics of standard ontology languages. Examples of such ontology management systems include Protégé [26], Jena [4], Sesame [27], Pellet [24], KAON [14], Jastor [21], D2RQ [18], RStar [13], and SnoBase [11].

Table 1 summarizes a few ontology management systems. It is important to note that these systems mainly focus on the manipulation of ontologies. The interoperability with other modeling languages and development tools comes as a secondary feature for these systems. That is, they assume separate workspaces for ontology management and software development, and fail to provide a tightly integrated environment for software and ontology engineering.

Table 1: Traditional Ontology Management Systems

Name	Functionalities	Standards	Interoperability
Jena	A program development framework for ontology manipulation and query	RDF, RDFS, OWL, SPARQL	N/A
Sesame	An RDF database allowing ontology manipulation and query	RDF, RDFS, OWL	N/A
Protégé	A graphical ontology editor and knowledge base framework for ontology manipulation and query	RDF, RDFS, OWL	Through plugins (with limited capability); UML → OWL ontology
KOAN	A suite of ontology management tools including ontology creation, ontology manipulation, and inference and query	RDF, RDFS, OWL	RDB schema → RDFS ontology
Jastor	A java code generator for creating Java beans from OWL ontologies	RDF, RDFS, OWL	OWL ontology → Java Beans
D2RQ	A language and a tool for specifying mappings between relational database schema and OWL ontologies	RDF, RDFS, OWL	RDB schema → OWL ontology

While these ontology engineering tools provide a stack of ontology management support, they also show certain limitations in supporting large-scale software engineering projects.

Participating in a number of enterprise application development projects by using the SnoBase and RStar Ontology Management System, we learned firsthand that it is critical to provide a comprehensive development environment including supporting tools and facilities for the application developers. A pick-and-choose approach to the best of the breed tools from different environments does not always work well for the majority of the developers and often results in a longer learning curve for the developers. A comprehensive ontology development environment often means a tight integration of tools for software *and* ontology engineering, and model import and transformation, among others.

Semantic markup languages such as W3C's RDF and OWL are based on the work in the logic and Artificial Intelligence communities, such as Description Logic and Knowledge Representation. The syntax of these languages is less intuitive to those trained for object-oriented programming and simple XML-based languages. The lack of a tightly integrated development environment for software and ontology engineering makes the job of subject matter experts and software engineers difficult, and often affects negatively to the adoption of the semantic technology in industry. An effective ontology application development environment should bridge this gap between software engineering and ontology engineering by providing a seamlessly integrated environment.

Another consideration for industry adoption of the semantic Web technology is the interoperability of the semantic markup languages with the well-established and widely-accepted industry standard modeling languages and methodologies such as Entity-Relation (ER) diagrams and Unified Modeling Language (UML). Enterprises developed software models in these languages for more than a decade and invested significantly in building systems around them. Despite all the theoretical advantages the semantic technology brings in, in practice, it is highly unlikely that the enterprises abandon the legacy systems and develop new systems around the semantic Web technology. Instead, users in industry would be interested in the interoperability of the modeling languages, and the reuse of the existing models and data with the semantic Web technology. The traditional ontology management systems currently provide only ad hoc and incomplete methods for the model interoperability. To address the practical requirements of industry, this paper introduces a novel approach to ontology engineering based on the Model Driven Architecture (MDA), which enables software engineers and users to design, build, integrate and manage ontologies and software applications in an integrated development environment.

3. Model-Driven Architecture

Before proposing the MDA-based system for ontology engineering, we summarize the Object Management Group's Model Driven Architecture, which is one of the two pillars of the system's architecture, along with Ontology Definition Metamodel.

In the history of software engineering, there has been a notable increase of the use of models and the level of abstraction in the models. Modeling has become separated from underlying development and deployment platforms, making them more reusable and easier to create and modify by domain experts, and requiring less knowledge of specific deployment systems. This trend places software modeling closer to knowledge engineering. The current stage in this evolution is the *Model Driven Architecture*, which grew out of the standards work conducted in the 1990s for the Unified Modeling Language (UML).

The basic idea of MDA is that the system functionality is defined as a platform-independent model, using an appropriate specification language and then translated to one or more platform-specific models for the actual implementation. To accomplish this goal, the MDA defines an architecture that provides a set of guidelines for structuring specifications expressed as models. The translation between platform-independent model and platform-specific models is normally performed using automated tools. Specifically, MDA defines three levels of abstraction: *Computation Independent Model* (CIM), *Platform Independent Model* (PIM) and *Platform Specific Model* (PSM). CIM is a view of a system that does not show the details of a system structure. In software engineering, it is also known as a *domain model*, which is concerned by domain experts. It is similar to the concept of ontology. PIM is a model that is computation dependent, but it is not aware of specific computer platform details. In other words, it is targeted for a technology-neutral virtual machine. Specification of complete system is completed with PSM. The goal is to move human work from PSM to CIM and PIM, and let the detail implementation for a specific platform be generated as much as possible by automated tools which perform the transformation from PIM to PSM.

MDA comprises of a four-layer metamodeling architecture: meta-metamodel (M3) layer, metamodel (M2) layer, model (M1) layer, and instance (M0) layer. Also, it utilizes several complementary standards from OMG including *Meta-Object Facility* (MOF), *Unified Modeling Language* (UML) and *XML Metadata Interchange* (XMI). On the top of the MDA architecture is the meta-metamodel, i.e., MOF. It defines an abstract language and framework for specifying, constructing and managing technology neutral metamodels. It is the foundation for defining any modeling language such as UML or even MOF itself. MOF also defines a framework for implementing repositories that hold metadata (e.g., models) described by metamodels [22]. The main objective of having the four layers with a common meta-metamodel is to support multiple metamodels and models and to enable their extensibility, integration and generic model and metamodel management.

All metamodels, standard or custom, defined by MOF are positioned on the M2 layer. One of these is UML, a graphical modeling language for specifying, visualizing and documenting software systems. With *UML profiles*, basic UML concepts (e.g., class, association, etc.) can be extended with new concepts (*stereotypes*) and adapted to specific modeling needs. The models of the real world, represented by concepts defined in the corresponding metamodel at M2 layer (e.g., UML metamodel) are on M1 layer. Finally, at M0 layer, are things from the real world. Another related standard is XMI. It defines mapping from MOF-defined metamodels to XML documents and schemas. Because of versatile software tool availability for XML, XMI representations of models, metamodels and meta-metamodel facilitate their sharing in software application development.

MOF tools use metamodels to generate code for managing models and metadata. The generated code includes access mechanisms, or application programming interfaces, to read and manipulate, serialize and transform, and abstract the details of various interfaces based on access patterns. *Eclipse Modeling Framework* (EMF) [20] provides a Java implementation of a core subset of the MOF API. EMF started out as an implementation of the MOF specification, and evolved into a generic modeling framework and code generation facility for building tools and other applications based on a structured data model. The MOF-like core metamodel in EMF is called *Ecore*. From a model specification written in XMI, EMF generates tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modeling tools like Rational Rose, then imported into EMF. It is important to note that EMF provides the foundation for interoperability with other EMF-based tools and applications. The proposed MDA-based system leverages EMF for implementing ontology management tools which run on the Eclipse environment, and utilizes its support for model interoperability.

4. Ontology Definition Metamodel

MDA and its four-layer architecture provide a solid basis for defining metamodels of any modeling language, and so provide a foundation for bringing together software engineering and methodologies such as UML with the semantic technology based on W3C's RDF and OWL. Once a semantic markup language such as OWL is defined in MOF, its users can utilize MOF's capabilities for modeling creation, model management, code generation, and interoperability with other MOF-defined metamodels.

Another OMG standard, Ontology Definition Metamodel (ODM) [23] took this approach. To comprehend common ontology concepts, ODM used as a starting point OWL, which is the result of the evolution of existing ontology representation languages. ODM defined individual constructs of OWL in MOF, creating an ODM metamodel. To leverage graphical modeling capabilities of UML in dealing with OWL constructs, ODM also defined an ontology UML profile to support UML notation for ontology definition. This profile enables graphical editing of ontologies in OWL using UML diagrams as well as other benefits of using mature UML CASE tools. Finally, the following bi-directional mappings between metamodels complete the picture:

1. mappings between OWL and ODM,
2. mappings between ODM and the ontology UML profile, and
3. mappings from the ontology UML profile to other UML profiles.

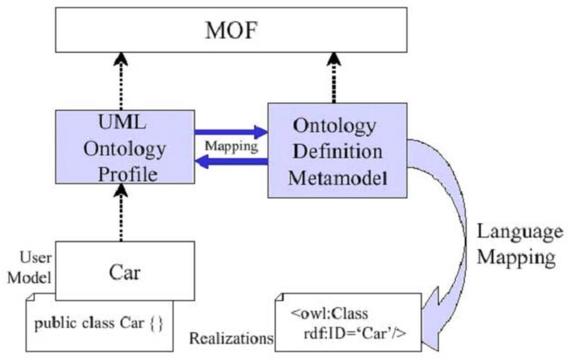


Figure 1: Bi-directional mapping among metamodels

Figure 1 shows a simple example of the bi-directional mappings between metamodels. In practice, both UML and ODM models are serialized in XMI, and OWL model in XML, the two-way mappings can implement XSLT-based transformations. Our work utilized EMF-based transformations, instead of XSLT, to leverage EMF's generic modeling framework and code generation facility for building tools and other applications. We implemented *EODM* (EMF-based ODM), which is the underlying object model generated from ODM by using EMF, for model transformations between OWL and UML. More details will be given in the next section.

Before moving to the main body of this paper, it is useful to briefly mention yet another related effort from W3C, namely, *Ontology Driven Architecture* (ODA) [16]. It combines MDA with the semantic technology differently from the ODM approach. It attempts to augment the MDA standards and methodology stack with the semantic technology to improve the discipline. It aims to enable unambiguous representation of domain terminology, distinct from the rules, enable automated consistency checking and validation of invariant rules, preconditions, and post-conditions, and support knowledge-based terminology mediation and transformation for increased scalability and composition of components. This effort still is in its infancy and at a draft stage.

5. EMF-Based Ontology Engineering System

For realizing an MDA-based system for ontology engineering, we utilized the Eclipse Modeling Framework, which is open source MDA infrastructure for integration of modeling tools [20]. A model specification described in various modeling languages including UML, XML Schema, and annotated Java source can be imported into EMF. Then EMF produces a set of Java classes for the model, a set of adapter classes that enable viewing and editing of the model, and a basic editor. In its current implementation, EMF does not provide formal semantics definitions, inference and the related model specifications. Our work adds this capability to EMF for providing a comprehensive ontology engineering environment and dynamic application integration.

For adding the semantic model transformation capability to EMF, we leverage the specification of Ontology Definition Metamodel. By using EMF and ODM, we generated a foundational memory model, i.e., Java classes, for the constructs of OWL. This foundational memory model is referred to as *EODM* (EMF-based Ontology Definition Metamodel). By adding several necessary helper classes and methods to EODM, we can use it to create, edit, and navigate any models in OWL.

Also, we added an OWL parser to EODM, which can load OWL files into EMF and generate OWL files from EMF, i.e., serialize EMF models to standard OWL files in XML. The parser utilizes an XMI adaptor which enables the transformation between the OWL models and EODM Ecore models. The transformation is made possible by the bi-directional mapping between OWL and the Ecore metamodel. The transformation opens a way to interoperability between OWL models and other EMF supported models, which currently include ones defined in UML, XML Schema, and annotated Java classes. The support of other models such as Entity Relationship models in EMF will be provided in the near future. By leveraging the OWL parser and the bi-directional transformation between the OWL models and the Ecore models, ontology application developers can develop ontologies using their favorite model building tools, import them into EMF, transform their models into OWL ontologies, enrich them with semantics, leverage their inference capability, and utilize the comprehensive development facility of Eclipse and EMF.

To be more specific, the EODM Ecore model is the MOF core model that represents ontologies in memory. It is an intermediate model for imported and transformed legacy models, as well as the generated ontology, Java code, Java editor and Java edit. The development environment allows its users to manipulate EODM Ecore models, enrich it with semantic specification, and generate Java code. A default set of bi-directional mappings between metamodels of legacy models and OWL are developed in EMF. Eclipse plug-in developers can extend the mappings to handle other types of legacy models, or other elements in legacy models specifying semantics. In the generated Java code, a small foot-print inference engine is shipped with the code and can be invoked by applications. The generated Java editor and Java edit provide ready-to-use visual tools to populate or manipulated instances of OWL models. The visual tools are actually copies of the standard methods of supporting application development in EMF. Figure 2 illustrates the operation of the EMF-based ontology engineering system.

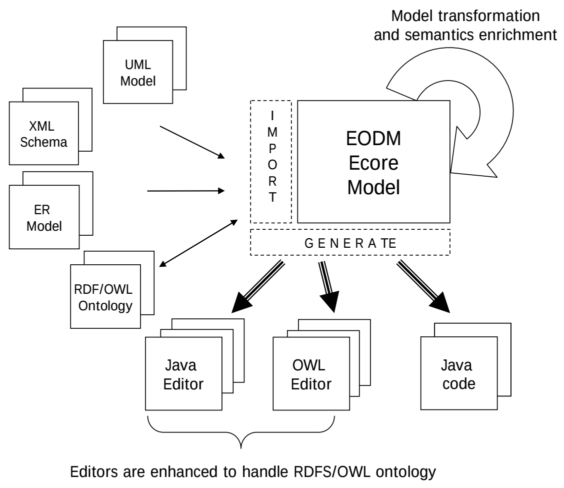


Figure 2: EMF-based ontology engineering system

6. Components of the EMF-Based Ontology Engineering System

This section presents the entire stack of components of the developed, EMF-based ontology engineering system. We had two primary design objectives for this system: first, support for the entire lifecycle of ontology engineering, and, second, avoiding reinvention of tools and facilities that are already proven to work in software engineering. To achieve these objectives, we designed a software stack which consists of six interdependent layers.

At the core of this EMF-based ontology engineering system is the EODM model, which is derived from the Ontology Definition Metamodel and implemented in Eclipse Modeling Framework. The bottom layer, *EODM core model*, provides the basic Java programming model for OWL ontologies with all the necessary getter and setter functions. It is automatically generated by EMF from the Rational Rose model for OWL. To this generated core model implementation, certain *utility classes and methods* are added, to benefit Java programmers. On top of the EODM core model comes the *OWL parser* which parses OWL ontologies, translates them into EODM models, and serializes EODM models to standard RDF/XML files. The next layer is the *inference engine*. It takes an EODM model as input, and executes user queries, reasoning about instances and relationships among instances and classes. The next layer is the *model transformation engine*. It imports existing conceptual models represented in various modeling languages such as UML, ER diagrams, and Java interfaces. Then, it transforms the models into one or more EODM models. Finally, the *OWL editor* provides a graphical ontology authoring environment where OWL ontologies in graphic notations are serialized to OWL files in a standard XML format. Figure 3 shows the components of the EMF-based ontology engineering system. In the rest of this section, we describe each component in detail.

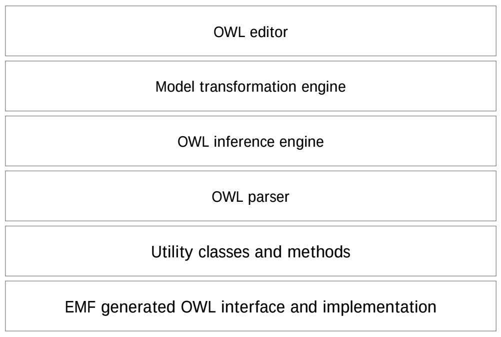

OWL editor
Model transformation engine
OWL inference engine
OWL parser
Utility classes and methods
EMF generated OWL interface and implementation

Figure 3: EMF-based ontology engineering system architecture

6.1. EODM Core Model

The EMF-based ontology engineering system provides tightly integrated environment for software and ontology engineering, providing a stack of useful components. EODM provides the run-time library that allows applications to input and output OWL ontologies, manipulate them by using Java objects, invoke the inference engine and access result sets, and transform among ontologies and other legacy models.

The EODM core model provides useful classes and methods to access OWL ontologies and their instances. Its metamodel is defined in the Ontology Definition Metamodel (ODM) specification [23]. It is an MOF2 compliant metamodel that allows users to define ontologies by using those constructs defined in RDF Schema and OWL. ODM comprises of two packages that define the metamodels of RDF and OWL, respectively. The OWL package inherits classes from the RDF package, and extends it. Figure 4 illustrates the class definition of the RDF package. The UML model of the packages is augmented by a number of bi-directional references to generate APIs that leverage notification and messaging mechanisms in EMF. Also, there are certain design patterns, such as Factory and Singleton, embedded in the code generation engine of EMF. Therefore, the EODM core model automatically complies with the design practices and benefit software engineers.

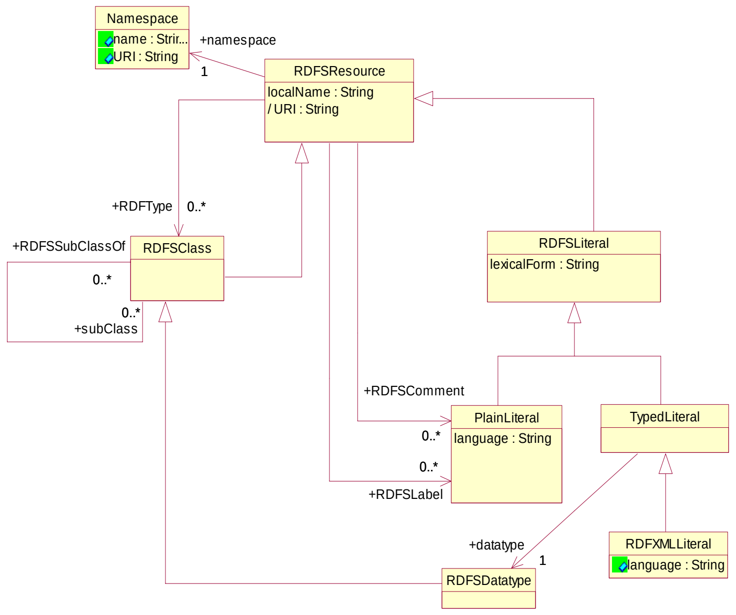


Figure 4: Class definition in ODM

6.2. OWL Parser

The OWL parser analyzes the XML syntax of OWL files and generates EODM models and a set of RDF triple statements. Figure 5 shows the parser process. The parser utilizes XML SAX API to correctly parse each node and its attributes. Then, the *RDF triple analyzer* assembles the resulting nodes and attributes to generate RDF triples by maintaining a state stack for keeping node and property states. In RDF and OWL, knowledge is simply a collection of statements, each with a subject, verb and object, and nothing else [25]. The RDF triple statements can be directly used by applications. They can be asserted into a working memory of inference engine for reasoning. They also can be stored in a database for triple-based RDF graph retrieval. A model wrapper can envelop RDF triples into an EODM model. Therefore, the OWL parser can create both RDF triple statements and an object-oriented memory model for further manipulation in applications. In addition, we also provide a tool for serializing EODM models into standard OWL files.

The OWL parser is completely compliant with W3C's XML syntax specification and passes all W3C's positive and negative test cases [7]. It utilizes a streaming XML parser, i.e., SAX parser, and once an RDF statement is formed, the parser can immediately export the statement. This property allows the parser to require minimal amount of memory and thus to be scalable in handing large-scale models. Also, it is important to note that the OWL parser can be used independently of other components of the system.

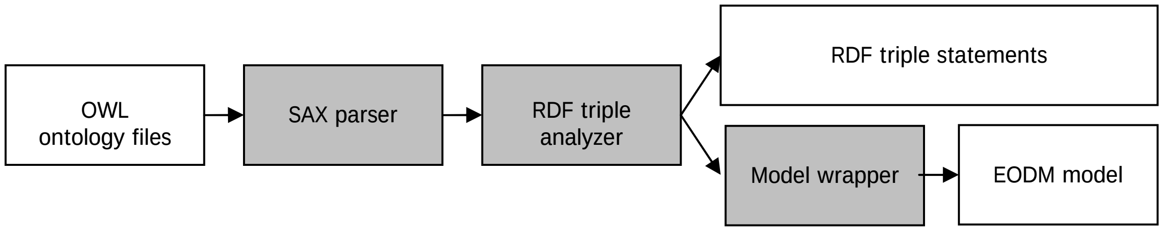


Figure 5: OWL parser process

6.3. Inference Engine

The inference engine of the EMF-based ontology engineering system approaches the core inference problem of OWL by a structural subsumption algorithm. The present Description Logic classification algorithm is based on the tableau algorithm [8], which can provide sound and complete reasoning on a very expressive language by satisfaction test. However, this approach focuses on the tractability of a single subsumption test and the worst case computational complexity is NEXP-time [12]. In practical cases, however, an algorithm with high efficiency but less expressiveness would be more useful in supporting large-scale taxonomic classification problems. To achieve a balance between efficiency and expressive power, we leveraged the structural subsumption algorithm, which is known to be an efficient technique but also known to be limited due to its inability to provide complete reasoning for expressive languages. The concepts and axioms supported by this approach is defined as follows:

Concepts: (Cyclic concept definitions are not supported)

C, D $\rightarrow$ A (atomic concept), $\top$ (universal concept), $\perp$ (bottom concept),
$C \sqcap D$ (intersection), $C \sqcup D$ (union), $\exists R.C$ (some value from restriction)
$\forall R.C$ (all value from restriction), $\exists R.\{x\}$ | (hasValue)

Axioms:

Axioms $\rightarrow$ $C \sqsubseteq D$ (concept inclusion), $R \sqsubseteq S$ (role inclusion)

In ontologies without an acyclic definition, every defined concept is treated as a restriction on some properties, and an atom concept is treated as a "special" restriction. For example, $C \equiv A \sqcap B \sqcap \forall R.(\forall S.C)$ is treated as a concept with restriction on R_A, R_B, R, where R_A, R_B is special restriction brought by atom concept A and B. To decide whether two concepts are subsumed by each other, we can recursively compare those restrictions by applying basic comparison rules captured in Table 1. Figure 6 illustrates a simple example of a structural subsumption test by using the comparison rules.

Table 2: Comparison rules for structural subsumption tests

Concept A	Concept B	$A \sqsubseteq B$ Condition
$\exists R.C$	$\exists S.D$	Iff $R \preceq S$ and $C \sqsubseteq D$
$\forall R.C$	$\forall S.D$	Iff $S \preceq R$ and $C \sqsubseteq D$
$\geqslant nR.C$	$\geqslant mS.D$	Iff $R \preceq S$ and $C \sqsubseteq D$ and $n \leqslant m$
$\leqslant nR.C$	$\leqslant mS.D$	Iff $S \preceq R$ and $D \sqsubseteq C$ and $n \geqslant m$

Given A≡∃R1.C⊓∃R2.(∀R4.D)⊓∀R3.E, B≡∃R2.(∀R4.F) ⊓∀R3.G, D⊑F, E⊑G, it can be concluded that A⊑B.

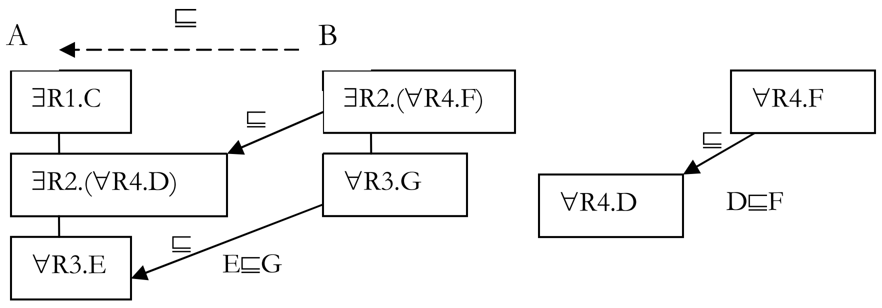


Figure 6: A simple structural subsumption test example

The main idea of the extended structural subsumption algorithm is to leverage the information of concept definitions by maintaining a classification tree to perform taxonomic classification. The algorithm starts with building the classification tree. Given an axiom C⊑D, the algorithm first recursively classifies C, D and all the sub constitutes until all of them have been correctly linked into the classification tree. Then, it adds the subsumption link between C and D, which will automatically remove all the outdated links. More details of the inference algorithm is a subject of another paper we are preparing. Figure 7 shows an example of classifying a TBox which contains the following definitions:

C ≡ ∃R3.(A⊓B) ⊓ ∃R4.D
F ≡ ∃R1.A ⊓ ∃R2.B
D ⊑ E,
∃R3.A ⊑ ∃R2.B
∃R4.E ⊑ ∃R1.A

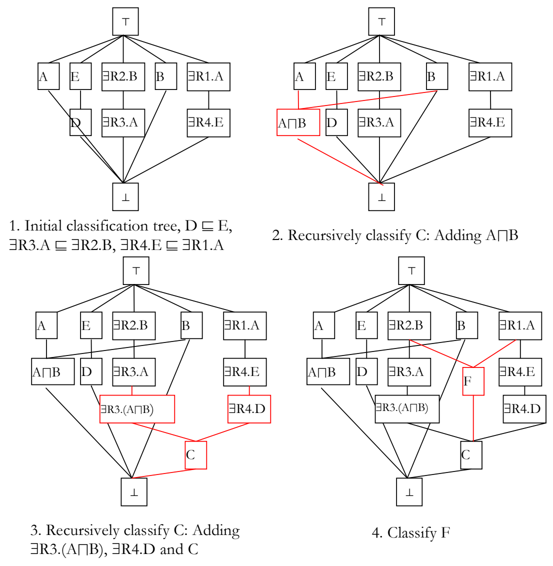

⊤
A
E
∃R2.B
B
∃R1.A
D
∃R3.A
∃R4.E
⊥
1. Initial classification tree, D ⊑ E, ∃R3.A ⊑ ∃R2.B, ∃R4.E ⊑ ∃R1.A
A⊓B
2. Recursively classify C: Adding A⊓B
∃R3.(A⊓B)
∃R4.D
C
3. Recursively classify C: Adding ∃R3.(A⊓B), ∃R4.D and C
F
4. Classify F

Figure 7: An example of classifying a TBox

6.4. Model Transformation

This layer in the EMF-based ontology engineering system addresses the ontology acquisition and model interoperability issues, as we discussed earlier. Enterprises developed IT models in various modeling languages such as UML and ER diagrams for several decades and invested heavily in building systems around them. It is important for the enterprises to protect their investment in the legacy systems. Also, it is important to leverage domain knowledge captured in the existing IT models. Thus, users in industry are interested in the interoperability of the modeling languages and the reuse of the existing models with the semantic Web technology. The interoperability allows exploiting the availability and features of UML tools for creation of vocabularies and ontologies. In addition, it allows augmenting the legacy models with formal semantics, and enabling an inference capability with the models, which can return sound and complete query results.

Figure 8 shows the Ecore metamodel structure. There already exist transformations defined between Ecore model and legacy modeling languages such as UML, XSD and Java interfaces. In EODM, we define a transformation between Ecore and OWL. Then, we utilize Ecore as the intermediate model to support model transformation between OWL and other modeling languages.

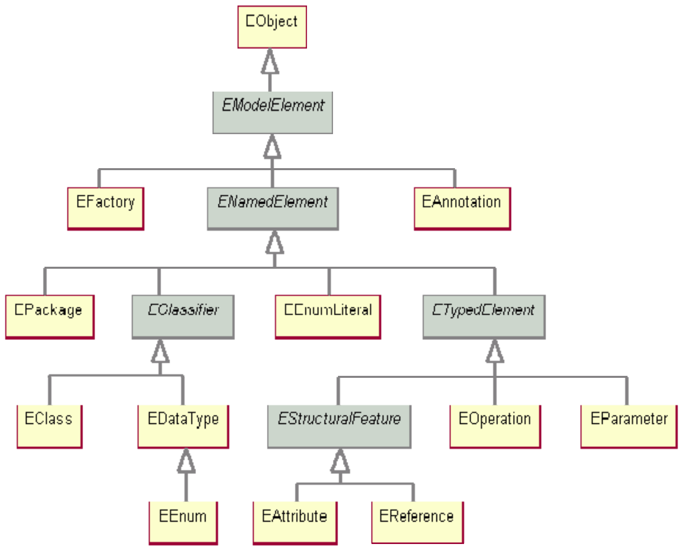


Figure 8: Ecore metamodel structure

Figure 9 depicts the bi-directional mappings defined between the Ecore metamodel and the EODM OWL metamodel. An OWL Ontology is transformed to an EPackage and vice versa; an OWL class to an EClass, etc. While the transformation from OWL to Ecore model looks straightforward, there are a few gaps. As in UML, Eclass is a first-class entity in the Ecore model. All other entities such as properties are subordinates to Eclass. In OWL, however, all entities in OWL are equal. Thus, different entities must have different names in OWL. For example, if two properties belonging to two different Eclasses have an identical name, a straightforward transformation will cause a name conflict problem. The EODM Transformation engine renames properties with an identical name to ensure a unique name for every entity. Another gap comes from the difference in expressiveness from different modeling languages. OWL is a formal language which is based on Description Logic. OWL is more expressive than the Ecore model. There are several OWL constructs that the Ecore model does support, e.g., OWL property restrictions used for precise definition of concepts. Therefore, some semantics are lost inevitably when conducting transformation from OWL to Ecore. Also, The Ecore model does not support inference of OWL. Particularly, anonymous classes created by using OWL restrictions make the situation with inference even more difficult. To address these gaps, the EODM Model Transformation engine currently employs the following tactics:

- It appends all unsupported OWL constructs as comments;
- It utilizes the inference engine during transformation to capture all implicit subsumption relationships;
- It only transforms named OWL classes, and discards all anonymous classes; and
- It renames properties with an identical name to ensure a unique name for every entity.

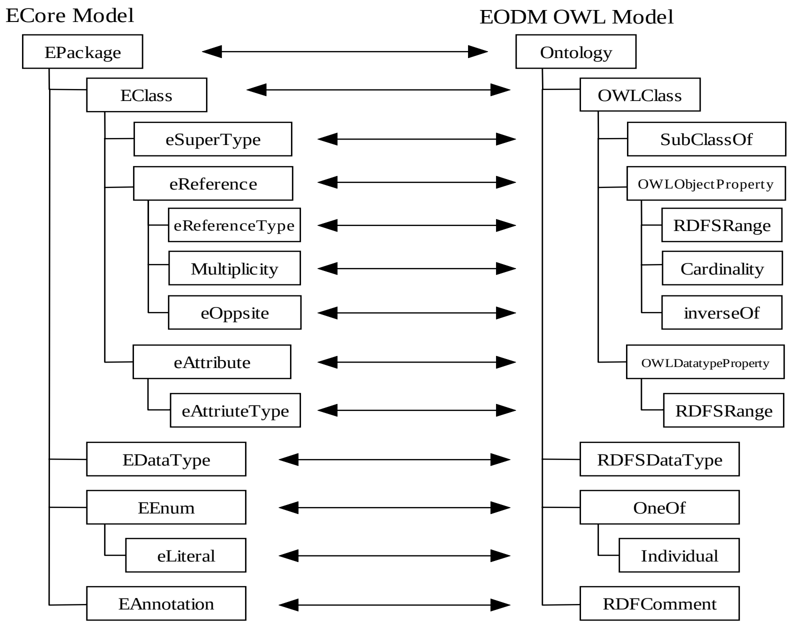
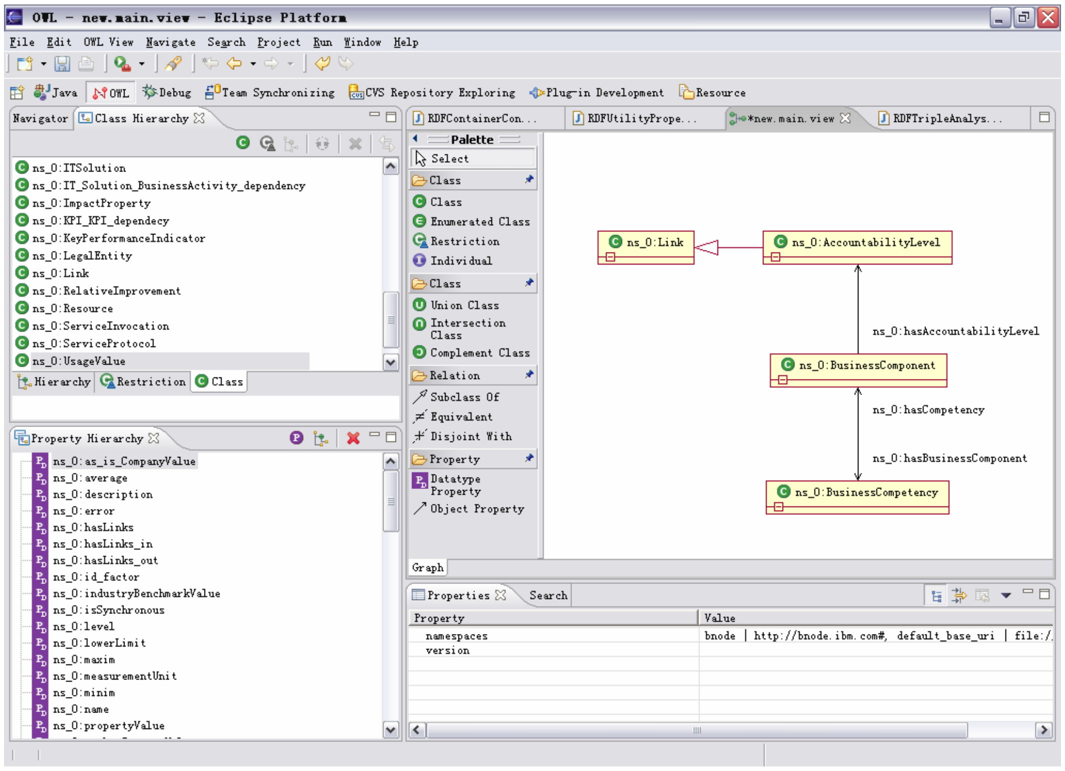


Figure 9: Transformation between OWL and Ecore

6.5. OWL Editor

Finally, the OWL editor provides ontology authors with a GUI which enables them to visually create and update OWL ontologies. We utilized EMF to automatically generate a tree-based editor in Eclipse, and replaced the default XMI parser in the generated editor with the OWL parser of EODM. The editor framework in EMF follows the Model-Control-View pattern [6] and uses the adaptor pattern [6] to provide a generic content management capability. In addition to EMF, we utilized the Graphic Editing Framework (GEF) [19] in developing the OWL editor, to provide the foundation for graphic views of OWL ontologies. GEF also supports drag and drop operations and drawing functions. The OWL editor provides two hierarchical views; one for OWL classes and restrictions, and the other for OWL objects and datatype properties. The OWL editor provides multiple views of OWL ontologies and support different development perspectives. Operations in different views are automatically synchronized in the Eclipse platform. Figure 10 gives a screenshot of the OWL editor in EODM.

In the left hand side of the screen, there are two views of class and property hierarchy, which are constructed based on rdfs:subClassOf and rdfs:subPropertyOf. Users can also add classes and properties such as subClassOf and subPropertyOf in the two trees. In the right hand side, there are a palette, a graphical editor and a property view. Users can drag and drop elements in the palette to the editor. Detailed information about the ontology and its elements that are not showed in the limited space of the editor is viewed and edited in the property view.



Property	Value
namespaces	bnode | http://bnode.ibm.com#, default_base_uri | file:/.
version	

Figure 10: Screenshot of OWL editor

7. Use Scenarios

This section presents a use scenario illustrating how the features of the proposed EMF-based ontology engineering tool can be utilized in real-world applications. Our example is the *model-driven business transformation* [10]. Business transformation employs business models such as *component business models* [17] to identify opportunities for reducing costs or improve business processes. The model-driven approach to business transformation requires a model representation of a variety of business entities such as business processes, components, competencies, activities, resources, metrics, KPIs (Key Performance Indicators), etc. and their relations. Semantic models or ontologies provide useful representation of business models because they can effectively represent different types of relations among business entities. Also, the automatic reasoning capability of semantic models provides an effective method for analyzing business models for identifying cost-saving or process improvement opportunities.

For example, business performance metrics are associated with business activities. By using the relations between business activities and metrics, and also the relations between business components and business activities represented in a semantic model, a business analyst can infer relations between business components and metrics. This type of analysis provides business insights into how the corporate can improve its performance metrics by addressing issues with the business components associated with the selected set of metrics. Then, by identifying, again in the semantic model, IT systems associated with the business components, the analyst may be able to suggest recommendations about IT system management to improve performance metrics.

The first step in realizing this model-driven business analysis scenario is the construction of semantic models of various business entities including business processes, components, competencies, activities, resources, operational metrics, KPIs. In many cases in most enterprises, the classes and relations of these business entities are already captured in certain legacy modeling languages such as UML class diagrams, ER diagrams, relational data models, Java interfaces, spreadsheets, or text documents. Therefore, the task of semantic model construction simplifies to transforming the legacy models and merging them into OWL ontologies. The merged OWL ontologies can be enriched with certain semantics such as generalization and specification, and cardinality constraints to enhance the effects of business analysis queries.

To summarize the model transformation process using the EMF-based ontology engineering system, it starts by capturing formal and informal semantics of legacy models. The model transformation engine transforms *formal* semantics of input legacy models into OWL models, by utilizing pre-defined mappings between OWL and the metamodels of the input models. The ontology engineering system allows an expert to look into annotations and code of legacy models, and represent the semantics in OWL models. *Informal* semantics are captured as additional axioms and added to the OWL models by using the OWL editor. Optional functions of the system, such as the source code analysis or natural language processing, facilitate automatically capturing of certain informal semantics and improve the productivity of human experts. The overall process of capturing of formal and informal semantics of legacy models and representing them in OWL models is referred to as *semantics enrichment*.

We have implemented a non-trivial model transformation by using the EODM transformation engine. The source model is the Financial Business Object Model (FS-BOM) from IBM's Information FrameWork (IFW). IFW permits many types of information models required by complex enterprise systems for storing and correlating classes in a consistent manner. FS-BOM is a component of the overall suite of IFW for Financial Services. It was written in UML and provides an enterprise-wide, generic and flexible model for financial services business. It is usually used as a starting point for analysis and design of business systems. Its rich content can be viewed from the fact that it has 582 classes and 5878 attributes and associations. This complicated structure indicates that more knowledge may be buried deep in the FS-BOM. Therefore, an inference-enabled representation, for example, by using OWL, is desirable. This transformation was conducted by using the EODM transformation engine.

8. Concluding Remarks

As the Semantic Web shapes the future of the Web, it becomes more and more important in software engineering and enterprise application development. However, the adoption of Semantic Web by industry has been slowed by a gap between ontology engineering tools and the traditional software engineering. Ontology engineering and software engineering have been established on different modeling languages and methodologies, which has caused difficulties in large-scale enterprise application development involving the Semantic Web technologies. Currently, transformation of UML models to OWL ontologies and vice versa is conducted only in an ad hoc and incomplete way.

This paper presented a novel approach to bridging this gap between two different, but complementary engineering disciplines with a systematic approach. We leveraged OMG's Model-Driven Architecture and Ontology Definition Metamodel to provide model transformation, utilizing underlying standards including MOF-based metamodels, XMI representation, UML extension with profiling, and EMF implementation of MOF. This approach allows seamlessly supporting legacy models in UML and other languages in Semantic Web-based software development. In addition, it allows exploiting the availability and features of UML tools for creation of vocabularies and ontologies. Furthermore, it supports code generation and facilitates tool development. This paper presented the methodology and architecture of the EMF-based ontology engineering system, and mappings between UML and OWL for model transformation. In addition, it presented the entire stack of the developed ontology engineering system. Finally, it presented use scenarios illustrating how the features of this system can be utilized in real-world applications.

This MDA-based approach to ontology engineering is still in its infancy. For this approach to meet its promises and scale for industry applications, a number of technical challenges need to be addressed. Some directions for further investigation include:

- A complete definition of bi-directional mappings between the Ecore metamodel and semantic metamodels to support sound and complete model transformation;
- Support for more legacy modeling languages and methodologies in addition to UML, XSD and Java interfaces which we have addressed in the current system, e.g., relational data models and spreadsheets traditionally popular in the business environment;
- Validation of the proposed advantage of utilizing features of visual UML tools for creating and editing ontologies in real-world applications;
- Evaluation of EMF's capability of code generation for facilitating tool development;
- Augmenting the proposed model transformation method with capabilities for source code analysis and text mining to facilitate acquisition of certain informal semantics of legacy models; and
- Maturation of the holistic EMF-based ontology engineering framework by applying and validating it in real-world business applications.

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic WEB," Scientific American, 2001.

[2] D. Brickley and R. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema," W3C Recommendation, http://www.w3.org/TR/rdf-schema/, 2004.

[3] A. Brown, "An introduction to Model Driven Architecture – Part I: MDA and today's systems", http://www-106.ibm.com/developerworks/rational/library/3100.html, 2004.

[4] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, "Jena: Implementing the Semantic Web Recommendations", Proc. of WWW 2004.

[5] D. Djuric, D. Gaševic, and V. Devedžic, "Ontology Modeling and MDA," Journal of Object technology, Vol. 4, No. 1, 2005.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley Professional; 1st edition, January 15, 1995.

[7] J. Grant and D. Beckett, "RDF Test Cases," http://www.w3.org/TR/rdf-testcases/, 10 Feb. 2004.

[8] J. Hladik, "Implementation and Optimisation of a Tableau Algorithm for the Guarded Fragment," Lecture Notes In Computer Science; Vol. 2381 archive, Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Pages: 145 - 159, 2002.

[9] G. Klyne and J. Carroll, "Resource Description Framework (RDF): Concepts and Abstract Syntax," W3C Recommendation, http://www.w3.org/TR/rdf-concepts/, 2004.

[10] J. Lee, "Model-Driven Business Transformation and Semantic Web," The Communications of the ACM, Special Issue on Semantic eBusiness Vision, December 2005.

[11] J. Lee and R. Goodwin, "Ontology Management for Large-Scale E-Commerce Applications," International Workshop on Data Engineering Issues in E-Commerce, Tokyo, Japan, April 9, 2005.

[12] C. Lutz, "NEXP TIME-Complete Description Logics with Concrete Domains," ACM Transactions on Computational Logic (TOCL) archive, Volume 5, Issue 4, October 2004.

[13] L. Ma, Z. Su, Y. Pan, L. Zhang, T. Liu, "RStar: An RDF Storage and Query System for Enterprise Resource Management", Proc. of ACM CIKM, pp. 484-491, 2004.

[14] D. Oberle, R. Volz, B. Motik, S. Staab, "An Extensible Ontology Software Environment," In Steffen Staab and Rudi Studer, Handbook on Ontologies, chapter III, pp. 311-333. Springer, 2004.

[15] M. K. Smith, C. Welty, and D. L. McGuinness, "OWL Web Ontology language Guide," http://www.w3.org/TR/owl-guide/, 2004.

[16] P. Tetlow, et al., "Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering," http://www.w3.org/2001/sw/BestPractices/SE/ODA/, 2005.

[17] CBM: Component Business Modeling, http://www-306.ibm.com/e-business/ondemand/us/innovation/cbm/cbm_b.shtml.

[18] D2RQ V0.4: Treating Non-RDF Databases as Virtual RDF Graphs, http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rq/.

[19] GEF (Graphical Editing Framework), http://www.eclipse.org/gef, 2004.

[20] EMF (Eclipse Modeling Framework), http://www.eclipse.org/emf, 2004.
[21] Jastor, http://jastor.sourceforge.net/.
[22] MOF: Meta-Object Facility, Version 1.4, http://www.omg.org/technology/documents/formal/mof.htm.
[23] ODM: Ontology Definition Metamodel, http://www.omg.org/docs/ontology/04-08-01.pdf, 2004.
[24] Pellet: an Open-Source Java Based OWL DL Reasoner, http://www.mindswap.org/2003/pellet/index.shtml.
[25] Primer: Getting into RDF & Semantic Web using N3, http://www.w3.org/2000/10/swap Primer.html.
[26] Protégé, http://protege.stanford.edu/index.html, 2004.
[27] Sesame, an Open Source RDF Database with Support for RDF Schema Inferencing and Querying," http://www.openrdf.org/, 2002.