UML and the Semantic Web

Stephen Cranefield
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
E-mail: scranefield@infoscience.otago.ac.nz

Abstract

This paper discusses technology to support the use of UML for representing ontologies and domain knowledge in the Semantic Web. Two mappings have been defined and implemented using XSLT to produce Java classes and an RDF schema from an ontology represented as a UML class diagram and encoded using XMI. A Java application can encode domain knowledge as an object diagram realised as a network of instances of the generated classes. Support is provided for marshalling and unmarshalling this object-oriented knowledge to and from an RDF/XML serialisation.

1 Introduction

The growth of the World Wide Web and its integration into business and everyday life has been phenomenal. The number of Web servers has grown from 26 in 1992 [1] to an estimated six million in 2000 [2]. However, this growth has meant that the wealth of available information on any subject is swamped by an overwhelming mass of irrelevant information, and the complexity and dynamic nature of the current Web structure means it is only possible for people to find, index and make effective use of a tiny fraction of the Web.

The solution to this problem is to let computer software relieve us of much of the burden of locating resources on the Web that are relevant to our needs and extracting, integrating and indexing the information contained within. To enable this, the Web must evolve from its current status as a network of resources intended for human comprehension. The Semantic Web concept [3] requires that

Web content be interpretable by both humans and machines. In particular, resources on the Web need to be encoded in, or annotated with, structured machine-readable descriptions of their contents. To this end, the World Wide Web Consortium (W3C) has introduced the Extensible Markup Language (XML) and the Resource Description Framework (RDF) as standard mechanisms for embedded structured data and metadata in Web documents. These languages both have an associated schema language (XML Schema and RDF Schema respectively) allowing communities to define their own vocabularies for indicating document structure and describing its content.

However, the concept of the Semantic Web involves more than just communities sharing information after agreeing on common data and metadata formats. The aim is to allow information to be shared across communities by providing the ability to translate information between different representations. This requires that the information in a Web resource be encoded as *knowledge* expressed using terms or structures that have been explicitly defined in a domain *ontology*. Ontologies are, in essence, schemas that are expressed in a high-level modelling language suitable for modelling the concepts in the domain, the relationships between them, and any logical constraints on their interpretation. They are not concerned with the physical format of documents but focus on the conceptual content.

Currently, there is a lot of research effort underway to develop ontology representation languages compatible with World Wide Web standards, particularly in the Ontology Inference Layer (OIL [4]) and DARPA Agent Markup Language (DAML [5]) projects. Derived from frame-based representation languages from the artificial intelligence knowledge representation community, OIL and DAML schema build on top of RDF Schema by adding modelling constructs from description logic [6]. This style of language has a well understood semantic basis but lacks both a wide user community outside AI research laboratories and a standard graphical presentation—an important consideration for aiding the human comprehension of ontologies.

This paper discusses Semantic Web technology based on an alternative paradigm that also supports the modelling of concepts in a domain (an ontology) and the expression of information in terms of those concepts. This is the paradigm of object-oriented modelling from the software engineering community. In particular, there is an expressive and standardised modelling language, the Unified Modeling Language (UML [7]), which has graphical and XML-based formats, a huge user community, a high level of commercial tool support and an associated constraint language with the expressive power of first-order logic. Although developed to support analysis and design in software engineering, UML is beginning

to be used for other modelling problems, one notable example being its adoption by the Meta Data Coalition [8] for representing metadata schemas for enterprise data.

The proposed application of UML to the Semantic Web is based on the following three claims:

- UML class diagrams provide a static modelling capability that is well suited for representing ontologies [9].
- UML object diagrams can be interpreted as declarative representations of knowledge [10].
- If a Semantic Web application is being constructing using object-oriented technology, it may be advantageous to use the same paradigm for modelling ontologies and knowledge.

Further discussion of these points is beyond the scope of this paper which focuses on technology to support the use of object-oriented modelling for the Semantic Web.

2 Required technology for UML and the Semantic Web

To enable the use of UML representations of ontologies and knowledge, standard formats are needed to allow both ontologies and knowledge about domain objects to be published on the Web and transmitted between agents. In addition, there is a need for code libraries to help application writers parse and internalise this serialised information.

This technology already exists for UML class diagrams. The XML Model Interchange language (XMI) defines a standard way to serialise UML models. There are also a number of Java class libraries existing or under development to provide a convenient interface for applications to access this information. However, there is currently no similar technology available to help Java applications construct, serialise and read object-oriented encodings of knowledge that are conceptualised as UML object diagrams. XMI documents can encode the structure of object diagrams, but this is necessarily done in a domain-independent way using separate but cross-referenced XML elements for each object, link, link end and attribute–value binding. What is required is a way to generate from an ontology a

domain-specific encoding format for knowledge about objects in that domain, and an application programmer interface (API) to allow convenient creation, import and export of that knowledge.

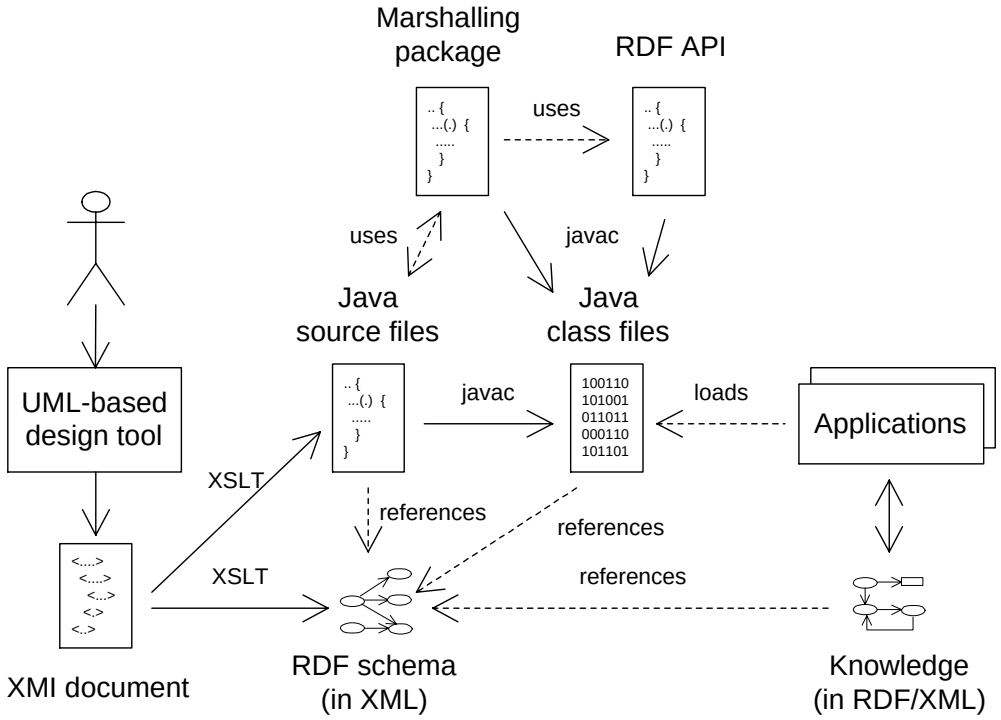


Figure 1: Overview of the implemented technology for object-oriented knowledge representation

Figure 1 shows an approach to object-oriented ontological and object-level knowledge representation that has been implemented and is described in this paper. First, a domain expert designs an ontology graphically using a CASE tool supporting the Unified Modeling Language, and then saves it using the standard XML-based format XMI (XML Model Interchange). A pair of XSLT (Extensible Stylesheet Language Transformations) stylesheets then take the XMI representation of the ontology as input and produce (i) a set of Java classes and interfaces corresponding to those in the ontology, and (ii) a representation of the ontology using RDF (Resource Description Framework) using the modelling concepts defined in RDF Schema. The generated Java classes allow an application

to represent knowledge about objects in the domain as in-memory data structures. The generated schema in RDF defines domain-specific concepts that an application can reference when serialising this knowledge using RDF (in its XML encoding). The marshalling and unmarshalling of object networks to and from RDF/XML documents is performed by a pair of Java classes: `MarshalHelper` and `UnmarshalHelper`. These delegate to the generated Java classes decisions about the names and types of fields to be serialised and unserialised, but are then called back to perform the translation to and from RDF making use of an existing Java RDF application programmer's interface [11].

Note that the generated RDF schema does not contain all the information from the original UML model. If an application needs access to full ontological information, it can use the original XMI document with the help of one of the currently available or forthcoming Java APIs supporting the processing of UML models. The purpose of the RDF schema is to define RDF resources corresponding to all the classes, interfaces, attributes and associations in the ontology in order to support RDF serialisation of instance information. For the sake of human readers, the schema records additional information such as subclass relationships and the domains and ranges of properties corresponding to attributes and associations. However, this information is not required for processing RDF-encoded instance information because each generated Java class contains specialised methods `marshalFields` and `unmarshalFields` containing hard-coded knowledge about the names and types of the class's fields. This is a design decision intended to avoid potentially expensive analysis of the schema during marshalling and unmarshalling. This it should be possible to use this serialisation mechanism in situations where optimised serialisation is important, such as in agent messaging systems.

3 An example domain

This section presents an example ontology modelled as a UML class diagram and some knowledge encoded as an object diagram. The ontology defines a subset of the concepts included in the CIA World Factbook and is adapted from an OIL representation of a similar subset [12].

3.1 An ontology in UML

Figure 2 presents the CIA World Factbook ontology represented as a UML class diagram. The version shown here is not a direct translation from OIL: there is an additional class (`AdministrativeDivision`), UML association classes are used where appropriate, and instead of defining the classes `City` and `Country` as specialised types of `Region` (GeographicalLocation in the OIL original), the ontology represents these as optional roles that a region may have.

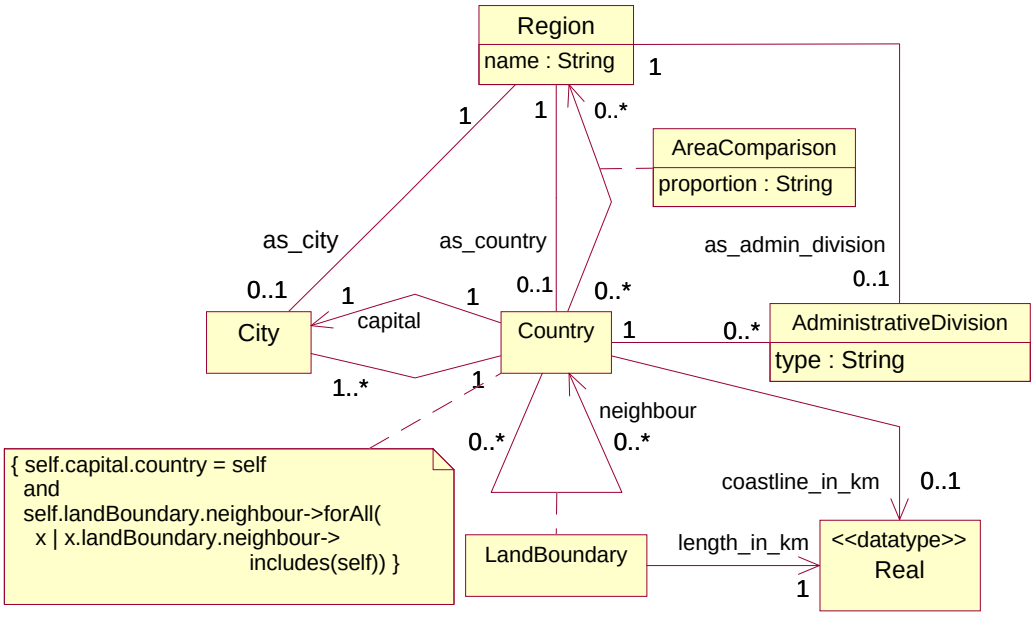


Figure 2: A UML ontology for a subset of the CIA World Factbook

The boxes in the diagram represent classes, and contain their names and (where applicable) their attributes. The lines between classes depict association relationships between classes. A class `A` has an association with another class `B` if an object of class `A` needs to maintain a reference to an object of class `B`. An association may be bidirectional, or (if a single arrowhead is present) unidirectional. A 'multiplicity' expression at the end of an association specifies how many objects of that class may take part in that relationship with a single object of the class at the other end of the association. This may be expressed as a range of values,

with '`*`' indicating no upper limit. Association ends may be optionally named. In the absence of a name, the name of the adjacent class, with the initial letter in lower case, is used as a default name. Associations can be explicitly represented as classes by attaching a class box to an association (see `LandBoundary` and `AreaComparison` in the figure). This is necessary when additional attributes or further associations are required to clarify the relationship between two classes.

The dog-eared rectangle in the lower left corner of the figure contains a constraint in the Object Constraint Language (OCL). This language provides a way to constrain the possible instances of a model in ways that cannot be expressed using UML's structural modelling elements alone. The constraint shown here states that i) a country's capital is a city in that country, and ii) if a country c has another as a neighbour, then that neighbouring country has c as a neighbour. Finally, the keyword "datatype" appearing in guillemets above the class `Real` indicates that this is a pre-existing built-in datatype. OCL defines a minimal set of primitive datatypes and it is currently assumed that the ontology designer has used these primitive types.

3.2 Knowledge as an object diagram

Figure 3 presents some knowledge about New Zealand from the CIA World Factbook, expressed as an object diagram. For brevity, this diagram omits most of New Zealand's cities and administrative divisions, and provides no information about the region named Colorado, to which New Zealand is compared in terms of area.

In object diagrams, rectangles denote objects, specifying their class (after an optional name and a colon) and the object's attribute values. The lines between objects show 'links': instances of associations between classes.

4 From UML to RDF and Java

The previous section presented some knowledge expressed as a UML object diagram. This is an abstract representation of that knowledge. To enable this style of knowledge representation to be used for Semantic Web applications it is necessary to define an API to allow creation of the knowledge in this form and a serialisation format to allow Web publication and transmission of the knowledge. These can be generated automatically from an XML encoding of the Word Factbook using the XSLT stylesheets that have been developed. One stylesheet produces an RDF

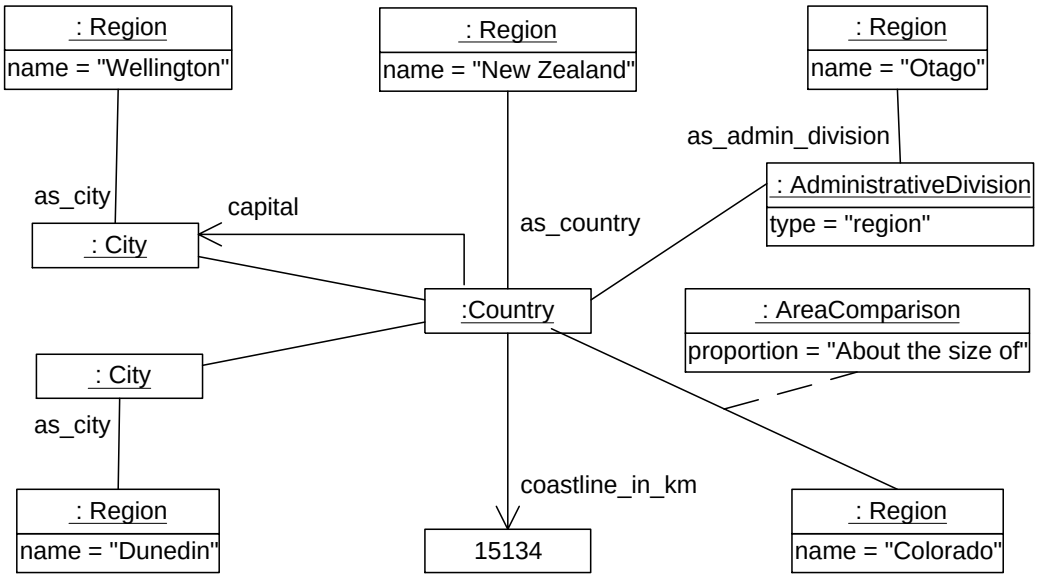


Figure 3: Information about New Zealand as a UML object diagram

schema corresponding to the ontology and the other produces a corresponding set of Java classes and interfaces.

XSLT is a language for transforming XML documents into other documents. An XSLT stylesheet is comprised of a set of templates that match nodes in the input document (represented internally as a tree) and transform them (possibly via the application of other templates) to produce an output tree. The output tree can then be output as text or as an HTML or XML document.

The main issue common to both mappings is the problem of translating from UML classes, which may have different types of features such as attributes, associations and association classes, to a model where classes only have fields or (in RDF) properties. It was also necessary to generate default names for fields where association ends are not named in the UML model. The OCL conventions for writing navigation paths through object structures were used to resolve these issues. Also, attributes and association ends with a multiplicity upper limit greater than one are represented as set-valued fields (bags in RDF Schema) or, in the case of association ends with a UML "ordered" constraint, list-valued fields (sequences in RDF Schema). Further details about the mappings have been discussed elsewhere [13] and are beyond the scope of this paper.

4.1 The generated RDF schema

The Resource Description Framework (RDF) is a simple resource–property–value model designed for expressing metadata about resources on the Web. RDF has a graphical syntax as well as an XML-based serialisation syntax. For readability, examples in this paper are presented in the graphical syntax, although in practice they are generated in the XML format.

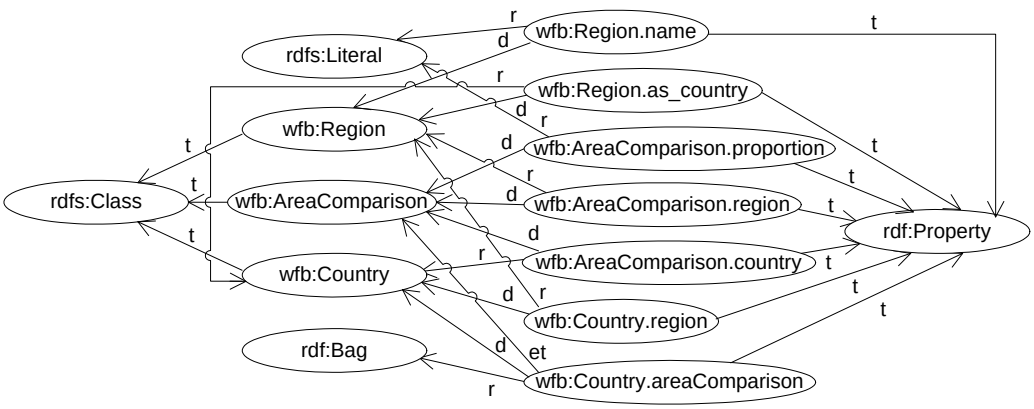
RDF Schema is a set of predefined resources (entities with uniform resource identifiers) and relationships between them that define a simple meta-model including concepts of classes, properties, subclass and subproperty relationships, a primitive type `Literal`, bag and sequence types, and domain and range constraints on properties. Domain schemas (i.e. ontologies) can then be expressed as sets of RDF triples using the (meta)classes and properties defined in RDF Schema. Schemas defined using RDF Schema are commonly called RDF schemas (small 's').

The main issue in generating an RDF schema that corresponds to an object-oriented model is that RDF properties are first-class objects and are not defined within the context of a particular class. This can lead to conflicting range declarations if the same property (e.g. `head`) is used to represent a field in two different classes (e.g. `Brew` and `Department`). The solution chosen was to prefix each property name representing a field with the name of the class. This has the disadvantage that in the presence of inheritance a class's fields may be represented by properties with different prefixes: some specifying the class itself and some naming a parent class. This might be confusing for a human reader but is not a problem for the current purpose: to specify a machine-readable format for object-oriented knowledge interchange.

Figure 4 presents a subset of the generated RDF schema corresponding to the UML model presented in Figure 2. Only the classes `Country` and `Region` and the relationships between them are included here.

In the standard RDF graphical notation used in the figure, an ellipse represents a resource with its *qualified name* shown inside as a namespace prefix followed by a local name. A namespace prefix abbreviates a Uniform Resource Identifier (URI) associated with a particular namespace, and the URI for the resource can be constructed by appending the local name to the namespace URI. A property is represented by an arc, with the qualified name for the property written beside the arc (in this case the arcs are given labels with the corresponding URIs shown in the table).

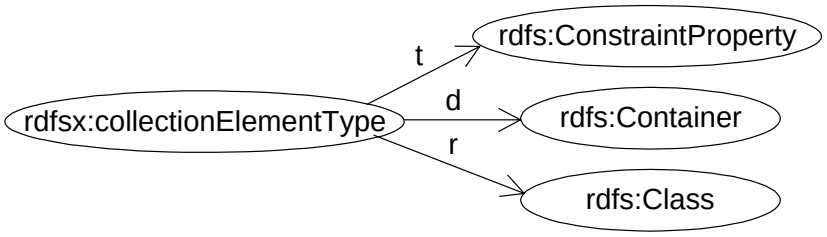
Figure 4 includes one property that is not part of RDF Schema. There is no



Property labels	Name space abbreviations
t = rdf:type	rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
d = rdfs:domain	rdfs = http://www.w3.org/2000/01/rdf-schema#
r = rdfs:range	rdfsx = http://nzdis.otago.ac.nz/0_1/rdf-schema-x#
et = rdfsx:collectionElementType	wfb = *any new namespace chosen for this schema*

Figure 4: Part of the World Factbook schema in RDF

mechanism in RDF Schema to parameterise a collection type (such as `rdf:Bag`) by the class of elements it may contain. Therefore, the non-standard property `rdfsx:collectionElementType` was introduced to represent this information (this is abbreviated in the figure by the arc label `et`). The definition of this property is shown in Figure 5. The object serialisation mechanism described in this paper does not require this information but it is useful to people reading the schema.

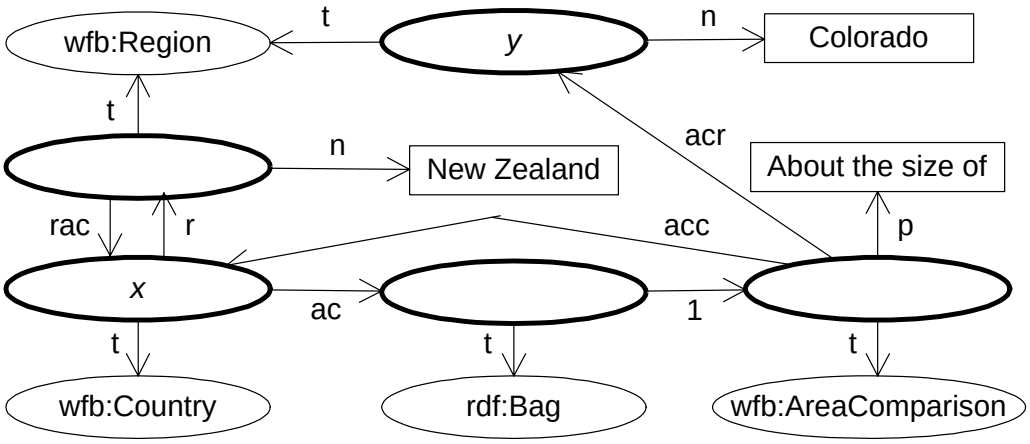


Property labels	Name space abbreviations
t = rdf:type	rdfs = http://www.w3.org/2000/01/rdf-schema#
d = rdfs:domain	rdfsx = http://nzdis.otago.ac.nz/0_1/rdf-schema-x#
r = rdfs:range	

Figure 5: An extension to RDF Schema

The schema in Figure 4 completely defines the encoding of instance data in RDF. Figure 6 shows how an object diagram is encoded in RDF with reference to the schema. This corresponds to the central `Region` and `Country` objects from Figure 3 together with the `AreaComparison` link to the Colorado `Region` object. The five resources outlined in bold are the ones being defined. There are two resources of type `wfb:Region`, one of type `wfb:Country`, one of type `rdf:Bag` (representing the set of the country's area comparisons) and one element in the bag, an instance of the area comparison association class (which is represented in RDF as the type `wfb:AreaComparison`). Depending on the needs of the application, defined objects might be assigned URIs or represented as anonymous resources. In this case, they are anonymous—the labels x and y are included to allow reference to these resources later in this paper. The rectangles represent RDF literals.

Note that while this graphical notation for RDF looks complicated, the XML encoding for RDF only requires five XML elements to represent the information.



Property labels	Name space abbreviations
t = rdf:type 1 = rdf:_1 n = wfb:Region.name rac = wfb:Region.as_country r = wfb:Country.region ac = wfb:Country.areaComparison p = wfb:AreaComparison.proportion acr = wfb:AreaComparison.region acc = wfb:AreaComparison.country	rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns# wfb = *the namespace chosen for the World Factbook schema*

Figure 6: Information about New Zealand encoded in RDF

4.2 The generated Java classes and marshalling framework

The generated RDF schema described in the previous section defines a domain-specific serialisation format for object-oriented representation of knowledge about the domain. To facilitate the processing of knowledge communicated in this form, a set of Java classes can also be generated from the ontology using XSLT. These allow Java applications to instantiate instances of the domain concepts. In addition, the generated classes, along with some additional utility classes, allow these in-memory structures to be marshalled and unmarshalled to and from the RDF serialisation format defined by the generated RDF schema. The aim of the marshalling code is to allow a Java application to maintain an internal representation of object-oriented knowledge and to easily read and write parts of this knowledge to and from a format suitable for transmission or publication on the Web.

Figure 7 presents a class diagram outlining the structure of the generated Java classes and the marshalling framework. The class `MarshalHelper` is part of a support package used by the generated classes. It contains a static method `marshalObjects` that provides the entry point for an application to marshal a network of objects. A similar class `UnmarshalHelper` is also provided, but is not discussed here. The class `DomainObject` is an abstract base class that all generated classes specialise (the specialisation relationship is represented by a closed arrow pointing to the more general class). The class `Region` is shown as an example of a generated class.

This diagram does not show all the fields and methods. In particular, the class `Region` also contains fields and methods related to the `as_city`, `as_country` and `as_admin_division` association ends from the ontology shown in Figure 2. There are some fields and methods depicted that are related to whether or not a field value is "known". This is discussed in Section 5.

There is a significant difference between knowledge represented propositionally and knowledge represented in the form of an object diagram. Propositions are self-contained statements of knowledge whereas object diagrams are networks of objects. When serialising knowledge, an application may only wish to include some of the information it knows about a domain. For example, Figure 6 compares New Zealand's area to that of Colorado, but doesn't provide the information that Colorado is an administrative division of the United States. To allow this selectivity, the `marshalObjects` method takes a collection of objects as an argument. Links to any objects outside this collection will not be serialised. To allow a particular entry point into the knowledge structure to be identified, a root object is specified and the method returns the qualified name of the RDF resource

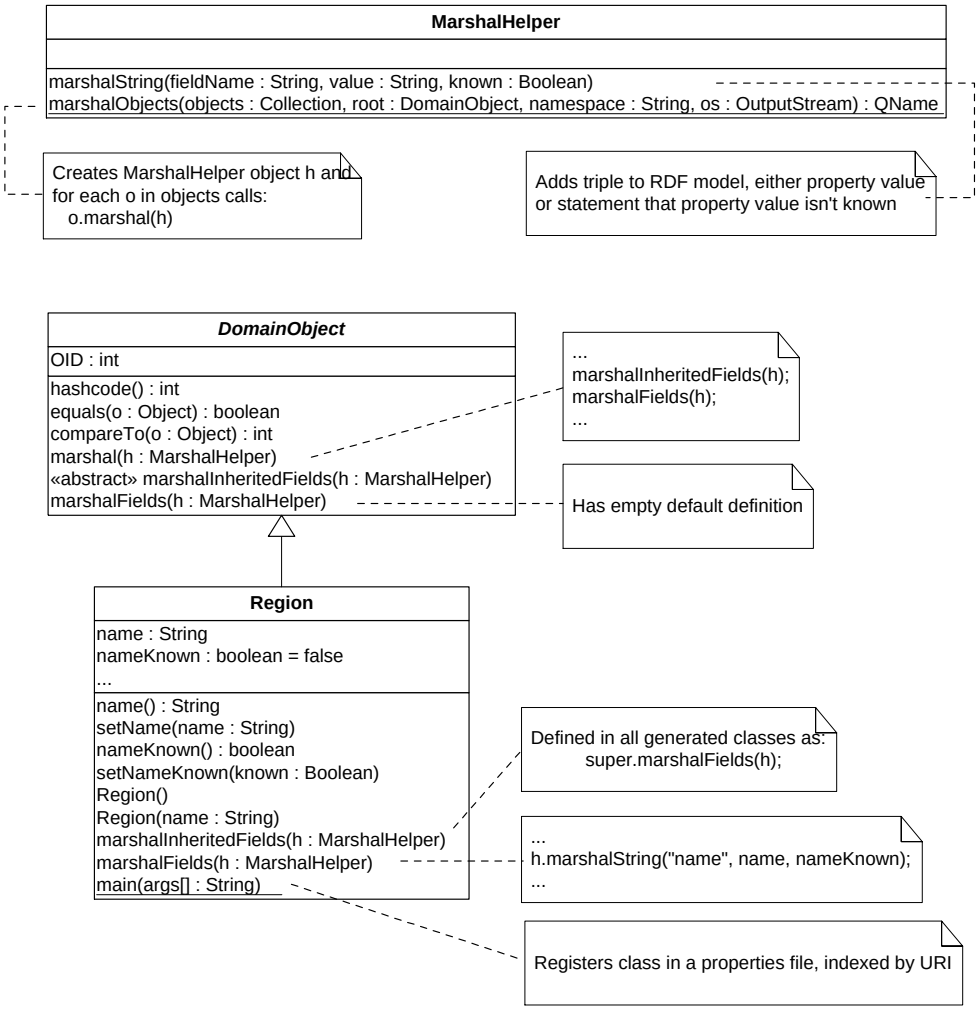


Figure 7: The structure of the generated classes and the marshalling methods

in the serialised model that represents that object. A namespace for the serialised information is also provided.

Figure 8 shows the Java code that would produce the RDF serialisation in Figure 6.

```
// Build object diagram
Region rNZ = new Region("New Zealand");
Country cNZ = new Country();
AreaComparison ac = new AreaComparison();
ac.setCountry(cNZ);
ac.setRegion(rNZ);
ac.setProportion("About the size of");
Set comparisons = new HashSet();
comparisons.add(ac);
Region rColorado = new Region("Colorado");
rNZ.setAs_country(cNZ);
cNZ.setRegion(rNZ);
cNZ.setAreaComparisonSet(comparisons);
// Now marshal it
Set toMarshal = new HashSet();
toMarshal.add(rNZ); toMarshal.add(cNZ);
toMarshal.add(ac); toMarshal.add(rColorado);
try {
  QName rootQName =
    MarshalHelper.marshalObjects(
      toMarshal, rNZ, "http://nzdis.otago.ac.nz/nzdata1#",
      new FileOutputStream("nz.xml"));
}
catch (MarshallingException e) { ... }
```

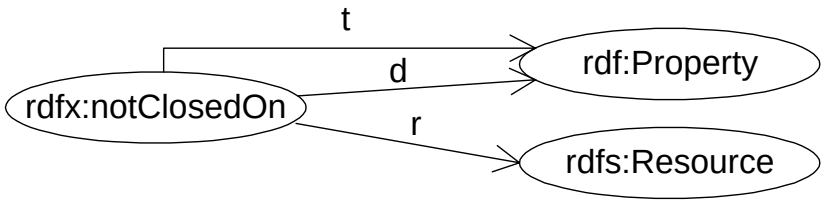
Figure 8: Sample Java code to create and marshal an object diagram

5 Modelling incomplete knowledge

Because object diagrams are inter-linked networks of objects rather than sets of discrete facts, and because classes may have attributes or associations that are optional (i.e. have a multiplicity lower bound of zero), it is important to be able

to distinguish between a statement that there are no values for a given property and the omission or lack or knowledge about a given property. In other words, the recipient of object-oriented information needs a way of knowing for which objects and which properties a closed world assumption can safely be made. This is achieved by including extra boolean fields in the generated Java classes that record for each regular field whether or not its value is 'known' or, for set- or list-valued fields, 'closed'—meaning that the contents of the set or list provide complete knowledge of that field. Setting the value of a single-valued field sets its 'known' field to true and all fields also have a method allowing the programmer to explicitly specify the status of the field.

When unmarshalling an object diagram from the RDF encoding it is assumed that complete information about all properties is included unless otherwise specified (although the opposite could equally well be implemented as the default assumption). Incomplete information is indicated using a non-standard RDF property `notClosedFor` that associates a property with a resource, meaning that complete information is not provided for that property applied to that resource. Figure 9 shows the declaration of the `notClosedOn` property.

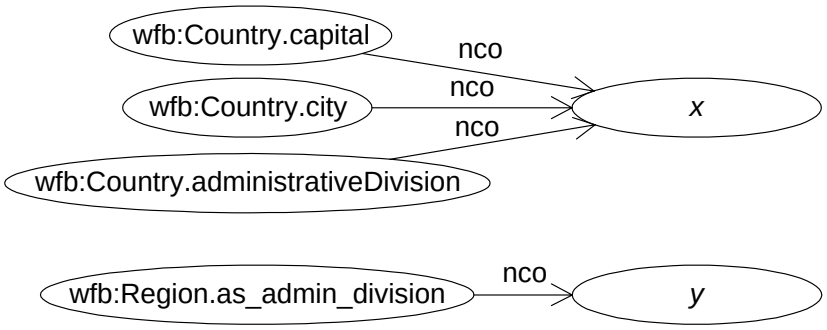


Property labels	Name space abbreviations
t = rdf:type	rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
d = rdfs:domain	rdfs = http://www.w3.org/2000/01/rdf-schema#
r = rdfs:range	rdfx = http://nzdis.otago.ac.nz/0_1/rdf-x#

Figure 9: Schema for the notClosedOn property

Figure 10 presents an example of this property applied to the encoding of knowledge about New Zealand that was shown in Figure 6. When combined with the RDF-encoded information in Figure 6, this specifies that the RDF model does not contain complete (or possibly any) information about the capital, cities and administrative divisions of the the country represented by the resource labelled

x. Also, there is possibly missing information about the administrative division property of the region represented by the resource labelled y.



Property labels	Name space abbreviations
nco = rdfx:notClosedOn	rdfx = http://nzdis.otago.ac.nz/0_1/rdf-x# wfb = *the namespace chosen for the World Factbook schema*

Figure 10: Meta-knowledge about incomplete information

In order to have this meta-level information added to the RDF serialisation, the Java code shown in Figure 8 must have the following lines added before the call to `marshalObjects`.

```
cNZ.setCapitalKnown(false);
cNZ.setCitySetClosed(false);
cNZ.setAdministrativeDivisionSetKnown(false);
rNZ.setAs_admin_divisionKnown(false);
```

6 Reasoning with OCL

The Semantic Web, as envisioned by Tim Berners-Lee [3], includes a logical layer which allows "the deduction of one type of document from a document of another type; the checking of a document against a set of rules of self-consistency; and the resolution of a query by conversion from terms unknown into terms known". One of the biggest challenges for the Semantic Web community is to find interoperable ways of incorporating inference rules into ontologies. There is much research to be done in this area. For example, DAML does not currently support the representation of inference rules, and although OIL 1.0 can express inference rules, the

form of these is not constrained and they are not semantically integrated with the language.

It is therefore an important question to evaluate how well UML fares in this regard. In fact, UML includes a powerful mechanism for expressing inference rules: the Object Constraint Language. OCL is essentially a variant of first-order logic with an object-oriented syntax. It is therefore sufficiently expressive to represent any first-order inference rules that an ontology designer may wish to specify. However, this expressiveness also means that reasoning about unconstrained OCL expressions will be undecidable in general.

The object-oriented syntax of OCL is also unlike any commonly used logical language, and attempting to write rules in OCL can be frustrating for the inexperienced. A constraint can often be expressed in several different ways and the resulting expression can look quite unlike its counterpart in first-order logic. Consider the constraint in Figure 2. The second conjunct specifies that the neighbourhood relationship between countries is reflexive. The form of this constraint might be immediately recognised as a standard pattern by an OCL expert but it is not obvious to the uninitiated.

To enable tractable reasoning about ontologies in UML, and to avoid the awkward syntax of OCL, it would be useful to define a macro language on top of OCL comprising predicates such as `reflexive(`*`path-expression`*`)` which are defined in terms of OCL. The set of macros could be chosen to ensure that reasoning over these expressions is tractable. This would also help to allow the translation of rules between UML-based and other representations of an ontology. This is a subject for future research.

7 Conclusion

This paper has described technology that facilitates the application of object-oriented modelling, and the Unified Modeling Language in particular, to the Semantic Web. From an ontology specified in UML, a corresponding RDF schema and a set of Java classes can be automatically generated to facilitate the use of object diagrams as internal knowledge representation structures and the import and export of these as RDF documents. A mechanism was also introduced for indicating when an object diagram has missing or incomplete knowledge.

Important areas for future work are the identification of tractable subsets of OCL for encoding inference rules and the definition of mappings between object-oriented representations of ontologies and knowledge and more traditional de-

scription logic-based formalisms. This would allow applications to choose the style of modelling most suitable for their needs while retaining interoperability with other subsets of the Semantic Web.

Acknowledgements

This work was done while visiting the Network Computing Group at the Institute for Information Technology, National Research Council of Canada, Ottawa, Canada. Thanks are due to Larry Korba and the NRC for hosting me and to the University of Otago for approving my research and study leave.

References

[1] Robert Cailliau. A little history of the World Wide Web. http://www.w3.org/History.html, 1995.

[2] Laura Carr. 100 numbers you need to know. TheStandard.com, Standard Media International, November 13 2000. http://www.thestandard.com/article/display/0,1151,20128,00.html.

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

[4] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. OIL in a nutshell. In R. Dieng and O. Corby, editors, *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000)*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer, 2000.

[5] DAML project home page. http://www.daml.org, 2000.

[6] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pages 193–238. CLSI Publications, 1996.

[7] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[8] Meta Data Coalition home page. http://www.mdcinfo.com/, 2000.

[9] S. Cranefield and M. Purvis. UML as an ontology modelling language. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-23/cranefield-ijcai99-iii.pdf.

[10] S. Cranefield and M. Purvis. Extending agent messaging to enable OO information exchange. In R. Trappl, editor, *Proceedings of the 2nd International Symposium "From Agent Theory to Agent Implementation" (AT2AI-2) at the 5th European Meeting on Cybernetics and Systems Research (EMCSR 2000)*, pages 573–578, Vienna, 2000. Austrian Society for Cybernetic Studies. Published under the title "Cybernetics and Systems 2000". An earlier version is available at http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2000-07.pdf.gz.

[11] S. Melnik. RDF API homepage. http://www-db-stanford.edu/~melnik/rdf/api.html, 2000.

[12] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The relation between ontologies and schema-languages: translating OIL-specifications in XML-Schema. In *Proceedings of the Workshop on Applications of Ontologies and Problem solving Methods, 14th European Conference on Artificial Intelligence (ECAI 2000)*, 2000. http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/7.pdf.

[13] S. Cranefield. Networked knowledge representation and exchange using UML and RDF. *Journal of Digital Information*, 1(8), 2001. http://jodi.ecs.soton.ac.uk/.