

Generating ontology-specific content languages

Stephen Cranefield and Martin Purvis
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
{scranefield,mpurvis}@infoscience.otago.ac.nz

ABSTRACT

This paper examines a recent trend amongst software agent application and platform developers to desire the ability to send domain-specific objects within inter-agent messages. If this feature is to be supported without departing from the notion that agents communicate in terms of knowledge, it is important that the meaning of such objects be well understood. Using an object-oriented metamodelling approach, the relationships between ontologies and agent communication and content languages in FIPA-style agent systems are examined. It is shown how object structures in messages can be considered as expressions in ontology-specific extensions of standard content languages. It is also argued that ontologies must distinguish between objects with and objects without identity.

1. INTRODUCTION

Agent communication languages (ACLs) in the style of the Knowledge Query and Manipulation Language (KQML) [15] and FIPA ACL [17] are based around the idea that “Communication can be best modelled as the exchange of declarative statements” [19]. Under this paradigm, agents send, receive and reply to requests for services and information with the intent of the message specified by a performative (such as ‘inform’ or ‘request’) describing the way in which an inner content expression should be interpreted. The content is encoded using a declarative knowledge representation language such as the Knowledge Interchange Format (KIF) [21] or FIPA’s Semantic Language (SL) [18]. In addition, the ACL specifies any ‘ontologies’ that define the terminology used to denote domain-specific concepts inside the message content.

This paper examines a recent trend amongst software agent application and platform developers to desire the ability to send domain-specific objects within inter-agent messages. If this feature is to be supported without departing from the notion that agents communicate in terms of knowledge, it is important that the meaning of such objects be well understood. To clarify this issue, this paper uses the Unified Modeling Language [4] and an object-oriented metamodelling approach to examine the relationships between ontologies and agent communication and content languages. It is shown how object structures in messages can be considered as expressions in ontology-specific extensions of standard declarative content languages. In particular, domain-specific ‘objects’ within messages can be viewed as object-oriented encodings of terms, propositions or definite descriptions. To make the discussion more concrete, the proposed approach is illustrated in terms of a particular model of agent communication: that defined by the Foundation for Intelligent Agents (FIPA) specifications [16]. However, this account of domain objects within messages is applicable to other styles of declarative-message-passing agent systems as well.

2. INCLUDING OBJECTS IN MESSAGES

Traditionally ontologies are used in agent systems “by reference”. An agent is not required to explicitly reason with the ontology, or even to have an online copy available. The names of ontologies can simply be used as a contract between agents undertaking a dialogue: they each claim to be using an interpretation of the terms used in the conversation that conforms to the ontology. The content language uses a string-based syntax to represent sentences in the language which are constructed using constants and function and predicate symbols from the ontology as well as built-in language symbols such as “and” and “or”.

However, the popularity of the Java programming language for agent development, as well as the increasing use of XML-based formats for serialising structured data, appear to be causing agent developers to look for ways of creating outgoing messages and analysing incoming ones that are more in line with the object-oriented paradigm than the traditional approach of building and parsing strings. This is evidenced by conversations between the authors and local agent researchers, messages on the FIPA agentcities mailing list, and the feature introduced into the JADE agent platform allowing application-specific Java classes to be associated with concepts in an ontology.

JADE [14] allows ontologies to be defined using a frame-based language. An ontology is defined at run time by constructing an ontology object and adding frames to it. A Java class can be associated with a frame so that an application can create instances of the frame as Java objects and insert them in the message content. The JADE messaging system then handles the serialisation of the objects, which can be customised by users for particular content languages.

Even in the traditional string-based model of message creation and decomposition there are signs that agent programmers desire an ability to send ‘objects’ within messages. The FIPA SL grammar includes a production for “functional terms”. In addition to some built-in function symbols for arithmetic and for constructing and performing operations on sets and sequences, functional terms can be built using function symbols from an ontology. The SL specification describes two uses for ontology-specific functional terms.

The first use applies when an ontology defines function symbols specific to its domain. Objects can then be referred to “via a functional relation . . . with other objects . . . rather than using the direct name of that object, for example, (*fatherOf Jesus*) rather than *God*”. This can be interpreted using the standard semantics for first-order logic—terms of this form correspond to descriptions

of particular pre-existing objects. In a first order logic without an equality symbol, a functional term denotes a particular object in the domain of discourse that is distinct from the object denoted by any other term. Adding an equality symbol introduces the possibility of making assertions that distinct terms in the language are equal, meaning that they denote the same domain object. The usual semantic models used to define first order logics with equality interpret terms by equivalence classes in the ‘term algebra’ with respect to the equality predicate. These equivalence classes can be considered to be an abstract representation of the actual domain objects represented by terms in the language.

The second use for ontology-specific functional expressions involves “descriptions where the function symbol should be interpreted as the constructor of an object, while the parameters represent the attributes of the object”. The following example is given:

```
(vehicle
  :colour red
  :max-speed 100
  :owner (Person :name Luis
           :nationality Portuguese))
```

The SL specification describes this usage as one where “the function symbol should be interpreted as the constructor of an object, while the parameters represent the attributes of the object”. However, this notion of constructor is not explained, nor is any guidance given on when a term of this form is considered to be well-formed. A number of questions are raised with this notation:

- *What constitutes a constructor in an ontology?* For frame-based and object-oriented ontology modelling languages the answer is fairly clear, but is there a set of requirements that could be used to identify constructors in other types of ontology modelling language? This paper focuses on the use of object-oriented ontologies and does not address this question for other modelling paradigms.
- *Does each appearance of a given constructor term denote the same object, or does every use of a constructor create a new object?* Allowing the universe of discourse to grow dynamically departs from the usual semantics of first order logic and would require a thorough investigation of its implications on the semantics of SL. This paper therefore attempts to classify the possible uses of ontology-specific functional terms without abandoning the usual closed (but possibly infinite) world model underlying first-order logic.
- *Can a meaning be given for such a functional term representing an object with identity?* One possible interpretation is that objects can be passed from one agent to another. However, this requires additional theory and infrastructure to model the dynamic association of objects to the agents responsible for them and to ensure the preservation of objects’ uniqueness or to account for their duplication and possible destruction. This is left for future work. For the present we prefer to retain the traditional view that the message content is a declarative representation of knowledge about the world.

We address these issues by proposing that a distinction be made between objects with and objects without identity, as in the Object

Query Language [5]. Our account of agent messaging does not allow for objects with identity to be transported within messages—it is only meaningful for objects *without* identity (i.e. complex data values) to be denoted by functional terms in a content language. For concepts representing classes of objects with identity, we regard functional expressions such as the `vehicle` one above to be invalid as a term denoting a domain object. Instead, we propose that this type of functional expression should be meaningful only as a form of proposition (describing some or all of an object’s attributes) or as a definite description (identifying an object in terms of the values of attributes). In both cases, the notation can be generalised to include networks of objects and (in the case of definite descriptions) variables.

The key difference between objects with and without identity is that it is possible to refer to the former by reference. We make no assumptions about the form of object references, but to be included within an agent message, a reference must be expressible as a ground term within some content language. Many different reference schemes could be used for referring to objects: names, CORBA interoperable object references (IORs), World Wide Web Uniform Resource Identifiers (URIs), etc. This is discussed further in Section 4.1. Note that we do not explicitly address the possibility of a reference being used to refer to different objects by different agents or at different times. Sufficient namespace or context information to resolve this problem is assumed to be included as part of a reference.

Whether or not instances of a given concept are considered to have identity will depend on the range of applications for which the ontology is designed. To be completely general, any object could potentially be referred to in an assertion—this is one of the design criteria for the Semantic Web where “anyone can say anything about anything” [3]. However, for purposes of simplicity and efficiency, it may sometimes be advantageous to declare concepts to be “value types”.

The difference between objects with and without identity can be seen in the following example. The FIPA agencities project [1] is building a test-bed for the large-scale deployment of FIPA agent-based services by forming a global network of agents providing information relating to particular cities. One of the ontologies under development in this project describes the concept of weather reports. A weather report is an example of a plausible value type. Although it is possible to imagine scenarios where agents may wish to refer to a weather report by reference (e.g. to disagree with it), in many common cases it would be sufficient to have the ability to include weather report structures within messages. Note that this would not mean that weather reports could not be recorded in an agent’s knowledge base—another object such as an array could hold weather reports internally as part of its own structure.

In contrast, a city is a clear example of an object that has identity and which could not be meaningfully included within a message.

3. A METAMODELLING VIEWPOINT

To analyse the relationship between ontologies and content languages expressions, this paper extends a metamodelling account of agent systems and models presented previously [13]. Figure 1 shows three levels of a metamodelling hierarchy. Level 0 comprises the objects that exist at run time: agents, domain objects and messages and their content expressions. Level 1 contains models of the concepts that are instantiated in Level 0, i.e. ontologies and defini-

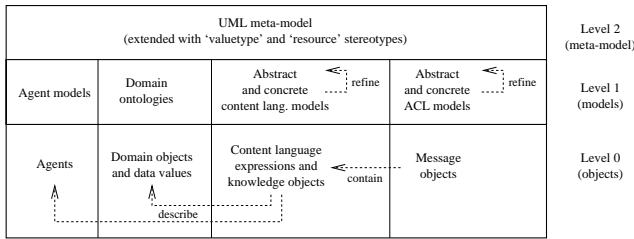


Figure 1: A metamodeling view of agent systems and models

tions of the abstract and concrete syntax for agent communication and content languages. The relationship between a Level 0 object and the corresponding Level 1 object is instantiation (or equivalently set membership if concepts are viewed semantically as sets of instances).

Note that analysing agent systems using the metamodeling view-point does not presuppose that agents are implemented using object-oriented technology. The same relationships between and across levels hold no matter how agents and languages are modelled and instantiated. Furthermore, Figure 1 could be extended to account for other multi-agent system concepts, such as conversations, which could be explicitly represented at Level 0 with conversation models at Level 1.

The Level 1 models must be expressed in some language. In this work, the Unified Modeling Language (UML) [4] is used as a unifying representation language for all agent-related models at Level 1. It has been argued elsewhere that this industry-standard object-oriented modelling language is a good candidate for representing ontologies [11, 12, 2, 9] and the abstract syntax of agent communication languages [13, 10]. Level 2 therefore contains the UML metamodel: a model of the concepts in UML. Other “metamodels” could also be included at this level to allow for different representation languages to be used at Level 1.

Figures 2 and 3 illustrate fragments of an agent communication language and a content language (respectively) modelled as UML class diagrams. The UML ‘lollipop’ symbol is used to indicate that particular classes implement interfaces from another model: the CL package shown in Figure 4. This package uses ‘marker’ interfaces (ones with no operations) to name the abstract concepts that any content language must implement in order to be used with the FIPA ACL. This abstraction provides a strongly typed way for expressions in any content language to be included within an ACL expression. For example the Object Query Language could be used as a form of definite description simply by defining a class `ObjectQueryDescriptor` with a string-valued attribute query and declaring it to implement `CL::DefDescriptor`.

The CL package (Figure 4) models abstract content language concepts that are required by the semantics of the FIPA ACL and its communicative act library, but otherwise makes no commitment about how particular content languages might be structured. For example, the concept of a variable is included because this seems to be intrinsic to the semantics of definite descriptions [23]. However, the concept of a predicate is not included. Although some FIPA communicative acts require propositions as parameters, the ACL is neutral about how propositions are represented—a content language is free to represent propositions in non-standard ways, e.g. as object structures asserting the attribute values [13].

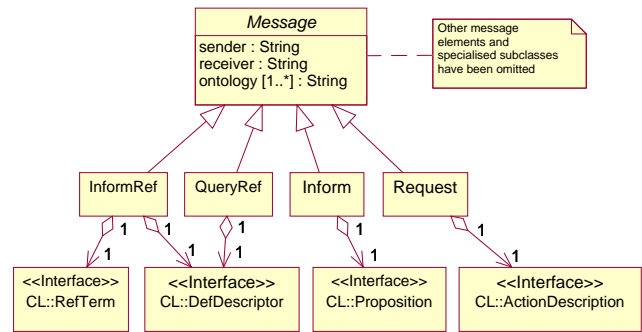


Figure 2: A partial UML model of an ACL

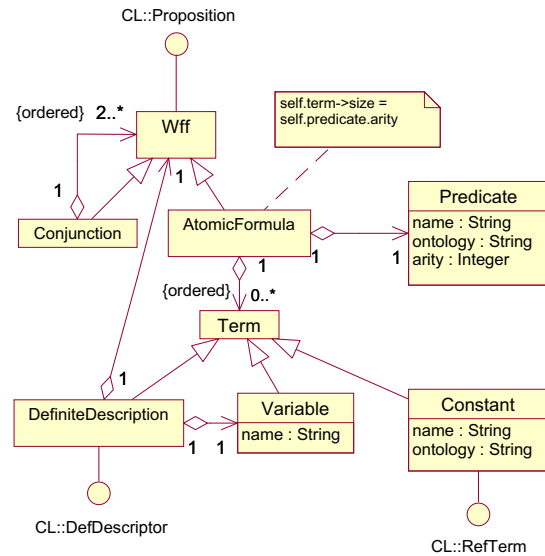


Figure 3: A partial UML model of an SL-like content language

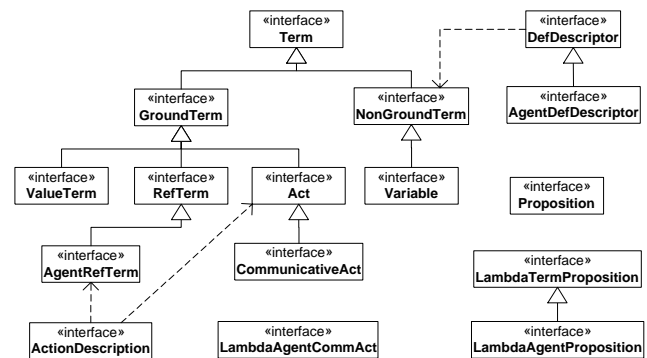


Figure 4: The CL package: generic content language concepts

In the figure, the dashed lines represent dependencies: for example, an action description requires descriptions of an act and an agent, but at this abstract level it is not appropriate to make any decisions about how the implementation structure of these three concepts should be related. A dependency is shown from `DefDescriptor` to `NonGroundTerm` because a definite description comprises two non-ground terms linked by common free variables: a query expression and a template for the result.

Some of the content language concepts shown are not explicitly named in the FIPA specifications: for example, some communicative acts (such as `proxy`) require content expressions representing communicative acts with the receiver (an agent) omitted. This is modelled by the interface `LambdaAgentCommAct`. The interface `LambdaTermProposition` represents a function that produces a proposition when a single parameter is supplied (used in the ‘call for proposals’ communicative act) and `LambdaAgentProposition` is a specialisation of this where the parameter is an agent reference (this concept is not currently used by FIPA ACL, but seems likely to be useful).

4. ONTOLOGIES AND OBJECT IDENTITY

In Section 2 it was argued that a distinction should be made between objects with and objects without identity. The approach taken in this paper is to make this distinction on a per-concept basis, with the ontology declaring for each concept whether its instances have identity or not, and how references to instances of that concept can be expressed. In this section we show how the extension mechanisms of UML can be used to model these aspects of an ontology.

4.1 Modelling references

The object model underlying UML does not take a position on the nature of object identity and reference. UML assumes that objects have an identity that is implicitly provided by the underlying implementation infrastructure and which does not need to be included as an attribute of the corresponding class in the model. UML makes no commitment about the nature of this identity or the way in which links between objects are implemented. This is too agnostic a stance for use in the description of multi-agent systems, at least for those based on the Foundation for Intelligent Physical Agents (FIPA) [16] specifications. One of the FIPA communicative acts is `query-ref` which is used to ask an agent for a reference to the object that uniquely satisfies a given “definite description” (a quantified propositional expression, where the bound variable represents the object of interest). The semantics of definite descriptions assume each object (taken in a broad sense to mean an identifiable entity) is identified by constants representing “standard names”. This is sufficient for agent systems that are disconnected from the real world and observe it without interacting with any of its entities. However, when agents are integrated into today’s pervasive network infrastructure, which includes telephone networks, distributed CORBA and Java objects and Web resources, this is no longer sufficient. Objects can be identified by phone numbers, CORBA interoperable object references (IORs), World Wide Web uniform resource identifiers (URIs), etc., and agents are free to depart from ACL-level communication to interact with these objects using the appropriate protocols for these references. We therefore propose an extension to the UML metamodel to allow classes to be annotated with their appropriate reference types.

Although the metamodel for UML is fixed, there is a way to define a ‘virtual’ extension of the metamodel by defining *stereotypes*: named specialisations of particular concepts (such as ‘class’) from

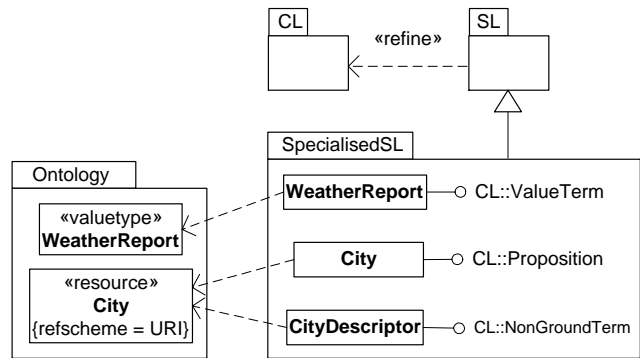


Figure 5: An ontology-specific content language

the metamodel. Model elements can be endorsed with the names of stereotypes within guillemets (French quotation marks) to indicate that they have a different intent or additional semantics beyond that normally associated with that type of model element.

For modelling classes of objects with identity we have introduced a stereotype `«resource»`. This represents a specialised subtype of the class concept in the UML metamodel, one whose instances (particular classes in a model) are constrained to include a one to many relationship with a `Reference` abstract class that is assumed to exist at the model level (Level 1) and to implement the `CL::RefTerm` interface. In an ontology, adorning a class with this stereotype is the same as manually adding the association between that class and the `Reference` class. It is therefore possible for any instances of that class to be linked to instances of particular reference classes such as `URI` (note that we do not assume there is a unique reference to each object). To provide more information about the particular type of reference, another UML extension mechanism, a tagged value, is used to name the type of reference used for objects of that class. The tag name is “refscheme”. The value of this tag should be the name of a predefined class extending `Reference`.

The bottom left side of Figure 5 illustrates the use of these mechanisms to specify that the class `City` represents resources having URIs (in the style of the Semantic Web where URIs can be used to refer to physical objects as well as Web documents). The rest of the figure will be explained in the following sections.

4.2 Modelling data values

To represent value types another stereotype, `«valuetype»` is introduced. Model concepts defined with this stereotype are not modelled as classes, but as instances of a different child of the `Class` concept’s parent class in the UML metamodel: `ValueType`. This is not a standard concept in UML but an extension that is required to adequately model objects without identity. UML has a similar concept `DataType`, but this has restrictions that seem to be unnecessary for value types: a data type may not have any outgoing associations with other types. However, it seems reasonable for a value type to have composition relationships with other value types as these do not require references for their implementation—they can be interpreted as defining a nested structure.

In Figure 5, the `WeatherReport` class is annotated with the stereotype `«valuetype»` to indicate that this represents a type of object without identity.

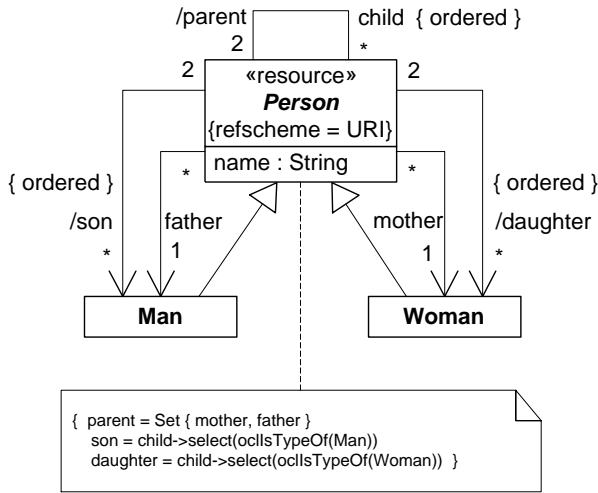


Figure 6: A class diagram representing a family ontology

5. ONTOLOGY-SPECIFIC CONTENT LANGUAGES

Figure 1 showed a number of relationships between different types of object: messages contain content language expressions, which describe domain objects and agents. At the model level, ACL and content language concrete syntaxes (e.g. models of XML encodings) refine abstract syntaxes. However, there is currently no clear account of the relationship between ontologies and content languages that explains how and when it makes sense to include domain-specific objects within messages. Strictly speaking, there is no direct connection between concepts in ontologies and those in content languages: as discussed in the introduction, content expressions traditionally refer to ontological entities by reference (i.e. by name) only.

The inclusion of domain-specific objects within messages can be considered either to be semantically incoherent, or (more charitably) as representing an object-oriented encoding of values, propositions and definite descriptions. This paper takes the latter approach, which can be explained as the use of an ontology-specific content language. Given an ontology, a specialised content language can be generated either as a simple application-specific representation or as an extension of an existing general-purpose content language (so that generic concepts like conjunction are available).

Figure 5 illustrates our approach to generating ontology-specific content languages. The dashed arrows are dependencies and therefore are directed from the generated classes back to the corresponding concepts in the ontology. The generalisation relationship between the generated Specialised SL package and the pre-existing SL one indicates that all concepts defined for SL (such as conjunction and negation) are (optionally) included in the generated package.

A valuetype declared in an ontology maps to an equivalent declaration (without the valuetype stereotype) in a new package corresponding to the generated language. This is because objects of that type can meaningfully be embedded in messages.

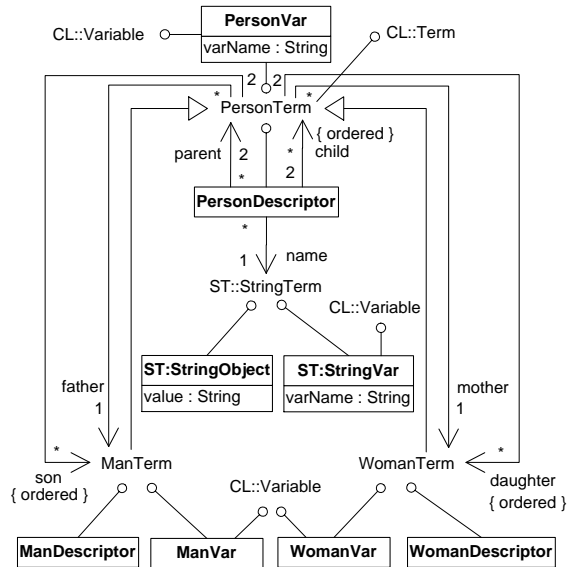


Figure 7: Generated descriptor classes for Person objects

A class with the resource stereotype maps to a structurally identical class that implements the `CL::Proposition` interface—this indicates that objects of this sort can be used as descriptions of corresponding domain entities (by describing their attribute values and links). In addition, a corresponding descriptor class is generated to allow an object-oriented structure to be used as the query part of a definite descriptor, therefore this class implements the interface `CL::NonGroundTerm`.

A class with no stereotype has a corresponding proposition class generated, but no definite descriptor class is generated as nothing is known about whether or not objects of this type have identity. It may therefore be impossible to return a reference in response to a `query-ref` message containing a definite description referring to an instance of this class.

Figure 7 shows the structure of the generated descriptor classes corresponding to a simple family ontology (expressed as a UML class diagram in Figure 6). Each attribute and navigable association end must be capable of taking a variable object as a value. Therefore, for each class an interface is defined and two subclasses that implement it: a descriptor class and a variable class. All association ends are directed at the interfaces instead of the classes. The package `ST` is assumed to hold variable and object types corresponding to each primitive type.

6. IMPLEMENTATION

The previous section presented a scheme by which extensions of content languages like FIPA SL can be automatically augmented with ontology-specific representations for propositions, definite descriptions and (for valuetypes) terms. It remains to formally define and implement these mappings. In previous work, a “UML data binding scheme” has been developed [7, 8]. This allows agent messages to be visualised as object diagrams and serialised using the XML-based Resource Description Framework [20]. The serialisation is performed with reference to RDF schemas that are generated from the UML definitions of the ontologies used and the

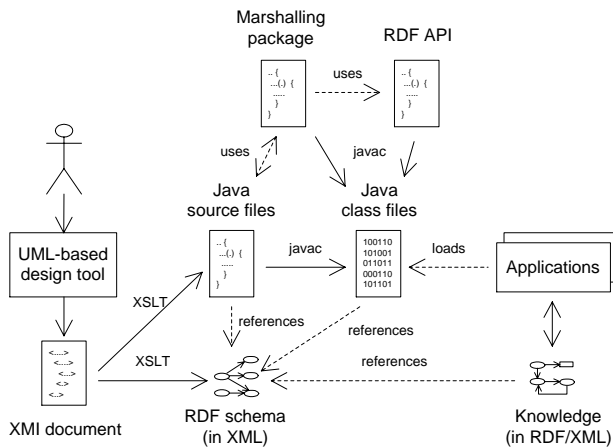


Figure 8: The UML data binding framework

ACL and content language. A set of Java classes corresponding to the concepts in these models are also generated, and these include marshalling support so that object diagrams can be converted between in-memory networks of Java objects and RDF serialisations with a single method call. The generation of RDF schemas and Java classes is performed using the Extensible Stylesheet Language Transformations language [24] starting from encodings of the UML models in the XML Model Interchange format (XMI). The UML data binding scheme is illustrated in Figure 8.

At present, classes in ontologies are mapped directly to RDF resources with properties that correspond directly to the attributes and association ends associated with the classes. Extending this to implement the mappings described in the previous section would provide agents with an automatically generated Java application programmer interface for including domain-specific objects within messages—but only when they represent propositions, definite descriptions or values of value types.

There is one feature of the UML data binding scheme that has not yet been addressed in the present work. If propositions are to be encoded as object structures, it must be possible to omit the values of some attributes and association ends if these are not of relevance to the discussion—otherwise an agent may have to send a network representing all its knowledge every time it answers a question. If a value is not present for a role of an optional association then the agent receiving the information must be able to distinguish between the case where there really is no value and the case where this information has been omitted. This is handled by declaring in the RDF serialisation of a message the property-resource pairs for which incomplete or no information is provided [8]. This, or an alternative mechanism, needs to be accounted for in the conceptualisation of knowledge as a UML object diagram. One possible approach would be to use a stereotype or tagged value to indicate potentially incomplete knowledge.

Although XSLT provides a convenient language for implementing transformations on XML-based data, it is debatable whether an XSLT stylesheet can be considered to be a *definition* of a mapping. One possible way in which the mapping from ontologies to specialised content languages could be defined is based on research from the Precise UML group [22]. Figure 9 shows how OCL constraints can be used to define the mapping between ab-

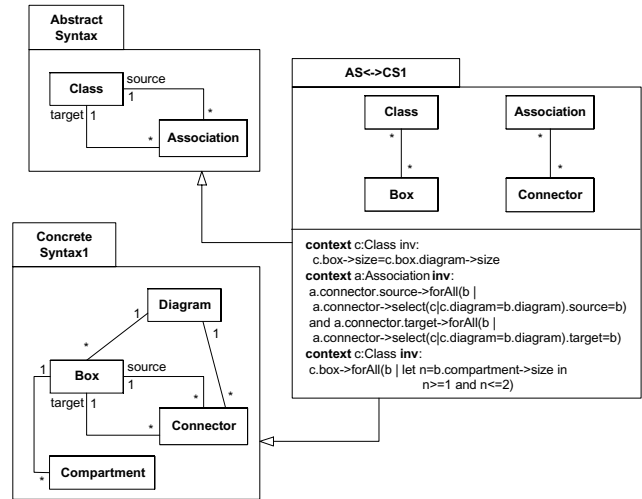


Figure 9: The pUML approach to mapping between abstract and concrete language models (modified from Clark *et al.* [6])

stract and concrete syntaxes for a language. This technique may provide a formal way to define the relationship between ontologies and ontology-specific content languages described in this paper.

7. CONCLUSION

This paper has analysed the relationship between ontologies and content languages and, based on a desire from agent programmers to include ‘objects’ within messages, has clarified when this can be considered to be meaningful. This ability can be provided in a principled way by generating an ontology-specific content language from an ontology.

It has also been argued that ontologies should distinguish between object types having an identity and those without. Two UML stereotypes have been proposed to indicate this distinction, which also affects whether domain-specific object structures in an ontology-specific content language can be used to represent propositions, values or definite descriptions.

Acknowledgements

Thanks to the members of the FIPA agentcities mailing list for stimulating our thoughts on this topic, especially Federico Bergenti who espoused the need for a “metamodel” for content languages and thereby inspired the design of the abstract content language model shown in Figure 4.

8. REFERENCES

- [1] agentcities.org. The Agentcities project Web site. <http://www.agentcities.org>, 2001.
- [2] F. Bergenti and A. Poggi. Exploiting UML in the design of multi-agent systems. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents World*, Lecture Notes in Computer Science 1972, pages 106–113. Springer, 2000. (an earlier version is available at <http://lia.deis.unibo.it/conf/ESAW00/pdf/ESAW13.pdf>).
- [3] T. Berners-Lee. Metadata architecture. World Wide Web Consortium Discussion Document, 1997. <http://www.w3.org/2000/01/sw/>.

