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

This paper discusses the potential benefits to ontology engineering in making the toolset of the Object Management Group's model-driven architecture (MDA) applicable to ontology modelling, and describes the design of an MDA-based tool to convert ontologies expressed in any language having a metamodel defined used the OMG's MOF model to an equivalent representation in RDF but with the same metamodel. It is shown how this representation, compared to the XMI format, provides a higher level generic serialisation format for MDA models (especially ontologies) that is amenable to analysis and transformation using existing RDF tools. This helps to bridge the gap between the MDA and ontology engineering by providing a route for ontologies in various ontology modelling languages to be imported into industrial-strength MDA model repositories and other tools, and by allowing these ontologies to be transformed to and from other forms of model.

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 (MDA) [1] approach to software development proposed by the Object Management Group (OMG) [2]—an industry body devoted to the development and promotion of standards for enterprise computing. 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) [3], the XML Metadata Interchange (XMI) framework [4], the Unified Modeling Language (UML) [5], UML profiles and the Query/Views/Transformations (QVT) language [6]. 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 and a growing research community have been working to bring about Tim Berners-Lee's vision of the "Semantic Web" [7]. 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) [8], as well as predefined vocabularies for defining conceptual models of a domain: RDF Schema (RDFS) [9] and the Web Ontology Language (OWL) [10]. 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. Each concept and property in an ontology is given a uniform resource identifier (URI), allowing information and queries to be encoded unambiguously. In addition, OWL supports the inclusion of certain types of constraint in an ontology, allowing new information to be deduced when combining instance data with these ontological constraints (e.g., the fact that a serial number is unique to a gun is crucial in matching a gun to its registered owner [11]).

Ontologies have also been seen as important in other fields of research, such as multi-agent systems (MAS) and natural language processing. Based on the principle that "communication can be best modelled as the exchange of declarative statements" [12], multi-agent systems researchers have developed agent communication languages (ACLs) [13, 14] that use logic-based representations to convey the content of a message. To ensure that agents have a shared understanding of the constants, functions and predicates appearing in a message, these ACLs include an 'ontology' field that refers the message recipient to a pre-defined ontology in which these terms are defined (or, at least, declared). The recipient may respond with a 'not understood' message if it does not know this ontology, or it may attempt to dynamically acquire or learn the ontology. Enabling agents to make use of ontologies that were not part of their initial configuration is a topic of current research; however, in the present state of the art, many multi-agent systems are hard-coded to communicate using a particular set of ontologies, which are only used as specifications to be followed by the developer, and the agents do not directly reason with ontologies as data.

In natural language understanding, generation and translation, ontologies are used to represent the various meanings of words and the relations between words, providing background knowledge that is essential for resolving the ambiguities of natural. 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)" [15]. 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 recognised as a challenge for the community in a recent workshop announcement: "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" [16].

Supporting a heterogeneity of modelling languages, while providing standard representations and APIs for model repositories and other tools, is one of the aims of the MDA. 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 [3]—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 [17], 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) [18], and transformed using the QVT language [3]. 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) [19]. The OMG's Ontology Definition Metamodel (ODM) working group is currently developing MOF-based metamodels for a number of ontology modelling languages and transformations between them [15]. However, this is still work in progress, and it will depend on the completion of the QVT language [6] 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 (NetBeans Meta-Data Repository [17]) and the Java Metadata Interface (JMI) [18], we have implemented software that can convert any model defined using a MOF-based language to and from an equivalent model in an RDF representation (using the original metamodel, with its classes and properties represented as Semantic Web resources—i.e. we do not transform the metamodel to an RDF schema). Once the model is in the 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 or OWL representation). Currently we use the Jena rule language [20] to transform RDF models, but other transformation languages could also be supported [21, 22, 23].

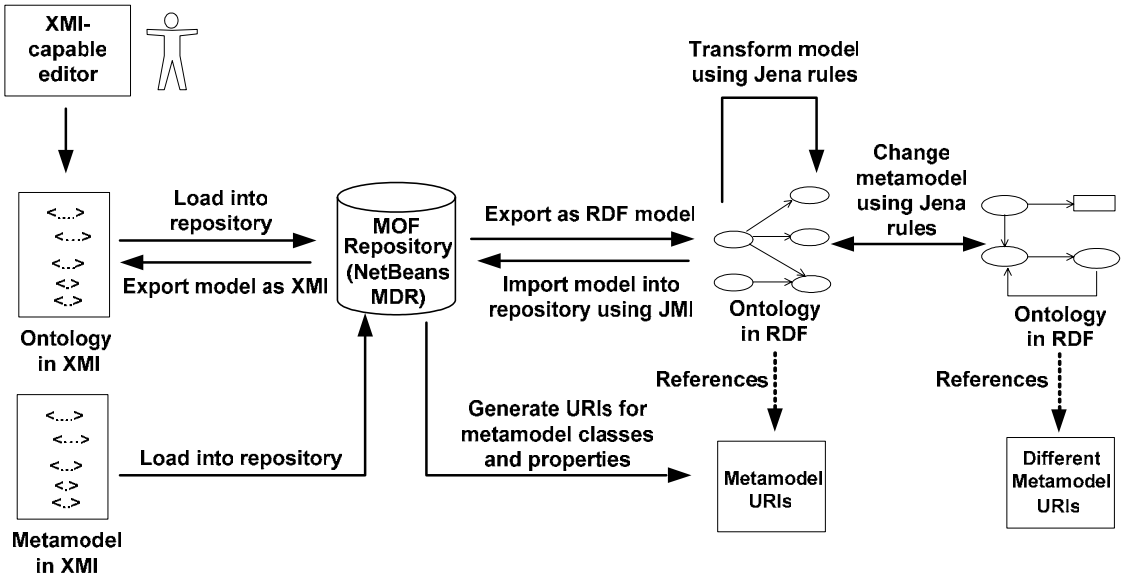

Figure 1: Overview of our approach to MDA-compliant ontology analysis and manipulation

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 languages that have been defined using the MOF model. Previous work on transforming ontologies in UML has focused on processing XMI files using the XSLT language [24]. 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 [23], 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 model-driven architecture. Section 3 briefly discusses the Semantic Web standards and tools used in this work, before Section 4 presents the details of our mapping from a ontology stored in a MOF-based repository to an equivalent representation using RDF. Section 5 discusses the use of the RDF representation as a basis for querying and transforming ontologies. Section 6 discusses related work and Section 7 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 metamodel is defined in terms of the MOF model. We illustrate this for a simple model in UML 1.3.

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 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—UML and the CORBA interface definition language (IDL) in the figure. Each of these in turn is a metamodel for models defined using the language, i.e. they define the modelling concepts that can be used in the definition of models. The bottom level (M0) of the metamodelling hierarchy represents instances of models, e.g. real-world entities conceptualised in terms of an ontology.

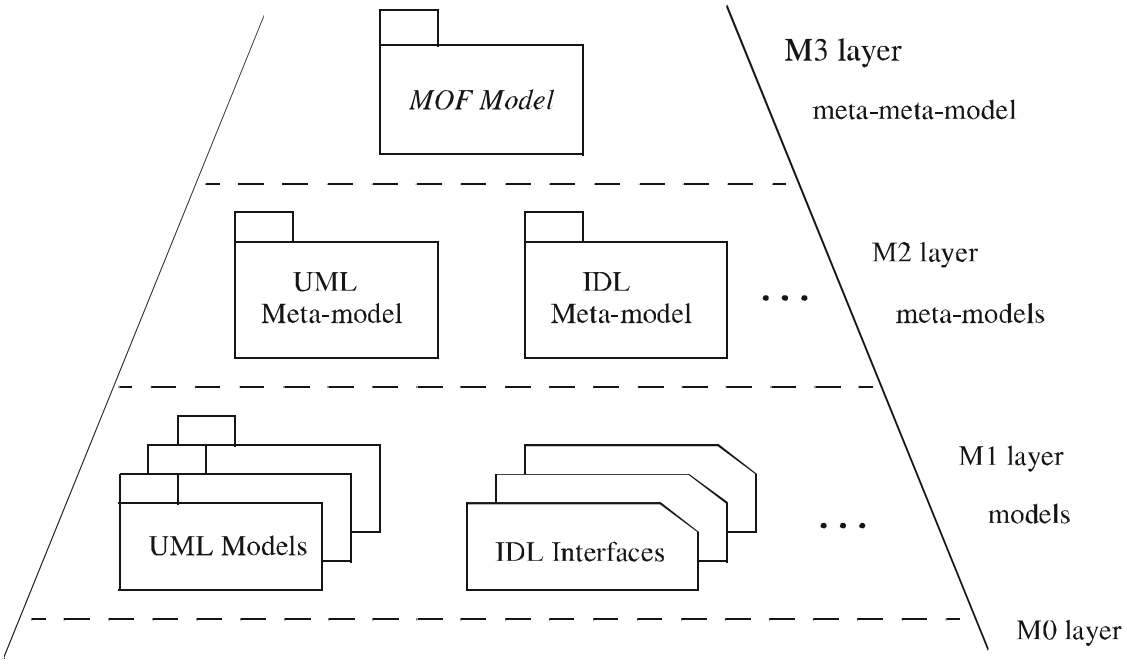

Figure 2: The MOF metamodelling hierarchy [3]

The MOF model is basically a simplified version of UML, and the UML notation can be used to depict metamodels defined in terms of the MOF model. Figure 3 shows a subset of the UML 1.3 metamodel as defined using the MOF model. This is known as UML's *physical metamodel* as it 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 [3]. 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 view of the UML model from Figure 4. This is how a MOF-based tool represents a model internally.

In this research we have used the NetBeans Meta-Data Repository (MDR) [17] to load an XMI serialisation of the metamodel for our chosen modelling language (currently UML 1.3) and then an XMI representation of a model (or, specifically, an ontology) in that language. The MDR is a MOF-based model repository that 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 the JMI API. With both the metamodel and one or more models loaded into the MDR, it is possible to use the JMI reflective interface (which makes no assumptions about the metamodel used) to extract enough information from the model to generate an RDF representation of it. Figure 6 gives an overview of the JMI view of the repository data, while details of the RDF representation are presented in Section 4.

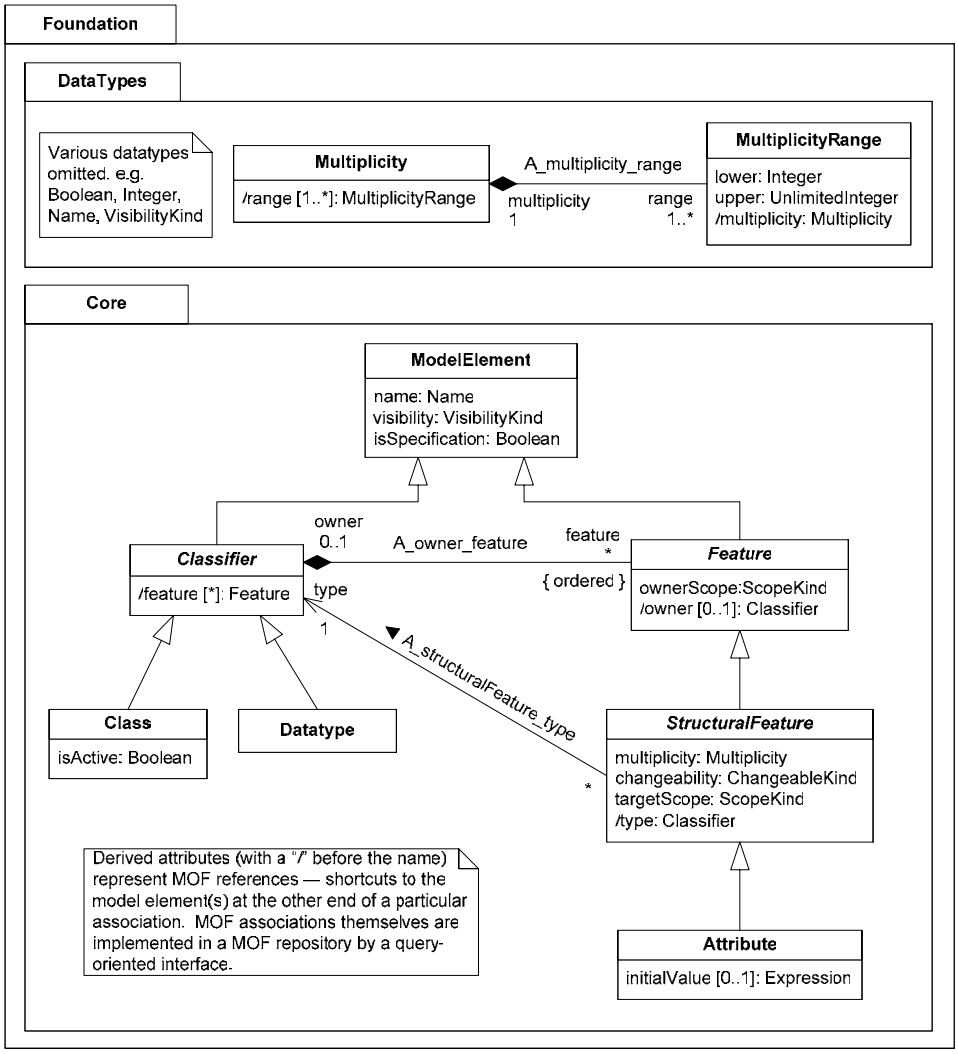

Figure 3: A subset of the UML 1.3 physical metamodel

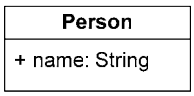

Figure 4: A simple UML model

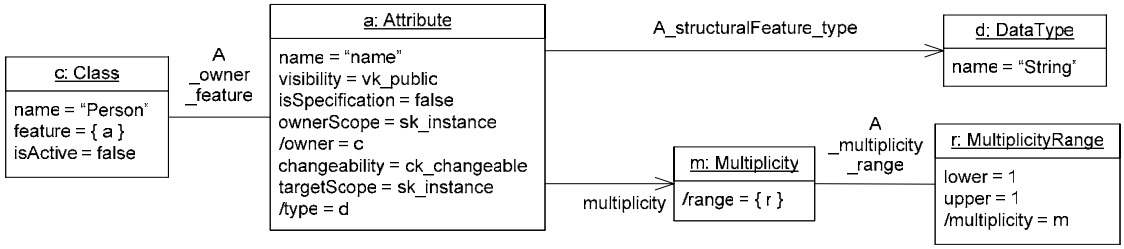

Figure 5: The model from Figure 4 as an instance of the UML physical metamodel

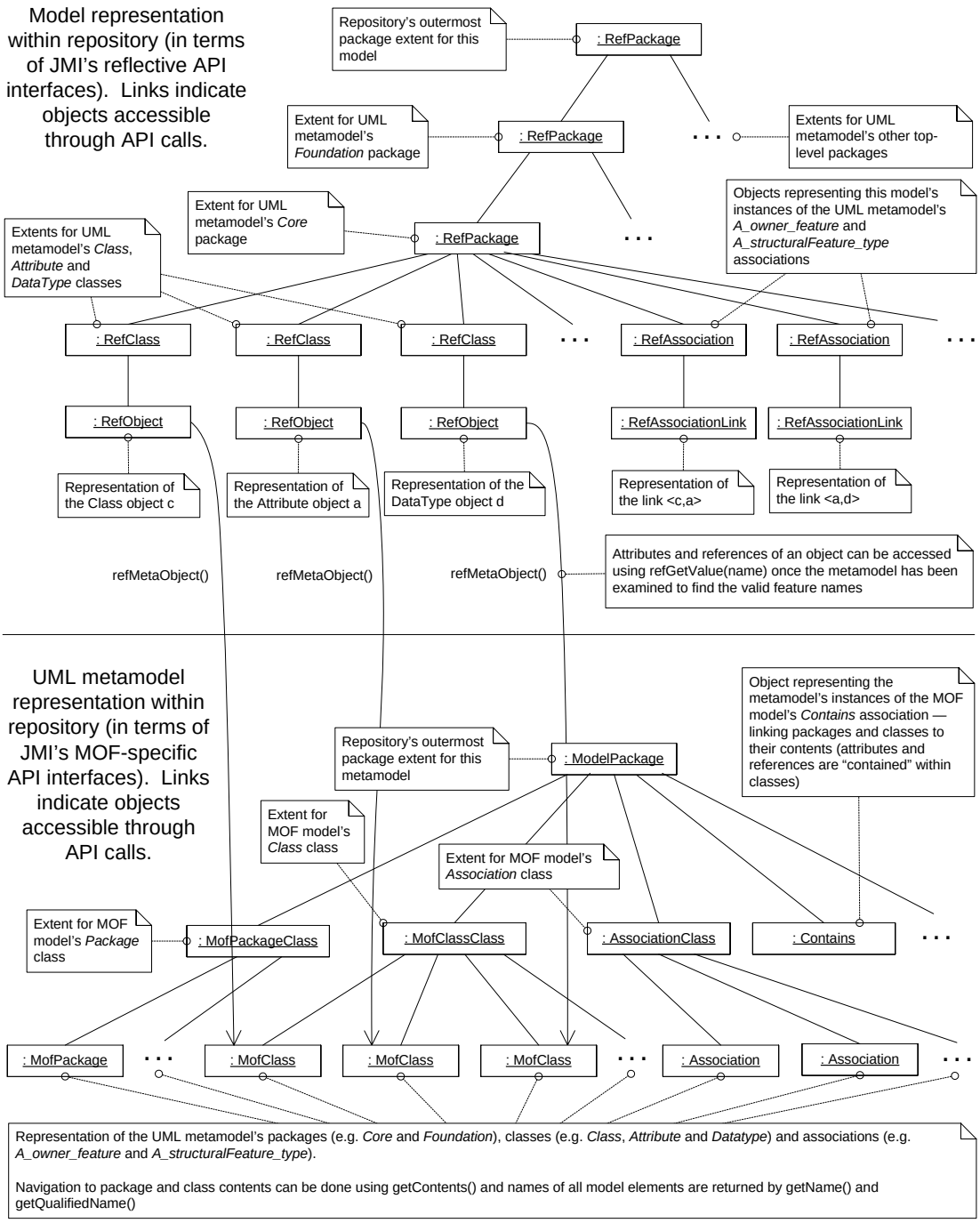

Figure 6: Representation of a model and its metamodel within the repository

The top half of Figure 6 illustrates the representation within the NetBeans 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 recursing through a set of package and class *extent* objects and objects representing associations. There is one extent for each of the packages and classes 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 and class extents and associations for a given package extent, all objects in a given class extent, and all links for a given association.

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 (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 knowledge of the metamodel (i.e. modelling language) used. The mapping from XMI to RDF discussed in Section 4 can therefore be done for any modelling language that has a definition in terms of the MOF model. This demonstrates how robust MDA tools such as the NetBeans MDR can potentially be used in conjunction with any ontology modelling language.

3 SEMANTIC WEB TECHNOLOGIES

This section gives an 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) [8] 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, where the subjects and predicates are resources identified by URIs, and the object can be another resource or a literal value. The RDF model forms a directed graph that encodes information about resources and their properties and relationships. 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 [9] and its extension, the Web Ontology Language (OWL) [10] 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 [11].

3.2 The Jena toolkit

Jena [20] is a Java toolkit 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 generating RDF statements, writing and reading RDF models and navigating and querying 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 ONTOLOGIES 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.

In previous work [24] 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 datatypes in the metamodel and use these as property and class resources in the RDF representation of the ontology.

In the repository, each element of a model is represented as an instance of its metaclass (as shown in Figure 5). In the view of the metamodel provided by the JMI reflective interface, each model element is represented by an instance of the RefObject interface. In most cases these map directly to RDF resources (the exception being metaclasses representing primitive datatypes, discussed below). 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 obtained using JMI's refMofId() call—this obtains a unique identifier for this RefObject that was generated by the repository when it was created. For each RefObject instance a statement of the following form[1] is asserted into the RDF model:

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

(m:MOF_ID rdf:type mm:QualifiedMetaclassName)

where m is the XML namespace chosen for the model, mm is a namespace that is uniquely associated with the metamodel and QualifiedMetaclassName is the fully qualified name of the metaclass for this RefObject, e.g. Foundation.Core.Class for the UML concept of a class.

Each RefObject will also have values for the attributes and references that are defined for its metaclass. These can be found using JMI as described in Section 2.1. For each attribute or reference, 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 as returned by the JMI reflective API. The representation in the RDF model is closely related to the JMI representation. 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). In the optional case, JMI encodes the lack of a value using a null Java value; however, we currently add no statement to the RDF model in this case[2].

In the multi-valued case the object of the RDF statement is a "blank" RDF node (one having a unique local identity within a given model, but without a URI) 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.

For the single-valued case, the representation of the value in RDF depends on whether the JMI encoding is a RefObject (representing a reference to another element in the model), a Java Boolean, Double, Float, Integer, Long or String (representing a value of a MOF primitive type), a RefEnum (representing a value of a MOF model enumeration type defined in the metamodel) or a RefStruct (representing a value of a MOF model structure type defined in the metamodel). RefObject instances map directly to RDF resources, using the chosen model namespace with the MOF repository ID as the local name. Values of the standard Java data types are mapped directly to RDF typed literals using XML Schema types. RefEnum values are also represented as typed literals, with the type URI being the metamodel namespace together with the type's qualified name in the metamodel, and the value literal is extracted from the RefEnum object using toString(). RefStruct values are treated similarly to RefObject instances, but are represented by blank nodes.

Associations are represented by JMI as instances of the interface RefAssociation, which represent sets of links. These are encoded in the RDF model by resources (with URIs formed from the model namespace plus the MOF repository ID) that represent these sets of links. 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

[2] Because our reverse mapping from the RDF model to a JMI representation within the repository assumes the XMI version of the metamodel is already present in the repository, it is not necessary to include in the RDF model statements expressing null values for attributes or references—when a new instance of the metamodel is created within the repository, initially empty objects representing all attribute values are also created.

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[3]: 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 an mmx_ns:IsAssociation property with value "true", where mmx_ns abbreviates a special namespace we reserve for metamodel-related properties that are not a direct representation of elements in the metamodel.

Due to the use of the JMI reflective API, the generation of the RDF representation of the ontology described above can be done without making any assumptions about the metamodel (i.e. modelling language) used. However, some metamodels need to declare a set of types that can be referenced in models, e.g. consider the reference to the data type String in Figure 4. To avoid a proliferation of different versions of the same datatypes from different metamodels, it is desirable to map these to a single common representation in the RDF representation: XML schema datatypes (as commonly used for RDF typed literals). This is currently done for UML by recognising the DataType metaclass. However, we intend to provide the transformation with an additional input specifying type mapping rules in order to avoid the need for any hard-coded knowledge about metamodels in the RDF generation program.

To illustrate the RDF mapping described above, a partial RDF representation of Figure 5 is shown below. For brevity we only show a few of the attribute values (chosen to illustrate the different encoding cases). The resource local names ID_c, ID_a, ID_d, ID_m and ID_r represent the MOF repository IDs for the model elements labelled c, a, d, m and r (respectively) in Figure 5, and ID_x, ID_y and ID_z represent the MOF repository IDs for the RefAssociation objects representing the link sets for the associations instantiated in the figure. Blank node identifiers are prefixed by "#" and we use the notation value^^type to indicate typed literals.

```
(m:ID_c  rdf:type  mm:Foundation.Core.Class)
(m:ID_c  mm:Foundation.Core.ModelElement.name  'Person'^^xsd:string)
(m:ID_c  mm:Foundation.Core.Classifier.feature  #A0)

(#A0  rdf:type  rdfs:Container)
(#A0  rdfs:member  m:ID_a)

(m:ID_a  rdf:type  mm:Foundation.Core.Attribute)
(m:ID_a  mm:Foundation.Core.ModelElement.visibility
            'vk_public'^^Foundation.Data_Types.VisibilityKind)
(m:ID_a  mm:Foundation.Core.StructuralFeature.multiplicity  m:ID_m)

(m:ID_m  rdf:type  mm:Foundation.Data_Types.Multiplicity)
(m:ID_m  mm:Foundation.Data_Types.Multiplicity.range  #A1)

(#A1  rdf:type  rdfs:Container)
(#A1  rdfs:member  m:ID_r)

(m:ID_r  rdf:type  mm:Foundation.Data_Types.MultiplicityRange)
(m:ID_r  mm:Foundation.Data_Types.MultiplicityRange.multiplicity  m:ID_m)
(m:ID_r  mm:Foundation.Data_Types.MultiplicityRange.lower  '1'^^xsd:int)
(m:ID_r  mm:Foundation.Data_Types.MultiplicityRange.upper  '1'^^xsd:int)
```

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

```
(m:ID_x  rdf:type  mm:Foundation.Core.A_owner_feature)
(m:ID_x  mmx_ns:IsAssociation  'true'^^xsd:boolean)
(m:ID_x  rdfs:member  #A2)

(#A2  rdf:type  rdf:Seq)
(#A2  rdf:_1  m:ID_c)
(#A2  rdf:_2  m:ID_a)

(m:ID_y  rdf:type  mm:Foundation.Core.A_structuralFeature_type)
(m:ID_y  mmx_ns:IsAssociation  'true'^^xsd:boolean)
(m:ID_y  rdfs:member  #A3)

(#A3  rdf:type  rdf:Seq)
(#A3  rdf:_1  m:ID_a)
(#A3  rdf:_2  xsd:string)

(m:ID_z  rdf:type  mm: Foundation.Data_Types.A_multiplicity_range)
(m:ID_z  mmx_ns:IsAssociation  'true'^^xsd:boolean)
(m:ID_z  rdfs:member  #A4)

(#A4  rdf:type  rdf:Seq)
(#A4  rdf:_1  m:ID_m)
(#A4  rdf:_2  m:ID_r)
```

In this section we have presented the mapping from a JMI encoding of an ontology to a representation in RDF. This mapping preserves all information in the ontology that is required by the metamodel and allows it to be analysed or transformed by any program that has knowledge (hard-coded or learned) about the structure of the metamodel. However, 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. It is straightforward to reverse the mapping described above to populate a MOF repository using JMI, provided that the XMI metamodel has already been loaded into the repository.

5 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 [23]. 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 [30] 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. Below we present two Jena rules that implement part of this transformation, given as input the RDF representation of an ontology encoded in a particular UML profile [29]:

```
[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)
]
```

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 then 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).

6 RELATED WORK

As discussed in the introduction, the OMG has released a call for proposals for an Ontology Definition Metamodel [15]. The current draft proposal [26] includes metamodels for RDFS, OWL, a "weakly constrained abstract formulation of Description Logic" (DL), entity-relationship models, topic maps and Simple Common Logic (SCL), and also includes the UML 2.0 metamodel within its scope. The DL metamodel is intended to be used as an interlingua when mapping between metamodels, i.e. they would first be mapped to this metamodel, thus avoiding the need for mappings between each pair of metamodels. The SCL metamodel is intended only for encoding predicates that cannot be expressed in other metamodels, and will therefore be the target of a one-way mapping from the DL metamodel. The proposal does not yet define any mappings between metamodels, but notes that the intention is to use the QVT language [3] for this purpose when it is finished.

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 because while the current ODM draft specifies metamodels but not (yet) mappings, 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 [23]. This can be used to analyse an ontology mapped to 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 [28].

We have previously used the XSLT language to define transformations on XMI serialisations of UML models [24], and this work has been extended by others [31, 32]. 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.

7 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 allows these ontologies to be transformed to and from other forms of models. 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 as currently proposed [33].

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. An alternative option for these is the use of CONSTRUCT queries in the SPARQL query language as these are "somewhat analogous to an XSL transformation of XML data" [28].

An important area for future workis to develop RDF-based mappings between the metamodels proposed in the ODM draft.

REFERENCES

[1] Object Management Group. Model-Driven Architecture home page. http://www.omg.org/mda/. Accessed 15/9/05

[2] Object Management Group home page. http://www.omg.org. Accessed 15/9/05

[3] Object Management Group. 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, 2002

[4] Object Management Group. OMG XML Metadata Interchange (XMI) Specification, version 1.2. OMG document formal/2002-01-01, 2002. http://www.omg.org/cgi-bin/doc?formal/2002-01-01, 2002

[5] Object Management Group. 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, 2003

[6] Object Management Group. MOF 2.0 Query / Views / Transformations RFP. OMG document ad/2002-04-10, ftp://ftp.omg.org/pub/docs/ad/03-03-40.pdf, 2002

[7] Berners-Lee, T. Semantic Web Road map, http://www.w3.org/DesignIssues/Semantic.html, 1998

[8] Klyne G. and Carroll J. (eds.) Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004

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

[10] McGuinness, D. L. and van Harmelen, F. Web Ontology Language Overview. W3C Recommendation, http://www.w3.org/TR/owl-features/, 2004

[11] Costello, R. The Robber and the Speeder, pages 33–35 of http://www.daml.org/meetings/2003/05/SWMU/briefings/08_Tutorial_D.ppt, 2003

[12] Genesereth, M. R. and Ketchpel, M. R. Software agents. Communications of the ACM, 37(7): 48–53, July 1994.

[13] Finin, T., Labrou, Y. and Mayfield, J. KQML as an agent communication language. In J. Bradshaw, editor, Software Agents. MIT Press, Cambridge, 1997

[14] Foundation for Intelligent Physical Agents. FIPA ACL message representation in string specification. FIPA specification 70, http://www.fipa.org/specs/fipa00070/, 2002

[15] Object Management Group. Ontology Definition Metamodel Request For Proposal. OMG document ad/2003-03-40, ftp://ftp.omg.org/pub/docs/ad/03-03-40.pdf, 2003

[16] Interoperability Working Days announcement, http://knowledgeweb.semanticweb.org/benchmarking_interoperability/working_days/, 2005

[17] Netbeans Metadata Repository. http://mdr.netbeans.org. Accessed 15/9/05.

[18] The Java Metadata Interface (JMI) Specification, version 1.0. JSR 40, http://www.jcp.org/en/jsr/detail?id=40, 2002

[19] Chang, D. T. and Kendall, E. K. Major Influences on the Design of ODM, 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, 2004

[20] Jena API home page. http://jena.sourceforge.net/. Accessed 15/9/2005

[21] Treehugger. http://rdfweb.org/people/damian/treehugger/. Accessed 15/9/2005

[22] RDF Twig. http://rdftwig.sourceforge.net/. Accessed 15/9/200

[23] Prud'hommeaux, E. and Seaborne, A. (eds.) SPARQL Query Language for RDF. W3C Working Draft, http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/, 2004

[24] Cranefield, S. Networked Knowledge Representation and Exchange using UML and RDF. Journal of Digital Information 1(8), http://jodi.ecs.soton.ac.uk/Articles/v01/i08/Cranefield/, 2001

[25] Clark, J. (ed.) XSL Transformations (XSLT) Version 1. W3C Recommendation, http://www.w3.org/TR/xslt, 1999

[26] DSTC Pty. Ltd., Gentleware, Inc., IBM and Sandpiper Software. Combined Revised Response to the OMG's RFP for an Ontology Definition Metamodel, http://codip.grci.com/odm/draft/, 2004

[27] 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

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

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

[30] Forgy, C. L. Rete: a fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, pp.17–37, 1982

[31] Pedrinaci, C. Bernaras, A. Smithers, T. Aguado, J. and Cendoya, M. A Framework for Ontology Reuse and Persistence Integrating UML and Sesame. 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 vol. 3040, pp. 37–46, Springer, 2004

[32] Gašević, D., Damjanović, V. and Devedžić, V. Analysis of MDA Support for Ontological Engineering. Proceedings of the 4th Workshop on Computational Intelligence and Information Technologies, http://cs.elfak.ni.ac.yu/ciit/w4/papers/10.pdf, 2004

[33] Colomb, R. L., Gerber, A. and Lawley, M. Issues in Mapping Metamodels in the Ontology Development Metamodel. Proceedings of the1st International Workshop on the Model-Driven Semantic Web, 8th International IEEE Enterprise Distributed Object Computing Conference, http://www.itee.uq.edu.au/~colomb/Papers/MappingODMDSTCIEEE.pdf, 2004