Bridging the Gap Between the Model-Driven Architecture and Ontology Engineering

Stephen Cranefield and Jin Pan

Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand

scranefield@infoscience.otago.ac.nz

ABSTRACT

Software engineers have many robust commercial tools available to them for creating and manipulating models. Due to the widespread adoption of the Object Management Group (OMG) standards for metamodel definition, model serialisation and programmatic access to models, many of these tools are interoperable. Currently this is not the case for ontology engineering tools. This paper discusses the potential benefits of making the OMG's Model Driven Architecture (MDA) technology applicable to ontology engineering, and in particular, describes a technique for converting ontologies serialised using the XML Metadata Interchange (XMI) format to an equivalent representation using the Resource Description Framework (RDF), without any loss of information. The resulting models can then be analysed and transformed using existing RDF tools. The technique is applicable to any ontology modelling language that has its abstract syntax defined using the OMG's Meta Object Facility (MOF) model.

This research helps to bridge the gap between the MDA and ontology engineering by providing a technique based on the familiar RDF language for defining transformations between other types of model (such as UML) and ontologies, between different ontology modelling languages, or to modify ontologies without changing the language.

KEYWORDS

Model-driven Architecture (MDA); Ontologies; MOF; JMI; RDF; Jena; NetBeans MDR; ODM

1 INTRODUCTION

The software infrastructure of large businesses and other organisations has become increasingly complex as advances in computing and networking technologies and changes in organisational structure have led to increases in the number, diversity, and interconnectivity of information systems. This, in turn, has resulted in advances in software analysis, design and development aimed at enabling software developers to work at increasingly higher levels of abstraction. One

significant example of this trend is the development of the Model Driven Architecture approach to software development (OMG, 2001) proposed by the Object Management Group (OMG)—an industry body devoted to the development and promotion of standards for enterprise computing (OMG, 1997). The MDA is based on the principle of using modelling languages to specify a system at various levels: a computation-independent model (CIM) to represent the system's environment and requirements, a platform-independent model (PIM) that describes the system architecture in a technology-neutral manner, and a platform-specific model (PSM) that expands the PIM with details specifying how the model is to be implemented using a specific *platform*—a set of subsystems and technologies. The vision underlying MDA is that automated mappings can be used to move from a PIM to a PSM (once a specific platform has been identified) and to round-trip between a PSM and code. The practical realisation of this vision is based on a number of existing and new OMG technologies, in particular the Meta-Object Facility (MOF) (OMG, 2002b), the XML Metadata Interchange (XMI) framework (OMG, 2002a), the Unified Modeling Language (UML) (OMG, 2003b), UML profiles and the Query/Views/Transformations (QVT) language (OMG, 2005). These have been designed to provide a general framework for defining modelling languages and corresponding UML-based graphical notations, and the facility to build editors and model repositories that have standard formats and interfaces for exchanging and interacting with models.

Concurrently with the OMG's work, the World Wide Web consortium (W3C) and a growing research community have been working to bring about Tim Berners-Lee's vision of the "Semantic Web" (Berners-Lee, 1998). As the Web grew rapidly and W3C technologies such as HTTP and XML became widely adopted, vast amounts of machine-readable information became available. However, although machines could access this information, there remained the significant challenge of structuring information so that automated processes could locate on-line information relevant to a particular task and integrate it with other information sources. The technology underlying the Semantic Web is designed to address these issues by providing a simple language for encoding semi-structured information within and about resources on the Web: the Resource Description Framework (RDF) (Manola and Miller, 2004), as well as predefined vocabularies for defining conceptual models of a domain: RDF Schema (RDFS) (Brickley and Guha, 2004) and the Web Ontology Language (OWL) (McGuinness and van Harmelen, 2004). Using RDF and the modelling terminology defined by RDFS or OWL, *ontologies* can be defined to model the concepts in a domain, the relationships between them, and the properties that can be used to describe instances of those concepts.

Ontologies have also been seen as important in other fields of research, such as multi-agent systems (MAS) and natural language processing. In general, across the fields in which ontologies are used, the ontology representation language can "range from a Taxonomy (knowledge with minimal hierarchy or a parent/child structure) to a Thesaurus (words and synonyms) to a Conceptual Model (with more complex knowledge) to a Logical Theory (with very rich, complex, consistent and meaningful knowledge)" (OMG, 2003a). As well as this diversity of languages, there is also a diversity of tools used to create ontologies, with a lack of interoperability between them. This was recently recognised as a challenge for the community by the European Knowledge Web research network: "There are many existing ontology development tools, and they are used by different groups of people for performing diverse tasks. Although each tool provides different functionalities, users tend to use just one tool, as they are not able to exchange their ontologies from one tool to another" (Knowledge Web, 2005).

Supporting a heterogeneity of modelling languages, while providing standard representations and application programming interfaces (APIs) for model repositories and other tools, is one of the aims of the model-driven architecture. There would therefore be great benefit in bringing ontology development into its scope. Essentially this means creating definitions of ontology

modelling languages (metamodels) in terms of the OMG's MOF model (OMG, 2002b)—a simplified version of UML used as a meta-meta-modelling language—and then defining transformations between these different languages where possible. Ontologies could then be stored in MOF-based repository tools such as the NetBeans Metadata Repository (NetBeans.org, 2002), imported and exported to and from these tools in an XML format using their existing support for the XMI standard, examined or modified using standard application programmer interfaces such as the Java Metadata Interface (JMI) (Java Community Process, 2002), and transformed using the QVT language (OMG, 2005). This would greatly increase the interoperability of ontology development and repository tools, would enable UML editors to be used to develop ontologies (using appropriate UML profiles), and would bring ontology development into the same tool environment as other model-based development undertaken within an organisation (e.g. traditional software engineering) (Chang and Kendall, 2004). Colomb et al. (2006) present the following scenario to illustrate the benefits of an MDA approach to ontology engineering:

> [C]onsider a large e-commerce exchange project. The developers might choose to represent the ontology specifying the shared world governing the exchange in OWL. But the exchange may have evolved from a single company's electronic procurement system … The original procurement system might have been designed using UML, so that it would be a significant saving in development cost to be able to translate the UML specification to OWL as a base for development of the ontology.

In response to an OMG request for proposals (OMG, 2003a), the OMG's Ontology Definition Metamodel (ODM) working group is developing MOF-based metamodels for a number of ontology modelling languages (topic maps, Simple Common Logic, RDF Schema and OWL) and transformations between them (Sandpiper Software, 2006). However, this is still work in progress, and it will depend on the completion of the QVT language, which is also still under development.

This paper describes a different approach to solving the same problem using existing technology, as shown in Figure 1. Using a MOF-based repository (the NetBeans Metadata Repository) and the Java Metadata Interface, we have implemented software that can convert any model defined using a MOF-based language to and from an equivalent model encoded in RDF. This conversion does not involve any loss of information. In particular, we do not convert the model to an RDF schema. Instead, each model element is represented by a resource in the RDF model that has a type declaration referring to that element's metaclass in the metamodel. Properties of the model elements and relationships between them are represented by RDF statements that refer to the properties and relationship types in the metamodel. In other words, we treat the metamodel as an RDF schema[1], automatically generating URLs that correspond to its classes and properties. Once the model is in this RDF format, it can be transformed using RDF transformation tools, either to modify the model without changing its metamodel, or to transform it to a representation using a different metamodel (e.g. to map an ontology defined in UML to an RDF Schema or OWL representation). Currently we use the Jena rule language (HP Labs, 2002) to transform RDF models, but other transformation languages could also be supported (Steer, 2003; Walsh, 2003; Prud'hommeaux and Seaborne, 2005).

Essentially this work demonstrates the advantages of RDF as an export format for models (and, in particular, ontologies) that have been represented in a MOF-based language, e.g. for ontologies defined in UML or, once the ODM work is finished, ontologies in any of the ontology modelling

[1] Although the RDF schema corresponding to the metamodel could be explicitly generated, we do not currently do this. The MOF repository requires an explicit representation of the metamodel in the XMI format and we use that as the definitive source of metamodel information.

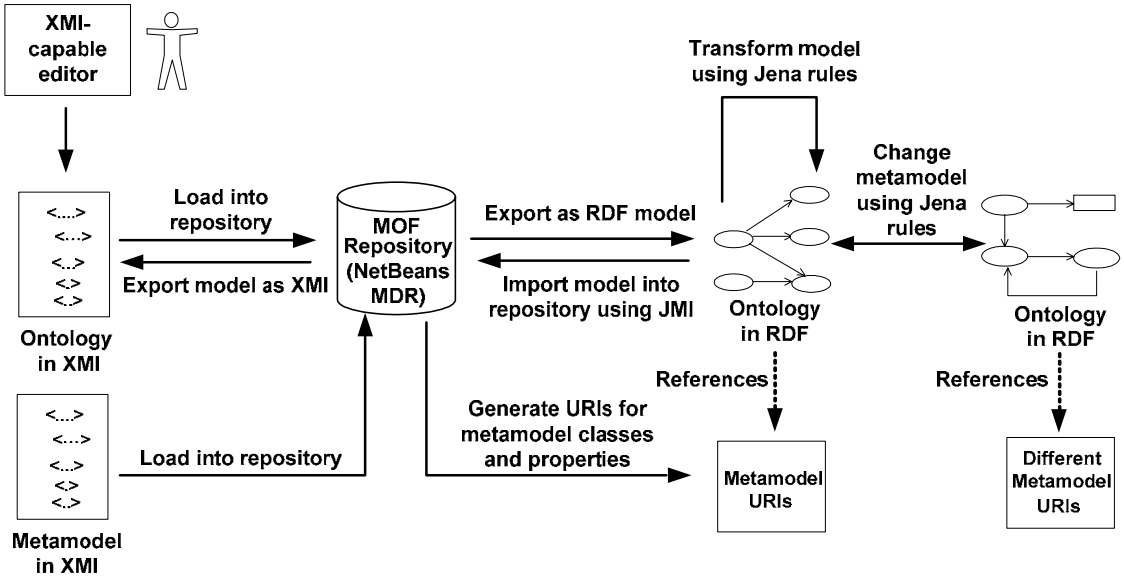

Figure 1. Overview of our approach to MDA-compliant ontology analysis and transformation

languages that have been defined in terms of the MOF model. Our previous work on transforming ontologies in UML (Cranefield, 2001) has focused on processing XMI files using the XSLT language (Clark, 1999). It is now our belief that XMI is too low level a representation for conveniently processing high-level models such as ontologies, and that the RDF data model provides a more suitable abstraction. The completion of the QVT language and the availability of robust implementations of it will provide a similarly high level way to query and transform ontologies developed using MDA tools, but we believe the simplicity and familiarity (to ontology developers) of the RDF data model and the available tools, e.g. the Jena rule language and the SPARQL query language (Prud'hommeaux and Seaborne, 2005) with its basis in SQL, hold significant advantages.

The structure of this paper is as follows: Section 2 provides an overview of the concept of metamodelling and the OMG's Meta-Object Facility, and Section 3 briefly discusses the Semantic Web standards and tools used in this work. The details of our mapping from the MOF's abstract information model to an equivalent representation using RDF are presented in Section 4. Section 5 describes how models can be converted between XMI files and our RDF representation using a MOF-based model repository and the Java Metadata Interface. Section 6 discusses the use of the RDF representation as a basis for querying and transforming ontologies. Section 7 discusses related work and Section 8 concludes the paper.

2 OMG MDA TECHNOLOGIES

In order to use ontologies (or any models) with MDA-based tools, it is first necessary to understand the concept of metamodelling and the structure of a MOF-based model repository. This section provides an overview of these concepts, and shows how our tool to convert between XMI and RDF representations of an ontology can work regardless of the language (i.e. metamodel) in which the ontology is represented—provided that the metamodel is defined in terms of the MOF model. We illustrate this for a simple model in UML 1.3.

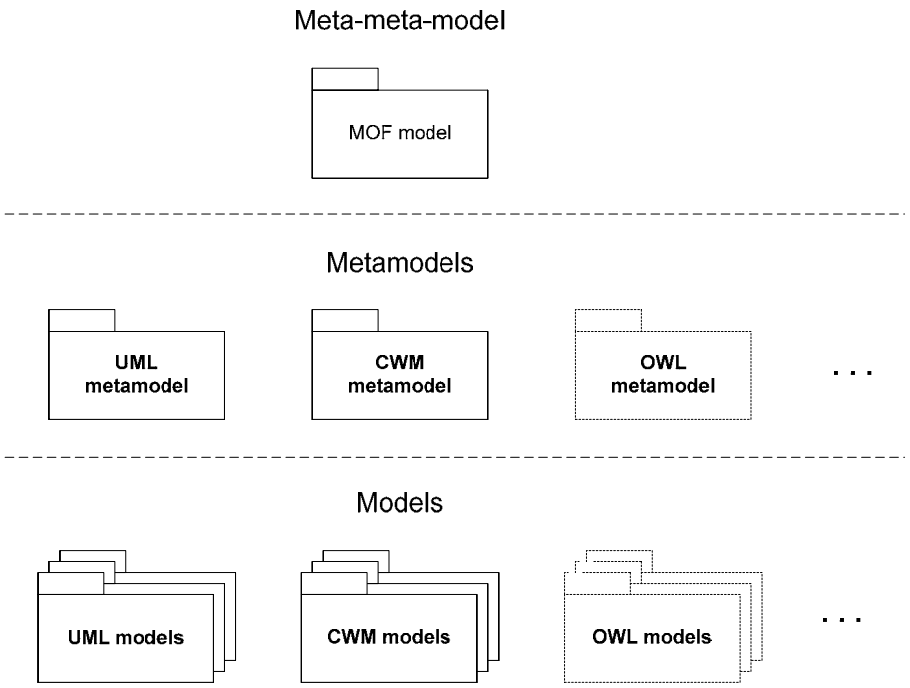

Figure 2. The MOF metamodelling hierarchy

2.1 Metamodelling and the Meta-Object Facility

The model-driven architecture is based on the concept of metamodelling. Just as an ontology can be considered to be a model defined in some modelling language, a modelling language can also be defined by a (meta)model that is expressed in another language—a metamodelling language. For example, UML is defined by a metamodel that contains (meta)classes such as Class, Association and AssociationEnd, and (meta)associations that define how instances of these can be related to each other (e.g. an association has two or more association ends). The OMG has defined a metamodelling language called the MOF model (OMG, 2002b) that is used as the basis for the model-driven architecture. To enable a modelling language to be used with MDA tools it must be given a definition in terms of the MOF model.

Figure 2 illustrates the metamodelling hierarchy underpinning the MDA. This shows how the MOF model is the metamodel for various modelling languages. Two of the key OMG metamodels, UML and the Common Warehouse Metamodel (CWM, for data warehousing metadata) are shown in the figure, along with a metamodel for the Web Ontology Language, OWL, which is under development by OMG's Ontology Definition Metamodel group. Various other metamodels have also been defined or are under development by the OMG[2]. Each of these metamodels defines the modelling concepts that can be used to create models using a particular language.

[2] These include the Software Process Engineering Metamodel, the Information Management Metamodel (an update of the CWM), the Business Process Definition Metamodel, the Organization Structure Metamodel, the Abstract Syntax Tree Metamodel, the Knowledge Discovery Metamodel, the Spacecraft Operations Language Metamodel, the Topic Maps Metamodel, the Simple Common Logic Metamodel, a generic Description Logic Metamodel, and the RDF Schema Metamodel.

A metamodel defined in terms of the MOF model[3] contains definitions of classes and data types, possibly organised into a nested package structure. A data type can be a primitive type, an enumeration type (having a finite set of named values), a structure type (having named and typed fields), a collection type (having a base type and a multiplicity) or an alias type (a named subtype of an existing type). Classes may have attributes and operations declared, with the operations having typed in, out, inout and return parameters, and possibly exceptions, which can be defined within the scope of classes and packages. Class attributes are specified by a name, a type, a multiplicity, a scope (either instance or classifier level—indicating whether each instance of the class may have a separate value for the attribute), and an *isDerived* flag declaring whether the attribute value can be computed from other information in the model[4]. The multiplicity is a structure comprising lower and upper cardinality bounds, an *isUnique* flag indicating whether a multi-valued attribute may have duplicate values, and an *isOrdered* flag indicating whether the ordering of multiple values is significant.

The MOF model is basically a simplified version of UML, and the UML notation can be used to depict metamodels defined in terms of it. Figure 3 shows a subset of the UML 1.3 *physical metamodel* (its definition in terms of the MOF model). This differs slightly from the metamodel diagrams in the main sections of the UML specification in order to fit the restrictions of the MOF model. In the figure, derived attributes (those with their names prefixed by '/') are used to indicate MOF *references*. In the MOF model, associations represent a query-oriented interface for discovering and modifying links between objects. They are implemented in a MOF-based model repository by objects that represent sets of links. If direct navigation is required from instances of a given class to the associated objects at the other end of an association, this can be provided by defining a *reference* that provides an attribute-like view of that association end.

In addition to a metamodel, a modelling language has a graphical or lexical notation for expressing models in that language. For example, Figure 4 shows a simple UML class diagram depicting a single class. However, the same information can also be represented as a set of instances of classes in the metamodel and links between them that are instances of associations in the metamodel. Figure 5 shows this representation of the UML model from Figure 4. This instance-based view is how models are stored internally by tools based on the MOF.

2.2 The MOF Abstract and IDL Mappings

The MOF *abstract mapping* defines how a model is structured as a set of instances of the metamodel's classes. These instances are contained within logical domains called *extents* and inter-related by links (instances of associations, which are also contained within extents) and reference values. The abstract mapping also defines a standard set of "technology neutral" data types: the primitive types Boolean, Integer, Long, Float, Double, and String, and the enumeration types, collection types, structure types and alias types produced recursively from these.

There are three types of extent. For each class in the metamodel there is a *class extent* that contains all instances of that (meta)class that occur in the model; for each association in the metamodel, there is an *association extent* that contains all instances of that association (i.e. links) that occur in the model; and for each package in the metamodel there is a *package extent* that contains the extents for any packages, classes and associations declared within that package.

The abstract mapping does not define any operations on instances and links and the extents that contain them, but instead provides "a style guide for metadata computational semantics" with conventions such as "access operations should not change metadata". To provide a way for client

[3] This research discussed in this paper is based on version 1.4 of the MOF.

[4] There is also a *visibility* flag, but the MOF 1.4 specification reserves this for future use, with its meaning undefined.

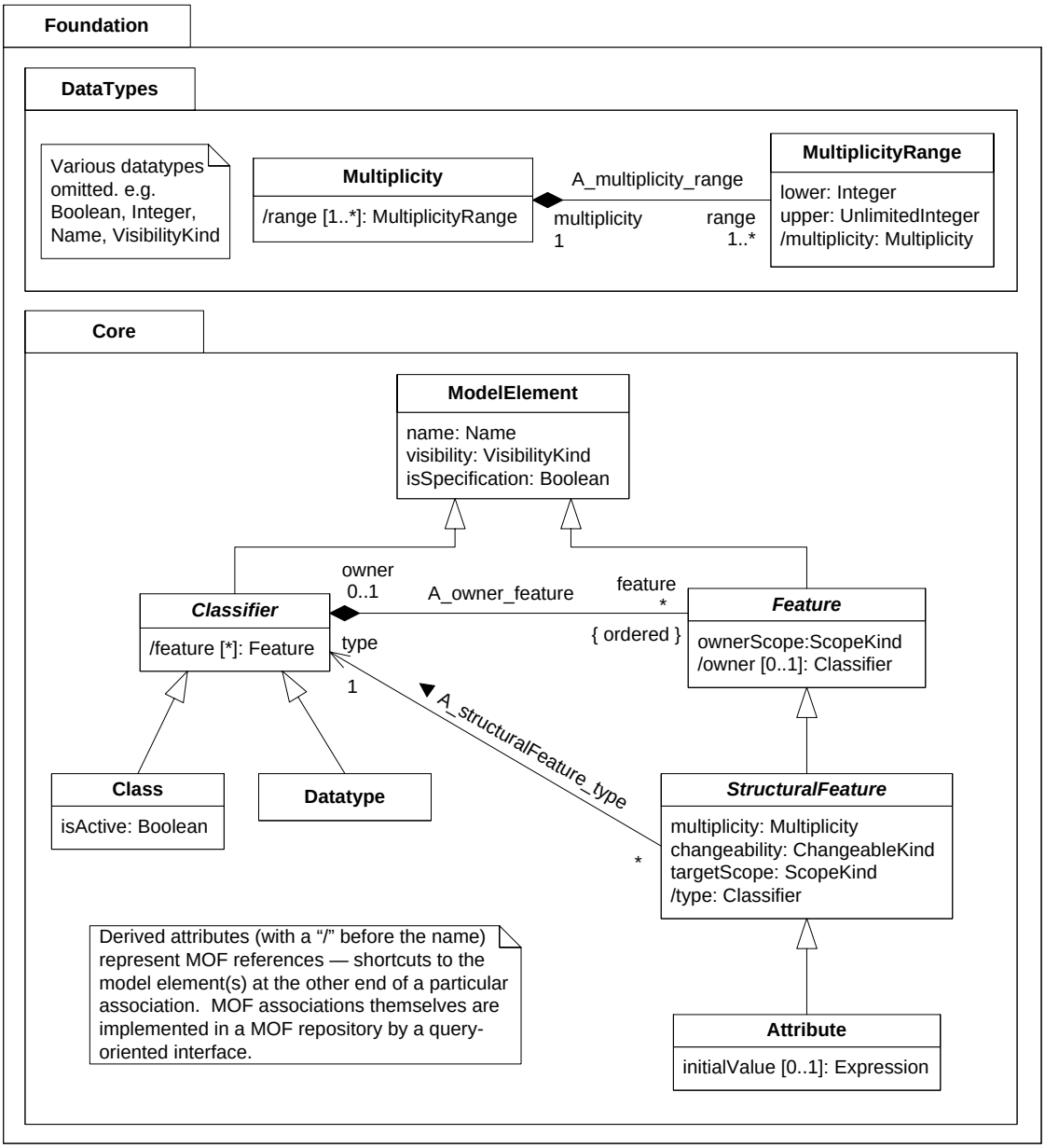

Figure 3. A subset of the UML 1.3 physical metamodel

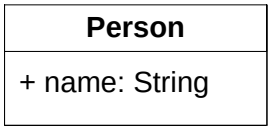

Figure 4. A simple UML model

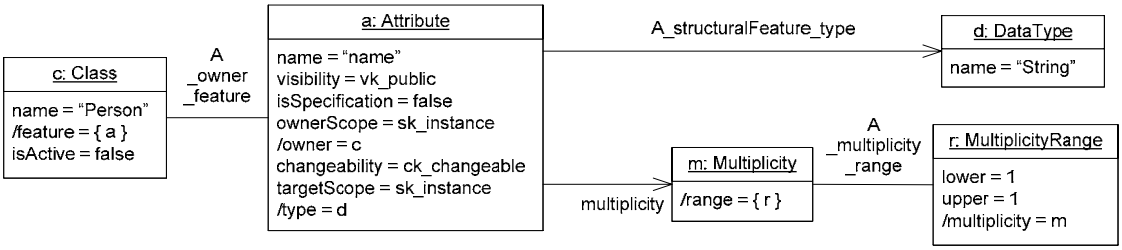

Figure 5. The model from Figure 4 as an instance of the UML physical metamodel

programs to create, inspect and modify models stored in a MOF-based tool such as a model repository, the MOF specification defines the MOF *IDL mapping*. Given a metamodel defined in terms of the MOF model, this mapping uses the CORBA middleware's interface definition language (IDL) to specify a set of interfaces that correspond to the extents of packages, classes and associations in the metamodel and the instances within them, and these provide metamodel-specific operations for accessing models. The MOF IDL mapping also defines a *reflective module*, which defines a set of interfaces that can be used by clients to access a model without compile-time knowledge of the metamodel used.

2.3 The Java Metadata Interface (JMI)

The MOF IDL mapping, in conjunction with the OMG's IDL to Java mapping, provides a mechanism for Java clients to access models in a MOF-based repository. However, the Java community considered that the combination of these two mappings resulted in a style of code that was unintuitive to Java programmers, included CORBA implementation-specific features, and did not take advantage of advanced Java features. A Java Specification Request was submitted to the Java Community Process, and the Java Metadata Interface (JMI) specification was subsequently produced (Java Community Process, 2002). JMI provides a Java mapping for the MOF that is a community standard. Like the MOF IDL mapping, it defines both a mechanism for generating metamodel-specific APIs as well as a generic reflective API.

In this research we have used the NetBeans Metadata Repository (MDR) (NetBeans.org, 2002) as our model repository. This is an add-on to the popular NetBeans integrated development environment. It can read and write models in the XMI format and provides access to them and the ability to create and modify models from Java using JMI.

3 SEMANTIC WEB TECHNOLOGIES

This section gives a brief overview of the Semantic Web standards and tools used in this research.

3.1 RDF, RDF Schema and OWL

The World Wide Web Consortium has developed the Resource Description Framework (RDF) (Manola and Miller, 2004) as the data model on which the Semantic Web is based. Based on the idea that any "resource" (whether online or offline, concrete or abstract) can be referenced by a uniform resource identifier (URI), information is represented in an *RDF model* as a set of ⟨subject, predicate, object⟩ triples (also known as statements), where each subject is a resource that is either identified by a URI[5] or is a *blank node* (discussed below), each predicate is a resource identified by a URI, and each object is either a resource (of either type: a blank node or one identified by a URI) or a literal value. The RDF model forms a directed graph that encodes

[5] More accurately, RDF uses *URI references* as identifiers. These are URIs that are possibly extended by an optional fragment identifier, using the syntax "URI#fragment-id"

information about resources and their properties and relationships. A blank node is a resource that has an intrinsic identity within a given model but has no globally unique identifier. Although any particular RDF model serialisation syntax may assign identifiers to blank nodes to allow these to appear as the objects of statements, these identifiers have no significance except to distinguish one blank node from another.

The use of URIs to identify both domain entities and the predicates used to describe them enables automated processes to have an unambiguous interpretation of RDF data that is published on the Web, provided that those processes are designed with a built-in understanding of the URIs used to encode concepts and properties in the problem domain, or have the capability of learning their meanings. To allow communities to define and share sets of terminologies, the RDF Schema language (Brickley and Guha, 2004) and its extension, the Web Ontology Language (OWL) (McGuinness and van Harmelen, 2004) were developed.

RDF Schema is a set of predefined resources and relationships between them that define a simple metamodel, including the concepts of primitive "literal" types, classes, properties, subclass properties of classes, and subproperty, domain and range (meta)properties of properties. Domain schemas (i.e. ontologies) can then be expressed as sets of RDF triples that reference the (meta)classes and properties defined in RDF Schema.

The Web Ontology language (OWL) is an extension of RDF Schema with semantics based on constructs from description logic, allowing classes to be defined in terms of other classes, and properties to have constraints placed upon them, such as restrictions on the number of values. This enables some forms of deduction to be made to infer additional information from known instance information in conjunction with an ontology, e.g., the ontological information that a serial number is unique to a gun is crucial in matching a particular gun to its registered owner (Costello, 2003).

3.2 The Jena Framework

Jena (HP Labs, 2002) is a Java framework for building Semantic Web applications. It provides a programmatic environment for handling RDF, RDF Schema and OWL data, including a rule-based inference engine with forward and backward chaining capabilities. Jena provides an API for writing and reading serialised RDF models and navigating, querying and modifying models. It has a large user community and is very well supported by the Semantic Web research group at HP Laboratories, Bristol, UK. The Jena rule language is of particular note for this research as it provides a declarative way of specifying transformations on ontologies, once they have been converted to an RDF representation.

4 MAPPING MOF-BASED MODELS TO RDF

This section discusses the details of our mapping from a model expressed in a MOF-based language to an equivalent representation using RDF. The motivation for defining this mapping is to provide a representation for MOF-based models using a graph-structured information model, rather than the tree-structured XMI representation. This allows transformations on models to be defined at a more abstract level than is possible using XSLT. This issue is discussed further in Section 7.

In previous work (Cranefield, 2001) we have proposed generating RDF schemas from ontologies modelled in UML, in order to provide a convenient RDF-based serialisation format for information expressed in terms of those ontologies. This is not our purpose here. A serialisation format for instance information does not need to encode all details of an ontology, so a relatively simple transformation from UML to RDF Schema was sufficient. For the current work, our aim is to generate a complete encoding of an ontology as an RDF model. This means that instead of

using RDF Schema as the type system for our RDF model, we generate URIs that correspond to the classes, associations, attributes, references and data types in the metamodel and use these as property and class resources in the RDF representation of the ontology.

4.1 The RDF Mapping

In a MOF-based repository, each element of a model is represented as an instance of its metaclass (as shown in Figure 5). In most cases these map directly to RDF resources (the exception being metaclasses representing primitive data types, discussed in Section 4.2). The URI for each of these resources is a combination of an XML namespace for the model (provided by the user and ending in '#' by convention) and a local name that is generated from the object's unique MOF identifier (which any MOF implementation is required to provide). The type of the model element is then recorded in the RDF model by a statement of the following form[6]:

(m:MOF_ID rdf:type mm:QualifiedMetaclassName)

where m is the abbreviation for an XML namespace chosen for the model, mm abbreviates a namespace uniquely associated with the metamodel and QualifiedMetaclassName is the fully qualified name of the model element's metaclass, e.g. Foundation.Core.Class for the UML concept of a class. See lines 1, 8, 13 and 19 of Listing 1 for example type declarations of this form.

Each model element will also have values for the attributes and references that are defined for its metaclass. Finding the metaclass's attributes and references and their values for a particular model element involves using an application programming interface such as the MOF IDL mapping or JMI to examine the model and its metamodel. This process is described (in terms of JMI) in Section 5.

For each of the metaclass's attributes and references, a statement of the following form is asserted into the RDF model to represent its value(s) for the given model element:

(m:MOF_ID mm:QualifiedPropertyName ValueResourceOrLiteral)

where QualifiedPropertyName is the fully qualified name of the attribute or reference in the metamodel, e.g. Foundation.Core.ModelElement.name for the name of a UML model element.

The encoding of ValueResourceOrLiteral depends on the multiplicity and type of the attribute or reference. Multiplicities are partitioned into three cases: single-valued (a multiplicity of 1..1), optional (a multiplicity of 0..1) or multi-valued (all other multiplicities). To illustrate the different cases, a partial RDF representation of Figure 5 is shown in Listing 1. For brevity we only show a few of the attribute values. The resource local names ID_c, ID_a, ID_m and ID_r represent the MOF repository identifiers for the model elements labelled c, a, m and r (respectively) in Figure 5, and ID_x, ID_y and ID_z represent the MOF repository identifiers for the association extents of the associations instantiated in the figure. RDF blank node identifiers are prefixed by "#" and we use the notation *value^^type* to indicate typed literals.

For metamodel attributes and references that are single valued, the representation of a value in the RDF model depends on whether the type of the attribute or reference is a metaclass, a primitive type, an enumeration type or a structure type[7]. Metaclass instances map directly to RDF

[6] In this paper we use the statement syntax from the Jena rule language for expressing RDF triples.

[7] MOF alias types are not handled as the implementation technology used (JMI) maps alias types to their base type.

```
1.  (m:ID_c  rdf:type  mm:Foundation.Core.Class)
2.  (m:ID_c  mm:Foundation.Core.ModelElement.name  'Person'^^xsd:string)
3.  (m:ID_c  mm:Foundation.Core.Classifier.feature  #A0)
4.
5.  (#A0  rdf:type  rdfs:Container)
6.  (#A0  rdfs:member  m:ID_a)
7.
8.  (m:ID_a  rdf:type  mm:Foundation.Core.Attribute)
9.  (m:ID_a  mm:Foundation.Core.ModelElement.visibility
10.             'vk_public'^^mm:Foundation.Data_Types.VisibilityKind)
11. (m:ID_a  mm:Foundation.Core.StructuralFeature.multiplicity  m:ID_m)
12.
13. (m:ID_m  rdf:type  mm:Foundation.Data_Types.Multiplicity)
14. (m:ID_m  mm:Foundation.Data_Types.Multiplicity.range  #A1)
15.
16. (#A1  rdf:type  rdfs:Container)
17. (#A1  rdfs:member  m:ID_r)
18.
19. (m:ID_r  rdf:type  mm:Foundation.Data_Types.MultiplicityRange)
20. (m:ID_r  mm:Foundation.Data_Types.MultiplicityRange.multiplicity  m:ID_m)
21. (m:ID_r  mm:Foundation.Data_Types.MultiplicityRange.lower  '1'^^xsd:int)
22. (m:ID_r  mm:Foundation.Data_Types.MultiplicityRange.upper  '1'^^xsd:int)
23.
24. (m:ID_x  rdf:type  mm:Foundation.Core.A_owner_feature)
25. (m:ID_x  rdf:type  mmx:Association)
26. (m:ID_x  rdfs:member  #A2)
27.
28. (#A2  rdf:type  rdf:Seq)
29. (#A2  rdf:_1  m:ID_c)
30. (#A2  rdf:_2  m:ID_a)
31.
32. (m:ID_y  rdf:type  mm:Foundation.Core.A_structuralFeature_type)
33. (m:ID_y  rdf:type  mmx:Association)
34. (m:ID_y  rdfs:member  #A3)
35.
36. (#A3  rdf:type  rdf:Seq)
37. (#A3  rdf:_1  m:ID_a)
38. (#A3  rdf:_2  xsd:string)
39.
40. (m:ID_z  rdf:type  mm:Foundation.Data_Types.A_multiplicity_range)
41. (m:ID_z  rdf:type  mmx:Association)
42. (m:ID_z  rdfs:member  #A4)
43.
44. (#A4  rdf:type  rdf:Seq)
45. (#A4  rdf:_1  m:ID_m)
46. (#A4  rdf:_2  m:ID_r)
```

Listing 1. A portion of the RDF model corresponding to Figure 5

resources, using the chosen model namespace with the MOF repository ID as the local name (see line 11 in the listing). Values of primitive types are mapped directly to RDF typed literals using XML Schema types (see line 2). Enumerated type values are also represented as typed literals where the type URI is the metamodel namespace together with the type's qualified name in the metamodel (see lines 9 and 10). Structure type values are encoded as blank nodes. The type of the value is asserted by an rdf:type statement with the blank node as the subject and the object being a URI constructed from the metamodel namespace and the fully qualified name of the structure type in the metamodel. The values of the structure's fields are recorded by additional RDF statements, in a similar way to the representation of object attribute values. These statements use predicate URIs constructed from the metamodel namespace and the fully qualified name of the structure field from the metamodel. The example does not illustrate this case as the UML 1.3 metamodel does not use structure types.

In the multi-valued case the object of the RDF statement is a blank node with type rdfs:Container. The container resource is then related to its contents—the individual values for the attribute or reference—by the rdfs:member property. Each individual value is then encoded as for the single-valued case. This case is illustrated in lines 14–17 of Listing 1.

The optional case includes two sub-cases: a multiplicity of 0..1 and a multiplicity of 0..n where n > 1 or n = -1 (indicating an unbounded multiplicity, shown as '*' in Figure 3). When the attribute or reference has a value, the encoding is the same as for the single-valued and multi-valued cases, respectively (e.g. see lines 3–6). When it does not have a value, the first sub-case is encoded by omitting the RDF triple that represents the attribute or reference value. For the second sub-case, the attribute or reference value is represented by a blank node of type rdfs:Container (as for the multi-valued case), but no rdfs:member triples are asserted.

Associations are represented in a MOF model repository by association extent objects, which represent sets of links. These are encoded in the RDF model by resources that represent these sets of links, with their URIs formed using the model namespace and the extent's MOF repository ID. Each link set resource has an rdf:type property that identifies the association in the metamodel (e.g. mm:Foundation.Core.A_owner_feature). Although we currently don't explicitly generate an RDF schema corresponding to the metamodel, we consider the metamodel association resources to be subclasses of rdfs:Container, and therefore use rdfs:member statements to represent the links that instantiate the association. Each link is represented as a resource of type rdf:Seq (a sequence) with two elements[8]: the resources at each end of the link. Finally, to make it easy to tell these link-set objects apart from sets of multi-valued reference values, we add a statement asserting that the link-set resource has type mmx:Association, where mmx abbreviates a special namespace we reserve for metamodel-related properties that are not a direct representation of existing properties in the metamodel. Effectively this means that we are treating all associations in the metamodel as subclasses of an abstract base class mmx:Association, but to avoid the need for inference using an explicit RDF schema representation of the metamodel, we explicitly include this extra rdf:type triple in the model. Lines 24–30 of the listing show a link set corresponding to the UML metamodel's association between classes and their attributes.

4.2 Mapping Data Types

Models generally need to include a set of data types that are used elsewhere in the model, e.g. note the reference to the data type String in Figure 5. To avoid a proliferation of different versions of the same data types from different models, it is desirable to map these to a single common representation in the RDF representation: XML schema data types (as commonly used

[8] Associations in the MOF model can only have two ends.

for RDF typed literals). This can be done for UML by detecting instances of the DataType metaclass. However, to avoid having any hard-coded knowledge about particular metamodels in the RDF mapping program, the metaclasses representing the concept of a data type in various metamodels and their associated type mapping rules are specified in a properties file that is read by the program. Line 38 in the listing shows the result of this mapping for the example: the object of that triple is a resource representing the XML Schema string type.

4.3 Current Limitations

The MOF RDF mapping described above does not address all possible features of a MOF-based metamodel. Values of alias types are not handled because our implementation technology (JMI) silently maps these to their base types. Collection types are also not handled. Correctly handling collection types requires allowing collections to appear as the values of object attributes and structure fields, and as the base data type for other collections. This is left for future work, but we note that the latest version of the MOF, version 2.0, does not include collection types.

Classifier-scoped metaclass attributes are currently not handled specially. At present, for each instance of a metaclass the mapping records the values for all of the metaclass's attributes (including classifier-scoped ones). If there is more than one instance of a given metaclass, the values for any classifier-scoped attributes will be recorded multiple times, and if there are no instances of a metaclass, then the values of classifier-scoped attributes will not be recorded.

The mapping does not take into account the value of the *isOrdered* and *isUnique* flags in the multiplicities of attributes, references and association ends in the metamodel. Handling all four possible value pairs for these flags requires replacing rdf:Container resources (used to represent sets of values) with resources of type rdf:Bag, rdf:Seq (or alternatively rdf:List), or some custom type representing an ordered set, where appropriate. However, writing declarative transformation rules to process all values of a multi-valued attribute or reference would then be more complex than the current approach, as different rule patterns would be needed for different cases. This issue is left as a subject for further research.

These limitations may not be an issue for many metamodels. For example, the UML 1.3 metamodel definition does not use MOF alias or collection types, classifier-scoped attributes or references, or multi-valued attributes or references with *isUnique* set to *false*. Although UML 1.3 includes some multiplicities with *isOrdered* set to true, for applications involving class diagrams without parameterised classes or association end qualifiers, the only information that might be lost when using the RDF mapping outlined above would be the order in which attributes are listed within a class.

4.4 Summary

In this section we have presented a translation from the MOF abstract mapping to a representation in RDF. Our aim is to provide a RDF-based serialisation format for ontologies defined using languages with MOF-based metamodels. The resulting RDF models can then be transformed using RDF processing tools, as discussed in Section 6, and then imported back into MOF-based model repositories. However, it is important to note that the RDF mapping does not involve converting a model (or, more specifically, an ontology) to an RDF schema, which would require translating the model to a different metamodel. Instead, we are using RDF as a format for representing the model elements as instances of the types defined in the metamodel. By treating the elements of the metamodel as resources (with automatically generated URIs) that can appear as the objects of rdf:type statements, RDF can be used to represent models in any MOF-based language without any loss of information. The models can then be analysed or transformed using RDF tools.

Although we generate URIs for resources representing elements of the metamodel, we have not attempted to generate an RDF schema or OWL ontology that represents the full structure of the metamodel. Instead we assume that the XMI version of the metamodel will remain as the definitive source of this information.

5 TRANSLATING BETWEEN XMI AND RDF USING JMI

The previous section defined an RDF representation for each type of element in a MOF-based model. To produce the complete RDF representation of a model, it is necessary to systematically examine the model to discover all its elements and their relationships, and then express all of this information using the RDF mapping. It is also necessary to be able to import a model encoded using the RDF mapping back into a repository, from which it can be exported as an XMI file. In this section we discuss how these export and import processes can be performed using the JMI API. While this discussion includes details specific to JMI, the processes can be performed in a very similar way by a client using the MOF IDL mapping[9].

5.1 Exporting a Model using the RDF Mapping

In order to examine a model, the metamodel and then the model must be loaded into a MOF-based repository (this is usually done by importing XMI files). It is then possible to use the JMI reflective interface (which makes no assumptions about the metamodel used) to traverse the repository's nested extent structure and find all model elements. Figure 6 gives an overview of the JMI view of the repository data.

The top half of Figure 6 illustrates the representation within the repository of the model from Figure 5 in terms of the interfaces of the JMI reflective API. The objects and links from Figure 5 are stored in the repository as instances of the RefObject and RefAssociationLink interfaces (the bottom row of objects in the top half of Figure 6). They can be discovered using the reflective API by recursively searching through a set of package, class and association extent objects. There is one extent for each of the packages, classes and associations in the metamodel, and they are nested according to the package structure of the metamodel. The reflective API provides operations to return all nested package, class and association extents for a given package extent, all objects in a given class extent, and all links in a given association extent.

The API also allows the values of the (metamodel-level) attributes and references of a RefObject to be obtained, e.g. the value "visibility = vk_public" in Figure 5. However, the valid attributes and references for that type of object in the metamodel must be known in order to do this. These can be found by navigating from a RefObject in the model to the object representing its class in the metamodel using the refMetaObject() operation, and examining the structure of that class. The lower half of Figure 6 shows the representation of the metamodel in the repository. The JMI reflective API could be used to examine the metamodel; however, as it is known that the metamodel (for whatever modelling language is used) is defined in terms of the MOF model, it is more convenient to use a specialised JMI API for examining instances of the MOF model. The structure of the metamodel within the repository is also based around extents, but this time they reflect the structure of the packages and classes in the MOF model. In the MOF model, there is an association called "Contains" that is used in metamodel definitions to link packages to the packages, classes and associations they contain, and classes to the attributes and references they "contain". The Contains association object in the metamodel can be used to find this information, but there is also a convenient getContents() operation in the MofPackage and MofClass interfaces

[9] The main difference between the MOF IDL mapping and JMI is that the IDL mapping's RefObject interface is used to represent both objects and class extents, whereas JMI uses separate RefObject and RefClass interfaces.

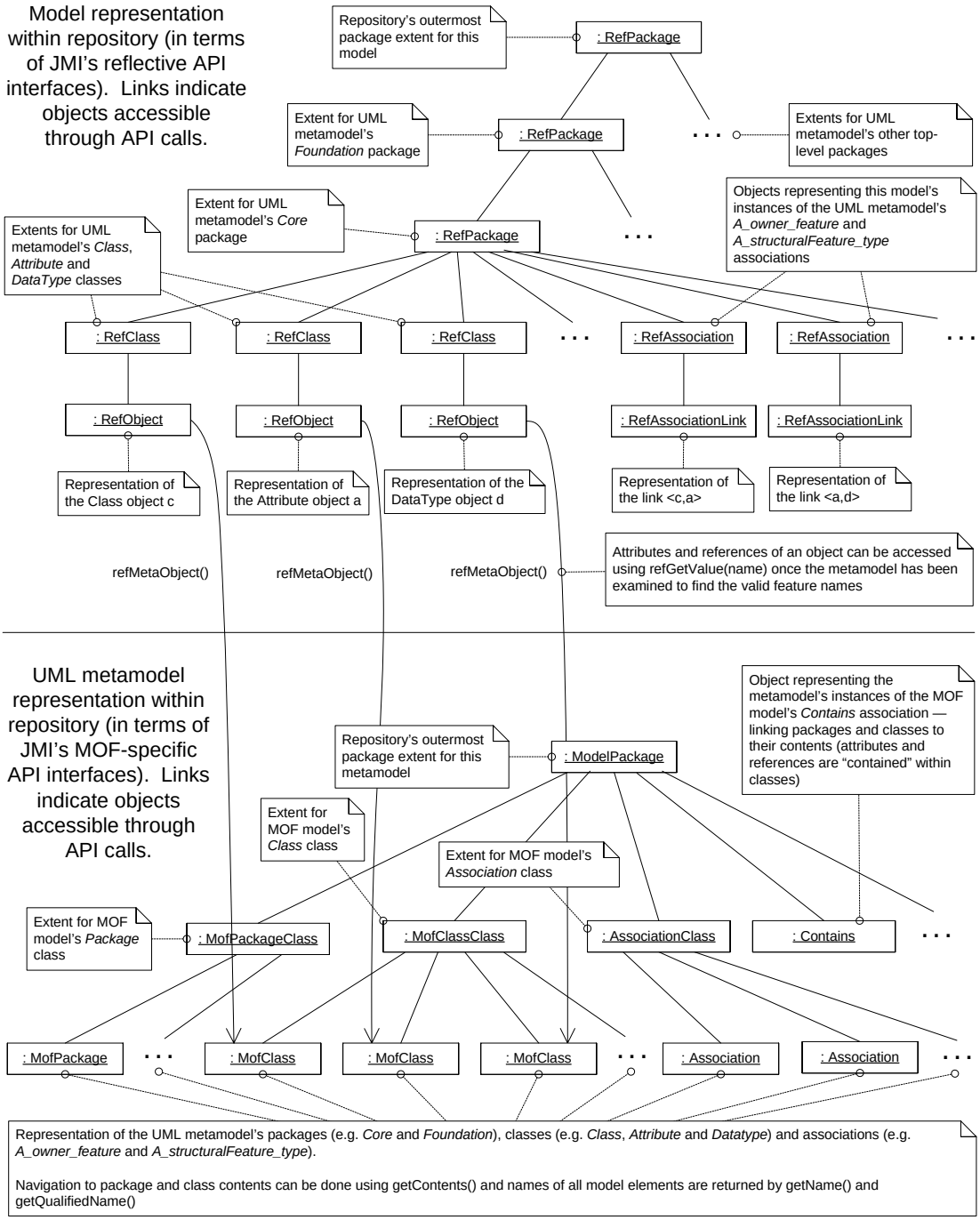

Figure 6. Representation of a model and its metamodel within the repository

(which represent packages and classes in the metamodel, respectively). The methods getName() and getQualifiedName() can then be used to find the names of metaclasses and their attributes and references, either with or without package qualifiers, respectively.

It is important to note that this analysis of a model using JMI can be done without any prior knowledge of the metamodel used. The RDF mapping discussed in Section 4 can therefore be performed for any modelling language that has a definition in terms of the MOF model.

5.2 Importing a Model in the RDF Representation

Once a model has been exported from a repository in the RDF format, it can be transformed using RDF tools. For that to be useful, we need to be able to load the resulting RDF models back into a MOF-compatible repository, which can then be used to provide programmatic access to the transformed models via JMI or to export them as XMI files.

The JMI reflective API can be used to populate an initially empty model with the model elements found in an RDF model, provided that the metamodel has already been loaded into the repository. The importation process begins by locating all class and package extents and storing them in a map data structure that associates them with the URI for the metamodel element they represent. It is necessary to access the metamodel using refMetaObject() to find the name of the metamodel element corresponding to each extent. The RDF model can then be examined to find all resources, and the type URIs for these are used to look up the corresponding extent. An appropriate method can then be called on the extent to create a new data type or class instance or a link. Creating values of structure types also requires accessing the metamodel in order to find the order of the structure fields (this is not included in the RDF mapping, which uses properties based on the field names). This additional information is necessary because the JMI method for creating structure type values takes the field values as a list parameter.

Just as a tool that imports XMI files can only handle XML files that conform to the XMI specification, model importation from RDF can only be performed for a subset of RDF models: those using the modelling conventions defined in Section 4, and which refer to classes and properties from a metamodel known to the tool (identified by its namespace or specified by the user).

6 ANALYSING AND TRANSFORMING ONTOLOGIES IN RDF

Our aim in this work was to generate a serialisation format for ontologies in MOF-based languages that provides a suitable level of abstraction for analysis and transformation, and which can be imported and exported into MOF-based repositories. For querying an ontology in RDF, the best tool is probably the SPARQL query language (Prud'hommeaux and Seaborne, 2005). In this work we have concentrated on the transformation of ontologies, and we have chosen to use the Jena rule language. We believe that for ease of understanding and maintenance, transformations should be specified in a declarative language, and the Rete-based production system (Forgy, 1982) implemented by Jena's forward-chaining inference engine is a well known model of rule execution due to the popularity of the CLIPS and JESS expert system shells.

We envisage two types of transformation being used with ontologies: transformations of models without changing the metamodel, and transformations of models between different metamodels. The latter type of transformation is what the final ODM specification will define: transformations that will allow ontologies and other types of domain model to be exchanged between tools that are based on different modelling languages. An example of the former is the generation of an ontology-specific content language for agent communication, given a specific ontology as input. It is common practice amongst users of the FIPA agent communication language to include object-like structures (encoded in the FIPA SL content language) within messages, with the

structure of these objects based on the structure of classes in the ontology, e.g. to express information using a functional term such as `(car :number-plate AAA1234 :colour blue)`. The most coherent explanation of this usage is that this term does not represent an instance of an ontological concept (you can't send cars within messages!), but rather represents an expression representing a proposition about a particular car in an ontology-specific content language. This content language can be derived from an ontology by a standard transformation. Listing 2 presents two Jena rules that implement part of this transformation, given as input the RDF representation of an ontology encoded in a particular UML profile (Cranefield and Purvis, 2002).

```
[rename_class:
     (?class rdf:type mm:Foundation.Core.Class),
     (?stereotype
       rdf:type
       mm:Foundation.Extension_Mechanisms.Stereotype),
     (?stereotype
       mm:Foundation.Core.ModelElement.name
       'resourceType'),
     (?stereotype
       mm:Foundation.Extension_Mechanisms.Stereotype.extendedElement
       ?container),
     (?container rdfs:member ?class),
     (?class mm:Foundation.Core.ModelElement.name ?classname),
     concat(?classname, 'Proposition', ?newclassname)
 ->
     // Delete triple declaring old name of class
     // (identified by LHS clause index, starting at 0)
     drop(5),
     // Assert new class name
     (?class mm:Foundation.Core.ModelElement.name ?newclassname),
     // Assert that ?stereotype and ?container should be deleted
     (?stereotype temp:to_be_deleted stereotype(?container))
]

[delete_stereotype:
     (?stereotype temp:to_be_deleted stereotype(?container1)),
     (?association1
       rdf:type
       mm:Foundation.Extension_Mechanisms.A_stereotype_extendedElement),
     (?association1 rdfs:member ?link1),
     (?link1 rdf:_1 ?stereotype),
     (?association2
       rdf:type
       mm:Foundation.Core.A_namespace_ownedElement),
     (?association2 rdfs:member ?link2),
     (?link2 rdf:_2 ?stereotype),
     (?namespace
       mm:Foundation.Core.Namespace.ownedElement
       ?container2),
     (?container2 rdfs:member ?stereotype)
 ->
     drop(0,2,5,8),
     dropTriplesWithSubjects(?stereotype,?container1,?link1,?link2)
]
```

Listing 2. Jena rules to rename UML classes having the «resourceType» stereotype

The first rule fires once for each class having a «resourceType» stereotype. It renames the class by deleting the triple that associates it with its name and asserting a replacement triple with a literal value that is the original name with "Proposition" appended. The concat primitive has been defined to concatenate its first two arguments (assumed to be string literals) and instantiate its third argument to be a new typed xsd:string literal that is the concatenation of the first two arguments. The rule also asserts another triple that records the information that the stereotype should be deleted in subsequent processing. The object of that triple uses the Jena rule language's functor notation to represent the type of the resource to be deleted (a stereotype) and a related container resource that should also be deleted. The addition of that triple to the model will match the first clause in the left hand side of the second rule, and the remaining clauses will successfully match against any correct UML 1.3 model in our RDF representation. This rule will then fire, deleting all triples related to the stereotype as well as the fact that triggered the rule.

The built-in drop primitive removes triples matching certain patterns on the left hand side of the rule, given their indices, and the dropTriplesWithSubjects primitive has been defined to remove all triples having any of the arguments as its subject. Both primitives are non-monotonic in the sense that the resulting deletions do not cause the Rete rule mechanism to undo any prior rule firings that depended on the presence of those facts.

For the transformation to an ontology-specific content language, additional rules (not shown) are needed to modify the multiplicities of each relevant class's attributes and opposite association ends so that they become optional (an ontology may specify that a car must always have a colour, but it is not necessary for a proposition about the car to include that information).

7 RELATED WORK

As discussed in the introduction, the OMG issued a call for proposals for an Ontology Definition Metamodel (ODM) (OMG, 2003a). The current proposal (Sandpiper Software, 2006) includes metamodels for the Common Logic (CL) family of knowledge interchange languages (under final consideration by the International Organization for Standardization), RDF and RDF Schema, OWL, topic maps, and a "basic, minimally constrained" description logic. Mappings are defined from each metamodel (except CL) to and from OWL Full, but there is only a one-way mapping from OWL Full to CL. It is stated that some additional mappings are planned in the future: from UML 2.0 to and from CL, and a lossy mapping from CL to OWL. Once MOF-based metamodels for ontology modelling languages are standardised by the OMG, we hope that ontology engineering tool developers will see the interoperability benefits of supporting the resulting XMI serialisation formats. This will then provide a link between our work and these tools. As discussed in Section 5, our technique allows the conversion from an XMI file to our RDF representation, the transformation of the model using RDF tools, and the conversion of the result back to XMI.

In this paper we have discussed our approach to mapping ontologies to an RDF representation and analysing or transforming them using that format, using a UML model as an example. However, due to our use of a MOF-based repository and the reflective JMI interface, our technique will work with any modelling language that is defined using the MOF model. Thus, we could also use any of the languages for which the ODM work defines a metamodel. The work reported here is complementary to the ODM. While the current ODM draft specifies mappings between metamodels using QVT, this language has not yet completed the OMG standardization process. We have demonstrated that converting ontologies to equivalent representations in RDF allows existing RDF tools to be used to transform ontologies. We have shown that the Jena rule language can be used for this purpose. An alternative is to use the SPARQL RDF query language (Prud'hommeaux and Seaborne, 2005). This can be used to analyse an ontology represented in RDF using our technique by the use of queries (phrased in terms of properties and classes

corresponding to the metamodel), and can also be used to achieve transformations by the use of the CONSTRUCT operation (McCarthy, 2005).

We have previously used the XSLT language to define transformations on XMI serialisations of UML models (Cranefield, 2001), and this work has been extended by others (Pedrinaci et al., 2004; Gašević et al., 2004). However, these XSLT stylesheets were difficult to maintain, especially as several new versions of both XMI and UML have since been developed. In the current work, the use of NetBeans MDR removes the need to keep up with changing versions of XMI—instead we rely on the active development community of this tool to keep it up to date. However, a more significant concern is that the XMI format is defined in terms of the XML infoset (its abstract data model), which is at a more detailed level of information encoding than is of concern in ontology modelling. While ontology modelling languages are concerned with the relationships between concepts, the relationships between elements in an XMI file can be expressed in several different ways: by element nesting, by nesting of "proxy elements" that reference other elements using attribute references, or by directly using attributes as references. A stylesheet designed to process XMI data must be prepared to handle all of these, leading to duplicated logic. We believe that the RDF representation of MOF-based models presented in this paper and the use of tools to transform models at the RDF level provide a simpler and more natural generic approach to the serialisation, analysis and transformation of ontologies in various languages.

Sergey Melnik has previously implemented a mapping from UML to RDF (Melnik, 2000) and made this available via a now defunct Web site (Internet Archive, 2006). The work described in this paper uses his approach of creating URIs for metamodel elements, but was able to take advantage of more recent specifications and tools such as JMI and the NetBeans MDR to automate this process and can therefore handle models in any language defined using the MOF.

8 CONCLUSION

This paper has presented a technique for serialising models expressed in any MOF-based modelling language to an equivalent RDF representation, and shown how the resulting format is amenable to transformation using RDF tools, particularly the Jena rule language. This demonstrates that the aims of the OMG's Ontology Definition Model can be achieved today using current technology rather than waiting for the finalisation of the QVT language and its implementation within MOF-based tools. The significance of this work to ontology engineering is that it, in conjunction with the MOF-based metamodels for common ontology languages under development for the ODM specification, provides a route for ontologies in various ontology modelling languages to be imported into industrial-strength MDA model repositories and other tools, and provides a technique for defining transformations between other types of model (such as UML) and ontologies, between different ontology modelling languages, or to modify ontologies without changing the language. It also provides a generic serialisation format for all ontology modelling languages based on the RDF language, which is much better known in the ontology modelling community than XMI. We believe the simplicity and familiarity of the RDF model and its tools may offer advantages over the more complex QVT approach (OMG, 2005).

As discussed in Section 4.3, the MOF RDF mapping we have presented does not currently address all features of the MOF abstract mapping. The missing features would be straightforward to handle, but we have not yet added support for them as they were not needed for our initial application using UML 1.3 models.

We have proposed the use of the Jena rule language to transform ontologies in the RDF representation. The Jena rule engine was designed to make simple deductions, not transformations, and lacks the rich control features that fuller featured production systems have

(such as rule salience). This may make it difficult to use for complex multi-phase transformations. Alternative approaches worth exploring are the use of the SweetRules language (Grosof and Neogy, 2004) or CONSTRUCT queries in the SPARQL query language—the latter have been described as "somewhat analogous to an XSL transformation of XML data" (McCarthy, 2005).

The techniques described in this paper have been implemented using Java, the NetBeans Metadata Repository and the Jena toolkit, and tested with models in UML 1.3 as well as a simple metamodel that includes a structure type (these do not appear in the UML 1.3 metamodel) (Pan, 2006). We have used the Jena rule language to define the complete transformation to convert an ontology modeled in UML into an ontology-specific content language, as described in Section 6. One difficulty faced was that bugs in the rules can be difficult to discover—an RDF model that does not conform to the metamodel may cause JMI exceptions when it is imported into the repository, it may cause a malformed XMI file to be exported from the repository, which may be reported when loaded into another tool, or the error may simply cause some intended model elements to be missing from the XMI file. This problem could be solved if rather than simply generating URIs for elements of the metamodel, an RDF schema or OWL ontology corresponding to the metamodel were generated. The result of a transformation could then be validated against this schema or ontology, allowing early detection of errors.

Another area for future work is to implement the ontology metamodel translations defined by the ODM group as transformations on models encoded using the RDF mapping.

REFERENCES

Berners-Lee, T., 1998. Semantic Web Road map. Retrieved 2005-09-15, from http://www.w3.org/DesignIssues/Semantic.html.

Brickley, D. and Guha, R.V. (eds), 2004. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

Chang, D. T. and Kendall, E. K., 2004. Major influences on the design of ODM, in: Proceedings of the 1st International Workshop on the Model-Driven Semantic Web, 8th International IEEE Enterprise Distributed Object Computing Conference, http://www.sandsoft.com/edoc2004/ChangODMDesignMDSW.pdf.

Clark, J. (ed.), 1999. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116.

Colomb, R., Raymond, K., Hart, L., Emery, P., Welty, C., Xie, G. T. and Kendall, E., 2006. The Object Management Group Ontology Definition Metamodel, in Calero, C.; Ruiz, F.; Piattini, M. (eds.): Ontologies for Software Engineering and Software Technology, Springer, 217-248.

Costello, R., 2003. The Robber and the Speeder. Pages 33–45 of http://www.daml.org/meetings/2003/05/SWMU/briefings/08_Tutorial_D.ppt.

Cranefield, S. and Purvis, M., 2002. A UML profile and mapping for the generation of ontology-specific content languages. Knowledge Engineering Review. 17(1), 21–39.

Cranefield, S., 2001. Networked knowledge representation and exchange using UML and RDF. Journal of Digital Information. 1(8), http://journals.tdl.org/jodi/article/view/jodi-34/31.

Djurić, D., Gašević, D. and Devedžić, V. Ontology Modeling and MDA. Journal of Object Technology 4(1), http://www.jot.fm/issues/issue_2005_01/article3, 2005

Forgy, C. L., 1982. Rete: a fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence. 19(1), 17–37.

Gašević, D., Damjanović, V. and Devedžić, V., 2004. Analysis of MDA support for ontological engineering, in: Proceedings of the 4th Workshop on Computational Intelligence and Information Technologies, http://cs.elfak.ni.ac.yu/ciit/w4/papers/11.pdf.

Grosof, B. and Neogy, C., 2004. SweetRules project home page. Retrieved 2006-10-18, from http://sweetrules.projects.semwebcentral.org/.

HP Labs, 2002. Jena – A Semantic Web Framework for Java. Retrieved 2005-09-15, from http://jena.sourceforge.net/.

Internet Archive, 2006. Archive of http://www.interdataworking.com/. Retrieved 2006-01-30, from http://web.archive.org/web/*/http://www.interdataworking.com.

Java Community Process, 2002. The Java Metadata Interface (JMI) Specification, Version 1.0. JSR 40, http://jcp.org/aboutJava/communityprocess/final/jsr040/.

Knowledge Web, 2005. Benchmarking the Interoperability of Ontology Development Tools. Retrieved 2006-01-23, from http://knowledgeweb.semanticweb.org/benchmarking_interoperability/.

Manola, F. and Miller, E. (eds), 2004. RDF Primer. W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

McCarthy, P. Search RDF data with SPARQL. IBM developerWorks article, http://www-128.ibm.com/developerworks/xml/library/j-sparql/, 2005

McGuinness, D. L. and van Harmelen, F. (eds), 2004. OWL Web Ontology Language Overview. W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Melnik, S., 2000. Representing UML in RDF. Retrieved 2006-01-29, from http://www-db.stanford.edu/~melnik/rdf/uml/.

NetBeans.org, 2002. Metadata Repository (MDR) Project Home. Retrieved 2005-09-15, from http://mdr.netbeans.org.

OMG, 1997. Object Management Group home page. Retrieved 2005-09-15, from http://www.omg.org.

OMG, 2001. OMG Model Driven Architecture. Retrieved 2005-09-15, from http://www.omg.org/mda/.

OMG, 2002a. OMG XML Metadata Interchange (XMI) Specification, Version 1.2. OMG document formal/2002-01-01, http://www.omg.org/cgi-bin/doc?formal/2002-01-01.

OMG, 2002b. Meta Object Facility (MOF) Specification, Version 1.4. OMG document formal/2002-04-03, http://www.omg.org/cgi-bin/doc?formal/2002-04-03.

OMG, 2002c. MOF 2.0 Query / Views / Transformations RFP. OMG document ad/2002-04-10, http://www.omg.org/cgi-bin/doc?ad/2002-04-10.

OMG, 2003a. Ontology Definition Metamodel Request For Proposal. OMG document ad/2003-03-40, http://www.omg.org/cgi-bin/doc?ad/2003-03-40.

OMG, 2003b. Unified Modeling Language: Superstructure Version 2.0, Final Adopted specification. OMG document ptc/2003-08-02, http://www.omg.org/cgi-bin/doc?ptc/2003-08-02.

OMG, 2005. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Final Adopted Specification. OMG document ptc/05-11-01, http://www.omg.org/cgi-bin/doc?ptc/2005-11-01.

Pan, J., 2006. Enabling the Model Driven Architecture using RDF. MSc thesis, Department of Information Science, University of Otago.

Pedrinaci, C. Bernaras, A. Smithers, T. Aguado, J. and Cendoya, M., 2004. A framework for ontology reuse and persistence integrating UML and Sesame, in: Current Topics in Artificial Intelligence, 10th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2003, and 5th Conference on Technology Transfer, TTIA 2003. – revised selected papers. Lecture Notes in Computer Science, 3040, Springer, pp. 37–46.

Prud'hommeaux, E. and Seaborne, A. (eds), 2005. SPARQL Query Language for RDF. W3C Working Draft, http://www.w3.org/TR/2005/WD-rdf-sparql-query-20051123/.

Sandpiper Software. Ontology Definition Metamodel: Sixth Revised Submission to OMG/RFP ad/2003-03-40, OMG document ad/2006-05-01, http://www.omg.org/docs/ad/06-05-01.pdf, 2006

Steer, D., 2003. TreeHugger 0.1. Retrieved 2005-09-15, from http://rdfweb.org/people/damian/treehugger/.

Walsh, N., 2003. RDF Twig. Retrieved 2005-09-15, from http://rdftwig.sourceforge.net