The Knowledge Engineering Review, Vol. 17:1, 21–39. © 2002, Cambridge University Press
DOI: 10.1017/S0269888902000309 Printed in the United Kingdom

A UML profile and mapping for the generation of ontology-specific content languages*

STEPHEN CRANEFIELD and MARTIN PURVIS

Department of Information Science, University of Otago, PO Box 56, Dunedin, New Zealand;
e-mail: scranefield @infoscience.otago.ac.nz, mpurvis @infoscience.otago.ac.nz

Abstract

This paper examines a perceived desire amongst software agent application and platform developers to have 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 defined. 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 argued that for use with distributed multi-agent systems, ontologies should describe the nature of object identity and reference for each defined concept, and a UML profile supporting these modelling capabilities is presented. Finally it is shown how, given an ontology in UML, an ontology-specific object-oriented content language can be generated, allowing object structures (viewed in the abstract as UML object diagrams) to be used within message content to represent propositions, definite descriptions or (for classes without identity) value expressions.

1 Introduction

Agent communication languages (ACLs) in the style of the Knowledge Query and Manipulation Language (KQML) (Finin *et al*., 1997) and the Foundation for Intelligent Physical Agents (FIPA) ACL (FIPA, 2000a) are based around the idea that "communication can be best modelled as the exchange of declarative statements" (Genesereth & Ketchpel, 1994). 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) (NCITS, 1998) or the FIPA Semantic Language (SL) (FIPA, 2000c). In addition, the ACL specifies any "ontologies" that define the terminology used to denote domain-specific concepts within the message content.

This paper examines a perceived desire amongst software agent application and platform developers to have 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 declarative knowledge, it is important that the meaning of such objects be well understood. To clarify this issue, we examine the relationships between ontologies and agent communication and content languages. To keep the discussion concrete, we focus on a particular choice of technologies: FIPA-style agent communication and the use of the Unified Modeling Language (UML) (Booch *et al*., 1998) for modelling both ontologies and the abstract syntax of languages. However, despite these specific

* 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 3.

choices, we believe the analysis is more general and this account of object-like expressions within messages is applicable to other styles of declarative-message-passing agent systems.

One significant feature of the FIPA ACL is the `query-ref` communicative act which is used to ask an agent to "inform the requester of the object identified by a descriptor". The recipient is supposed to respond with an `inform` message "containing the object[1] that corresponds to the descriptor" (FIPA, 2000b). This raises the question of what sort of term can be used to identify an object. The current FIPA ACL and SL specifications do not provide a clear answer to this question. The FIPA ACL semantics (FIPA, 2000b) assume that each object is identified by a constant representing a "standard name" (Sadek, 1990), and this provides a way of identifying objects by *reference* (assuming the agents have a common understanding of the referents of the standard names). Also, examples in the FIPA Communicative Act Library Specification (FIPA, 2000b) suggest that some types of object can be identified by *value*: one example shows a string literal being returned in response to a `(request ... (inform-ref ...))` communicative act (which is equivalent to a `query-ref`) while the example for `query-ref` itself shows a response containing an SL functional term representing a set of agent service descriptions.

Both references and value objects may, in different circumstances, be legitimate terms identifying the result of a `query-ref`, but how can a system designer (or an agent itself) determine what is a semantically meaningful response for a given query? A key aim of the work described in this paper is to answer this question by providing a strongly typed account of object-identifying and object-describing terms. In particular, we make a distinction between objects with and objects without identity, and we disallow terms *denoting* objects with identity from appearing within messages – this is because current ACL semantics do not support the idea of objects being moved or copied. However, this is not as restrictive as it might sound. Messages may include functional terms that denote *propositions about* and *descriptors of* objects with identity, and in this paper we show how the abstract syntax of such terms can be automatically generated from the ontology.

We also claim that the question of whether or not an object has identity is a fundamental property of the *concept* of which it is an instance, rather than being something that may vary on a per-instance basis. We therefore believe that ontology modelling languages need to provide a way of annotating concepts in an ontology to provide this information. Furthermore, for classes of objects with identity it should also be possible to declare the types of reference that may be used to refer to objects of that class. While standard names may be a sufficient form of object reference for agent systems that are disconnected from the real world and observe it without interacting with any of its entities, in reality agents are likely to be integrated into today's pervasive network infrastructure, which includes telephone networks, distributed CORBA and Java objects and Web resources. Objects can be identified by phone numbers, CORBA interoperable object references (IORs), World Wide Web uniform resource identifiers (URIs) and so on, and agents are free to depart from ACL-level communication to interact with these objects using the appropriate protocols for their references.

To address the above requirements in the context of ontology-modelling using UML, we present a UML *profile* (a set of extensions to the language) allowing the explicit modelling of concepts with or without identity, reference types and references. This, in turn, provides a sufficiently rich ontological representation for us to complete our account of how object structures in messages can be considered as expressions in ontology-specific extensions of standard declarative content languages. We define a mapping from ontologies defined using this profile to the abstract syntax (also in UML) of corresponding ontology-specific content languages. Under this mapping, a set of interrelated classes in an ontology maps to two corresponding sets of classes having similar (but not identical) structure and relationships: (a) descriptor classes, for use with `query-ref` messages, and either (b) object-oriented proposition classes, for use with `inform` messages (in the case of objects with identity) or (c) classes representing structured values, for use in the answer to a `query-ref` (in the case of objects without

[1] FIPA uses the word "object" in a broad sense to mean any identifiable entity.

identity). Although this mapping specifically produces a UML-based content language, it also helps to explain the possible meanings of ontology-related functional expressions in FIPA SL.

The structure of the paper is as follows. Section 2 discusses the current trend towards including object-like structures within the content of agent messages and raises some questions about the possible meanings of such objects. To help answer these questions, Section 3 presents a metamodelling framework that highlights the relationships between various models (such as ontologies and the abstract syntaxes of an ACL and content languages) that are needed in the design of an agent system. In Section 4 we show how we use UML to represent ontologies and the abstract syntax of agent communication languages. Section 5 presents a UML profile that allows ontologies to make a distinction between concepts with and without a notion of identity. In Section 6 we describe a mapping from ontologies defined in terms of this profile to specialised object-oriented content languages modelled using UML, and Section 7 discusses how the mapping can be used to provide an automatically generated application programmer interface for agent programmers to handle message content. Section 8 concludes the paper.

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 declare that they are using an interpretation of the terms used in the conversation that conforms to the ontology. The ontology provides a vocabulary which defines constant, predicate and function symbols that can be used within a logic-based content language. The content language uses a string-based syntax to represent sentences in the language, which are constructed from the symbols in 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. With this style of message processing, ontologies are seen as implicity or explicitly defining types that can be used to express content language components. This is evidenced by conversations between the authors and local agent researchers, messages on the FIPA agentcities working group mailing list,[2] and the feature introduced into the JADE agent platform[3] allowing application-specific Java classes to be associated with concepts in an ontology.

JADE allows ontologies to be defined using a frame-based language. An ontology is constructed dynamically at run-time by code that creates an instance of a generic ontology class and adds frame objects 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 (FIPA, 2000c) 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, `(father Of Jesus)` rather

[2] `http://www.fipa.org/mailman/listinfo/agentcities/`, now superseded by `http://www.agentcities.org/mailing-lists/listinfo/discussion`.
[3] `http://sharon.cselt.it/projects/jade/`.

than `God`". This form of functional term seems most applicable when using a traditional logic-based ontology modelling language and does not presuppose an object-oriented representation of messages (although there is an object-oriented equivalent: a functional relationship can be represented by an association between two classes with a multiplicity of "1" at one end of the association). In contrast, the second type of functional term described in the SL specification is clearly oriented towards object-oriented representation – it is used for "descriptions where the function symbol should be interpreted as the constructor of an object, while the parameters represent the attributes of the object". An example of the following form is given (FIPA, 2000c):

```
(vehicle
  :colour red
  :max-speed 100
  :owner (Person :name Stephen
                 :nationality NewZealander))
```

The notion of a "constructor" function appearing within a content expression is problematic as it suggests that objects can be constructed and sent by value from one agent to another agent within a message. This does not coexist well with the traditional view that message content is a declarative representation of knowledge about the world (as viewed by the agent that sent the message), and therefore should have a well-defined interpretation independent of any knowledge of the agent that sent the message. Denoting objects by value is not supported by the semantic model underlying FIPA communication (Sadek, 1990) in which objects are effectively identified by reference: using "standard names" (constants from a set that is assumed to cover the interpretation domain), functional terms of the first form discussed above (describing objects as functions of other objects) or referring expressions such as *definite descriptions* – expressions of the form $\iota x\, p(x)$ that refer to the unique[4] object satisfying a given predicate p (Russell, 1956).

While it may be a useful extension to current agent communication languages to allow objects to be sent by value, we believe there are some of theoretical and infrastructural issues that need to be resolved before this feature can be offered in a principled way by an agent communication language:

- Do we require a distributed object infrastructure that allows references to remain valid after the objects have been moved between hosts, as provided, for example, by CORBA (OMG, 2002)
- How can the semantics of agent communication be extended 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?

These issues are left for future work. For the present we note that the difficulty lies solely with the inclusion within messages of functional terms denoting objects having *identity* – those that can be distinguished from other objects even if their contents are identical. Not all objects need have this property, for example, the Object Data Management Group (ODMG) object model makes a distinction between objects with and objects without identity (Cattell *et al*., 2000). Objects with identity can have references while objects without identity represent simple structured values that cannot be passed by reference.

We consider that the question of whether instances of a given concept have identity is a fundamental property of that concept, and is therefore part of the process of designing an ontology. For a given concept, whether it should be modelled as having 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" (Berners-Lee, 1997). However, for purposes of simplicity and efficiency, it may

[4] The FIPA Communicative Act Library (FIPA, 2000b) lists two other types of referring expression: descriptors referring to (respectively) *any* object or *all* objects that satisfy a given proposition. Our approach extends readily to accommodate these.

sometimes be advantageous to declare concepts to be "value types", using CORBA's terminology for objects without identity (OMG, 2001).

The difference between objects with and without identity can be seen in the following example. The Agentcities project (Wilmott *et al*., 2001) 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. It is likely that an ontology in this domain will contain many concepts (hotels, restaurants and so on) that have a street address. The concept of a street address is a plausible example of a value type. It is unlikely that an agent would need to refer to a street address by reference (independently of any object located at that address) – it would be sufficient to have the ability to include street address structures within messages. In contrast, a city is a clear example of an object that has identity and which would have to be represented within a message by a reference.

Making this distinction allows us to identify the semantically problematic type of ontology-specific functional expression in SL: expressions that directly denote objects with identity. In our approach to agent messaging we choose to disallow such expressions from appearing within the content of ACL messages. However, we do support three other meaningful types of domain-specific functional expression:

- terms representing structured values with no identity,
- representations of predicates asserting information about existing objects having identity. and
- patterns forming the body of referring expressions such as definite descriptions.

For example, as vehicles have identity, our approach does not allow the `vehicle` expression above to be used as a *term* within a message. However, an expression of this form could be considered to play the role of a proposition about a particular car (when extended to include a reference to the car being described) or the role of a definite description of a car of interest.

In this paper we provide a principled account of the use of domain-specific expressions for representing structured values, propositions and definite descriptions. We generalise from the S-expression syntax of FIPA SL to a more abstract syntax where we consider ontology-specific content language expressions to be object structures that are instances of classes generated from (and closely related to) the classes defined in an ontology. This is based on the observation that content language expressions represent knowledge or queries about domain objects – they are not generally objects of discussion in their own right. A proposition describing one or more existing objects and the relationships between them can be represented as an object network listing some (but not necessarily all) of the objects' attribute values and the links between them. A similar structure containing one or more variables can form a pattern representing the body of a referring expression such as a definite description. These structures do not denote domain objects directly and therefore cause no semantic difficulties.

3 A metamodelling viewpoint

To clarify the issues discussed in the previous section it is useful to analyse the relationship between ontologies and content language expressions. In this section we present a metamodelling account of agent systems and models (elaborated from a previous account (Crenefield *et al*., 2000)). 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, e.g. ontologies and definitions 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).

The figure shows a number of relationships at the object and model layers. Messages contain content language expressions ("knowledge objects"), and these describe or make queries about both domain objects and agents. At the model level the connection between message and content expressions is

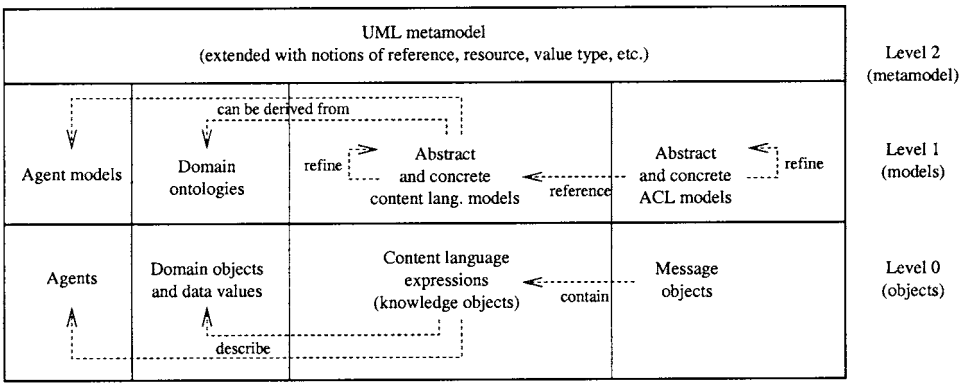

Figure 1 A metamodelling view of agent systems and models

mirrored by the need for the definition of an ACL to make reference to content language concepts; for example, the content tuple within a FIPA ACL expression must contain content language expressions representing domain objects, propositions, actions or definite descriptions.

The object-level relationship between content language expressions and the objects they denote is also mirrored at the model level. For a traditional logic-based content language, the term-building syntax and grammar of the language define the way in which symbols from an ontology can be combined to represent terms and propositions referring to the domain of interest. This paper shows how this idea can be extended to allow specialised domain-specific content languages to be generated from object-oriented ontologies, thus allowing for the use of functional terms built from "constructors" as content expressions. For example, a class defined in an ontology can have a counterpart as a proposition-building class in an ontology-specific content language – an instance of such a type can be interpreted as asserting some properties of one or more domain objects and some relationships that hold between them (further details are given in Section 6). This type of domain-specific expression can be convenient for application developers to work with when processing messages, particularly for those using object-oriented languages such as Java and object-oriented world models rather than traditional knowledge bases. To support the use of such representations, in conjunction with the generation of an ontology-specific content language, a content serialisation format and an application programmer interface for content marshalling and unmarshalling can also be generated given an ontology as input. Section 7 describes an implemented framework for this based on Java and the Resource Description Framework.

A similar connection can be seen to exist between models of agent types (particularly descriptions of their external interfaces) and the definition of certain types of content language expressions related to agents, e.g. expressions denoting actions that can form the content of request messages. This paper does not explore this issue, or present any proposals for modelling either agent types or action-denoting expressions. In future work it is intended to investigate possible mappings between agent models and strongly typed domain-specific representations for action descriptions. Of relevance to this work will be the various agent models developed by the agent-oriented software engineering community. Tveit (2001) and Wooldridge and Ciancarini (2001) have presented surveys of recent work in this area, and the architecture diagrams of Bergenti and Poggi (2000) are of particular relevance from a UML modelling perspective.

Figure 1 shows one other type of relationship within the model layer. Both ACLs and content languages can be defined by an abstract syntax (such as a UML model) as well as a concrete syntax (such as a grammar for a linear string-based realisation of the language). A concrete syntax can be seen as a refinement of the abstract syntax, and ideally the relationship between them would be defined by a formal mapping. This is a subject for future research.

Note that analysing agent systems using the metamodelling viewpoint as discussed above 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.

4 Modelling ontologies and languages in UML

In general, a three-level metamodelling hierarchy does not dictate the use of any particular language for expressing models at Level 1. By including different metamodels (i.e. modelling language definitions) at Level 2, different languages can be used to express Level 1 models. This is the approach taken by the Object Management Group's Model-Driven Architecture (MDA),[5] where a "met-ametamodel" is defined (this belongs in a fourth layer, not shown in Figure 1) and this can be used to define various modelling languages in the metamodel layer (Level 2).

In this paper, however, we focus on the use of UML as a unifying representation language for all agent-related models at Level 1. It has been argued elsewhere that UML, as an industry-standard object-oriented modelling language, is a good candidate both for the representation of ontologies (Bergenti & Poggi, 2000; Cranefield *et al.*, 2001; Cranefield & Purvis, 1999, 2000) and for the definition of the abstract syntax of agent communication languages (Cranefield *et al.*, 2000, 2002b). Therefore, for our current work, Level 2 of Figure 1 is occupied by the UML metamodel, with some extensions to allow the representation of concepts with or without identity, reference types and references. This is done using UML's standard extension mechanism of 'stereotypes'.

4.1 Modelling ontologies

Figure 2 presents a UML class diagram defining an ontology describing family relationships. To keep the example simple, only the relationships between parents and children are included. In order to understand this class diagram, it is sufficient to know the following:

- Rectangles depict classes.
  - The class name appears at the top (in italics if the class is abstract).
  - Any attributes appear below in a separate compartment.
  - A keyword enclosed in guillemets (French quotation marks) may appear at the top of the class box. This indicates that the box represents something other than a standard UML class, either a

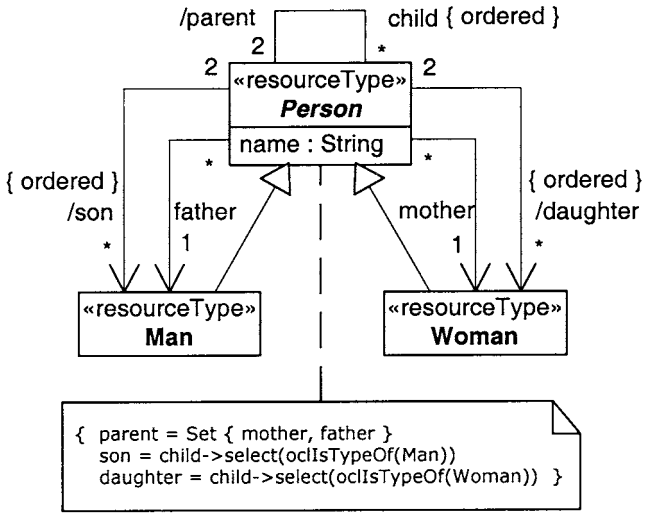

Figure 2 A class diagram representing a family ontology

[5] `http://www.omg.org/mda/`.

different type of UML "classifier" (such as an interface) or a "stereotype' – a subtype of some existing UML classifier type that is defined in an extension to the UML metamodel called a "profile'. In this case `resourceType` is a stereotype that will be explained in Section 5.1.

- Lines between classes represent association relationships.
  - Association ends may be labelled with "rolenames".
  - Association ends may be annotated with numbers indicating how many objects may have this association with a common instance of the class at the other end. "*" means "zero or more".
  - A stick arrowhead indicates that the association can be traversed in one direction only: the class at the tail of the arrow knows about the association but the class at the head does not.
  - An "ordered" constraint means that there is an order defined on the set of objects at that association end that are related to a common object at the other end. In implementation terms this means that the association end is represented by a list data structure within the class at the other end, rather than a set.
- A line with a triangular arrowhead represents generalisation, with the arrow pointing to the more general class.
- The name of any model element may be preceded by a forward slash character, indicating that the element can be computed from other elements in the model. In Figure 2 the association ends `parent`, `son` and `daughter` are annotated in this way.
- The dog-eared rectangle in Figure 2 contains constraints on the class `Person` expressed using the Object Constraint Language (Warmer & Kleppe, 1998). OCL can be used in conjunction with UML in order to constrain the possible models of a specification in ways that cannot be achieved using the UML structural elements alone. In this case the constraints specify how `parent`, `son` and `daughter` are related to `mother`, `father` and `child`.

UML contains many other modelling constructs besides those mentioned here. An overview of the full language is presented by Booch *et al*. (1998).

4.2 Modelling generic content language concepts

Figure 3 presents a UML diagram defining a set of interfaces representing types of content language expressions that the FIPA ACL and communicative act library refer to (we will refer to this as "the CL package"). The marker interface pattern (Grand, 1998) is used: the interfaces have no operations but represent distinct (externally defined) semantic notions. Interfaces are used because at this abstract

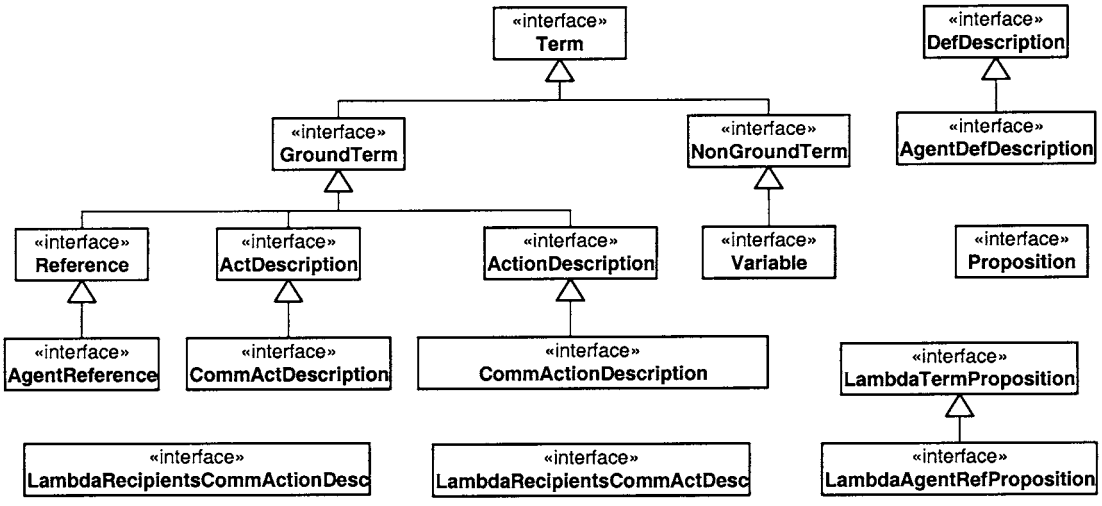

Figure 3 The CL package: generic content language concepts

level it is not appropriate to make any decisions about the implementation structure of these concepts.[6]

The key content language concepts identified by the FIPA abstract architecture are `Proposition`, `ActionDescription` and `Term`. `DefDescription` (definite description) is also required for use with the FIPA `query-ref` communicative act. The interface `Variable` is included because the notion of a variable is fundamental to the semantics of definite descriptions (Russell, 1956).

The diagram contains some additional interfaces representing concepts that are not explicitly mentioned by the ACL or communicative act definitions – we believe there are advantages to modelling these concepts explicitly:

- The notion of a reference is reified to allow a better understanding of the role that distributed reference schemes such as CORBA IORs and Web URIs can play within messages. This is further specialised to the notion of `AgentReference` – a reference to an agent.
- Terms describing communicative acts (CAs) and actions[7] are represented by the interfaces `CommActDescription` and `CommActionDescription`.
- A number of `Lambda` ... interfaces are introduced to represent the needs of communicative acts having arguments representing propositions with missing subjects (useful in "call for proposal"-style CAs) and communicative acts or actions with missing recipients (useful in proxy and propagate CAs).

Note that no commitment is made concerning the structure of propositions. All that the ACL and communicative act definitions require is that content that plays a particular semantic role (e.g. a proposition) can be identified as playing that role. In particular, note that the concept of a predicate is not included in the CL package. 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, for example as networks of objects asserting the values of the objects' attribute values and some relationships that hold between them.

4.3 Modelling specific content languages

Given the content language concepts model, a particular content language can be defined as a class diagram with UML realisation relationships used to indicate how particular content language classes correspond to generic content language concepts.

Figure 4 shows a fragment of the abstract syntax of a FIPA SL-style language (for simplicity we only show the syntax for conjunctions of well-formed formulae and omit other logical connectives, the modal operators for belief, uncertainty, persistent goals and intention, and the operators expressing the feasibility and performance of actions). The UML "lollipop" symbol is used to indicate realisation relationships, i.e. the declaration that a class implements the interface at the round end of the lollipop. In this case, the interfaces implemented are the general content language concepts modelled in the CL package from Figure 3, and the notion of "implementation" is that of the marker interface pattern: the class plays the semantic role represented by the interface. The solid diamonds on association ends indicate composition (whole-part) relationships.

4.4 Modelling an ACL

Figure 5 shows a partial model of a FIPA-style agent communication language. The lower part of the diagram shows how particular CA types are represented by specialisations of the notion of a message. Note that the specialised CA message classes declare their required content types by referring to the

[6] Alternatively, UML's "type" stereotype could have been used, but the notation for types and implementation classes is more cumbersome than that for interfaces and the classes that realise them.

[7] The terminology used here is that an act is something that can be peformed by an agent and an action is the performance of an act by an agent.

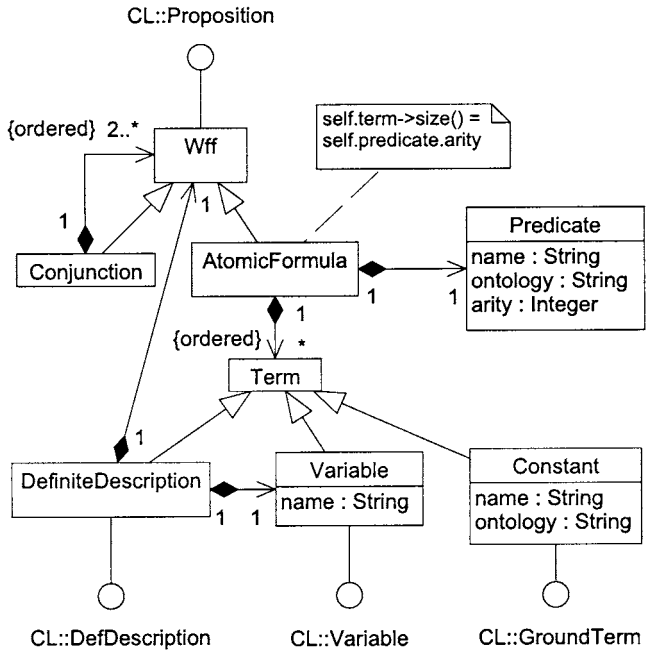

Figure 4 A partial model of a FIPA SL-style content language

interfaces defined in the content language concepts model presented in Figure 3. This provides a strongly typed account of the various numbers and types of content expression required in the body of different CAs. Expressions in any content language can be included within an ACL expression provided that they implement the appropriate interface from the CL package. For example, the Object Query Language could be used as a form of definite description simply by defining a class

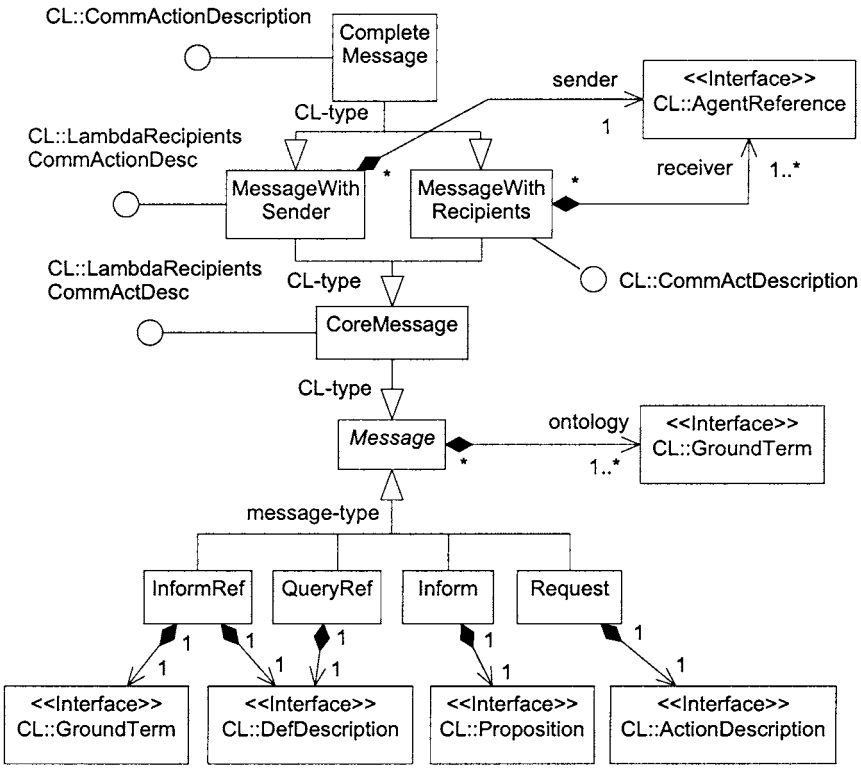

Figure 5 A partial model of a FIPA-style ACL

`OQLDescription` with a string-valued attribute `query` and declaring it to implement `CL::DefDescription`.

The diagram does not show all FIPA CAs, and an explicit `InformRef` message type is added to provide a more convenient way of responding to a `QueryRef` message (currently in FIPA it is necessary to send an `inform` message with a content proposition that equates a term with the definition description that was sent in the original query). The model also currently omits the various additional parameters such as `conversation-id` that a FIPA ACL message may optionally have.

The modelling of an ACL is complicated by the FIPA practice of reusing the ACL message syntax within SL to represent completely or partially specified communicative acts that can appear within the content of messages (e.g. as one of the arguments of a proxy CA). To account for this in a strongly typed modelling language requires the identification of the roles that can variously be played by complete and incomplete forms of message as content language expressions. The top half of the figure shows how different specialisations of the abstract message class are declared to "implement" different types of content language concept from the content language concepts model.

To separate the two distinct types of message specialisation discussed above (different types of communicative act represented and different types of content language expression represented), the UML notion of separate dimensions of specialisation is used. Discriminators (`message-type` and `CL-type` in the figure) can be used to indicate a particular dimension of specialisation. An instance of a concept that is specialised in multiple dimensions must either belong to a class that multiply inherits from specialised classes from all dimensions of specialisation or it must be "multiply classified" (i.e. belong to more than one class) to cover all specialisation dimensions. It would be impractical to use multiple inheritance to define classes covering all possible combinations of CAs and omitted or included sender and recipient links, so instances of messages in this model are required to be multiply classified (see Figure 13). The use of multiple classification complicates the mapping to conventional OO programming languages such as Java, but this feature can be simulated using delegation.

5 A UML profile for ontology modelling

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 describe a UML "profile" containing stereotypes that allow these aspects of an ontology to be represented in a class diagram. Note that this profile is designed to facilitate the use of UML in its own right as an ontology modelling language and therefore has different aims from the profile of Baclawski *et al*. (2001), which allows UML class diagrams to be used to represent models in the DAML language.[8]

5.1 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 and the way in which links between objects are implemented. This is too agnostic a stance for use in the description of distributed multi-agent systems. Agents, in general, may communicate about objects that are external to them and which have existing identifiers such as names defined in some controlled namespace, CORBA interoperable object references (IORs) or World Wide Web Uniform Resource Identifiers (URIs). It is therefore natural and parsimonious for agents to use these existing external identifiers when communicating with other agents. Furthermore, it would also be useful to provide a

[8] `http://www.daml.org/`.

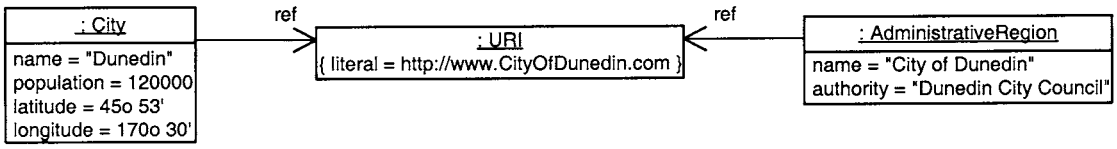

Figure 6 An object diagram with a shared external reference

way for ontology designers to declare, for each concept having a notion of identity, which reference scheme is used to publicly identify instances of that concept (if there are any constraints on the allowed scheme). This would provide a strongly typed account of the types of value that can be considered as legal references within the content of ACL messages. We have therefore included in our UML profile a stereotype of the metaclass `class` that allows classes to be annotated with their appropriate reference schemes in order to allow external references to be associated with objects in a UML object diagram.

Before presenting this extension, we first illustrate our notation for associating external references with particular objects in a UML object diagram. Figure 6 shows an object diagram with two objects, one of class `City` and one of class `AdministrativeRegion`, and an instance of a data type representing uniform resource identifiers. As UML has not defined a notation for data values, the value of the URI instance is shown using a UML tagged value with the tag name "literal". Both objects have a one-way link to the URI with the URI playing the role "ref".

Figure 6 illustrates an important benefit of adding the notion of external identifier to UML: two different objects (from an agent's internal viewpoint) can be associated with the same external reference. This allows an agent to have two different representations of an external object, and therefore conforms to an important principle of the Semantic Web: that "anyone can say anything about anything" (Berners-Lee, 1997). In the Semantic Web, a URI can be used to identify anything. Anyone can then make statements about that "resource" using the Resource Description Framework (Lassila & Swick, 1999). All they need to know is the URI – they can use properties from any ontology they like to describe the resource. This means that an agent's knowledge base may need to include information about any given object gathered from different sources, and represented using different ontologies. For example, Figure 6 shows two different views of the city of Dunedin, New Zealand: one describing it as a city and another describing it as an administrative region. The notation presented here provides UML with this ability to use multiple classes to describe a given entity that is identified by an external reference. However, rather than assuming the use of URIs as a uniform referencing system, we allow other types of reference scheme to be declared and associated with classes in ontologies. We now address the question of how such declarations can be made.

Figure 7 presents a portion of an ontology that defines the class used on the left-hand side of the object diagram in Figure 6. However, for the present we do not show the stereotype keywords necessary to indicate that the concept of a URI is represented by a data type, not a class.[9] The diagram also refers to an abstract data type `AnyReference`. This is defined in our UML profile to represent

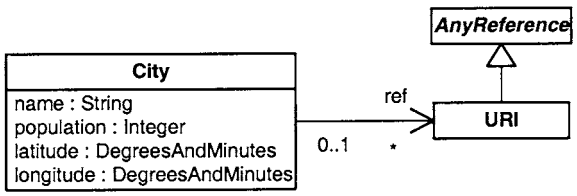

Figure 7 Defining a reference type in a class diagram

[9] In UML, the distinction between a class and a data type is that instances of a data type are "pure values" with no identity.

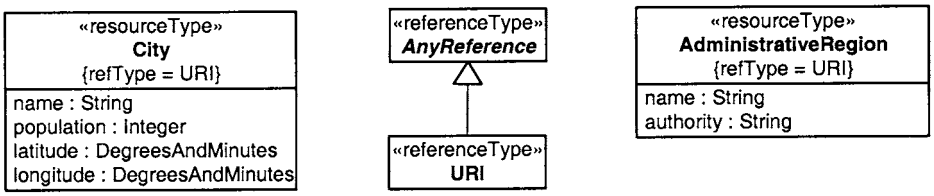

Figure 8 An example class diagram using the ontology-modelling profile

the top-level class in the generalisation hierarchy of all reference types. It is also declared to implement the `GroundTerm` interface from the `CL` package which means that any reference can play the role of a ground term in our framework. For simplicity, in Figure 7 we also assume that a data type `DegreesAndMinutes` has been defined to represent angles.

The key aspect of this diagram is the association defined between the class `City` and the data type `URI`. This shows that any `City` object has associations with zero or more URIs playing the role "ref" and that each URI can only be associated with one object of that class (but may possibly not be associated with any objects).

When defining ontologies for domains that will be the subject of agent communication, there will be a need for many concepts to be declared to have external references of a particular type. It would be tedious to have to show explicitly associations to reference types (as in Figure 7). We have therefore defined an extension of UML to provide a more concise way of providing this information in class diagrams. 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 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 and to allow references to be associated with particular objects we have defined three stereotypes. A class diagram using these stereotypes is shown in Figure 8 and the full profile is shown in Figure 9 using UML 1.4 notation.

The stereotype `Reference` is a specialisation of UML's metaclass `DataValue` that allows references to be represented in object diagrams. It defines the tag `literal` that was used in Figure 6 to display the URI's value. `ReferenceType` is a specialisation of the metaclass `DataType` and is used to define a particular type of reference. Any datatype declared with this stereotype must be a subclass of the class `AnyReference` which is introduced in a new UML "model library" associated with the profile. The stereotype causes the class's definition to be elaborated by adding a declaration that it implements the `Reference` interface from the `CL` package discussed in Section 4.2.

The stereotype `ResourceType` extends the metaclass `Class`. Any class defined with this stereotype will have its definition elaborated to include an association with either class `AnyReference` or a particular subclass of it that the ontology designer can optionally indicate using a `refType` tagged value. A fourth stereotype representing objects without identity is discussed in Section 5.2.

The profile definition in Figure 9 shows three UML packages: the profile itself and, above it, two "model libraries" – model-level packages that are referenced by the profile. In the `OntologyProfile` package, three standard UML metaclasses and a standard stereotype appear along the top. Below these are the four stereotypes being defined in the profile together with constraints defining their meaning. Two of the constraints have a non-standard stereotype: `elaboration`. These define an automatic extension of a model to include the additional relationships discussed above (this saves the ontology designer from adding these manually). These elaborations are currently defined using English, as UML does not specify any way of defining this type of automatic rewriting of a model: constraints are considered to be assertions for the purposes of validation. However, at least one UML modelling tool (Objecteering/UML) offers the facility to define transformation rules for "automatic completion of diagrams" (Softeam, 1999).

5.2 Modelling data values

The profile also includes a way to define classes of object without identity, which we call "value types" (after the CORBA terminology). UML has a metaclass `DataType` that represents sets of values without identity, 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. Therefore, to allow value types to be declared in an ontology, a stereotype `ValueType` is defined in the profile. This stereotype specialises the UML metaclass `Class` and is a subclass of the standard UML stereotype `type`.[10] It is constrained to only allow associations navigable away from value types if the opposite assocation ends are connected to data types or other value types.

6 Ontology-specific content languages

The metamodelling hierarchy in 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

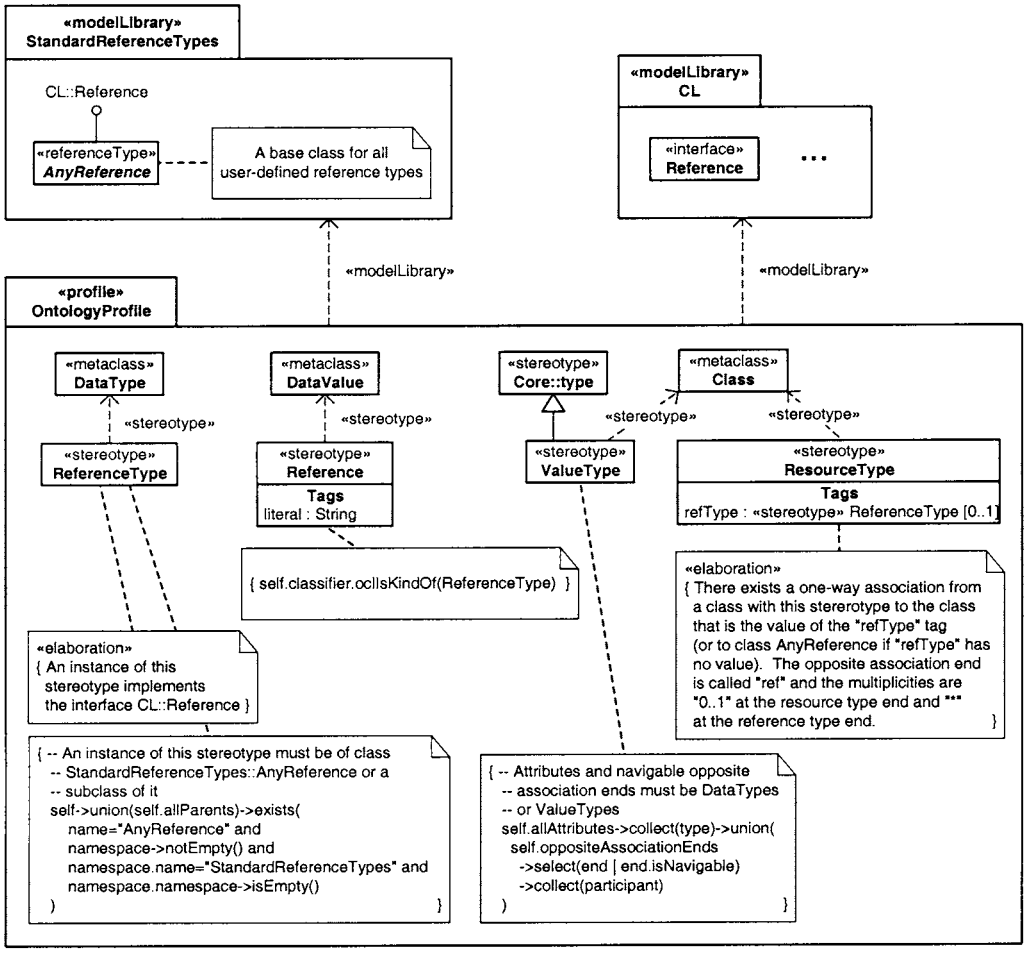

Figure 9 A UML profile for ontology-modelling

[10] This is based on the approach used in the UML Profile for CORBA (OMG, 2000).

within messages. 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 10 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 `OntologySpecificCL` package and the pre-existing `SL` one indicates that all concepts defined for SL (such as conjunction and negation) are included in the generated package. Similarly, the generated package allows ACL expressions to be used as content expressions by inheriting the classes from the ACL package.

The mapping generates a package containing a set of classes that define a specialised content language corresponding to an ontology. Each class in an ontology maps to one or more classes that are declared to play particular roles (ground term, proposition and so on) in the content language by implementing interfaces from the `CL` package.

A valuetype declared in an ontology maps to an equivalent class declaration (without the `valueType` stereotype). This class is declared to implement the interface `CL::GroundTerm` because objects of the class can meaningfully be embedded in messages as terms.

A class with the `resourceType` stereotype maps to a corresponding 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 some or all of their attribute values and links). The generated proposition class is identical to the class in the ontology, except that abstract classes become concrete (as agents may not know the precise class of an object), attributes and associations become

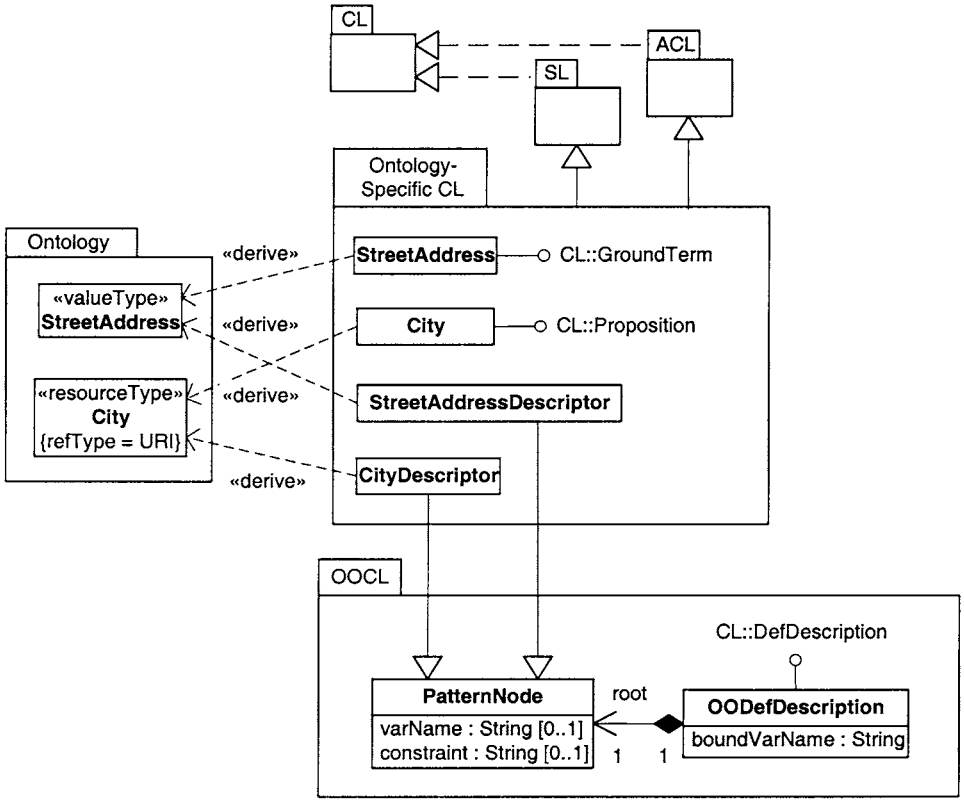

Figure 10 An ontology-specific content language

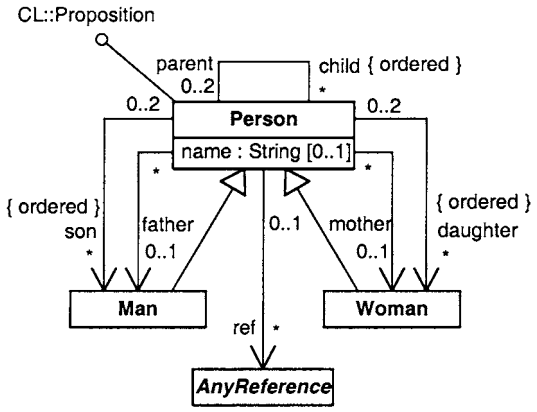

Figure 11 Proposition classes generated from the family ontology

optional (agents do not need to include complete information about an object within a message), stereotypes from the ontology-modelling profile and any constraints are removed, and any associations with reference classes that were implicitly indicated by the use of a `resourceType` stereotype and an optional `refType` tag are explicitly included. Figure 11 shows a set of predicate classes generated from the family ontology in Figure 2.

In addition, for both resource types and value types, corresponding classes are generated to allow ontology-specific object-oriented structures to play the role of definite descriptions within messages. These classes are designed so that definite descriptions can be constructed as networks of interlinked objects with the links representing relationships that are asserted to hold between the objects. The generated descriptor classes have the same structure as the proposition classes except attributes are treated differently (explained below). Each descriptor class is declared to be a specialisation of a class `PatternNode` from a package `OOCL` (containing classes useful to any content language that uses object networks to represent definite descriptors). The `PatternNode` class defines optional attributes to represent a variable to associate with the node and a constraint expression in OCL. The full `OOCL` package is shown at the bottom of Figure 10 – it includes the `OODefDescriptor` class that specifies the variable of interest in the query and has a link to the root node of the network of pattern nodes. In the generated descriptor classes attributes and reference associations are modelled by an association with a pattern node having an optional attribute giving a value for the associated object. This allows a pattern to associate only a variable or a constraint with an attribute of an object that is the subject of a query. The package `ST` (standard types) is defined to contain pattern classes for all primitive types (currently we assume the use of the Object Constraint Language primitive types). Figure 12 shows the structure of the generated descriptor classes corresponding to the family ontology.

A class in the ontology having no stereotype has corresponding proposition and definite description classes generated, but not a value class – in the absence of a `valueType` declaration it is not known if this would be meaningful.

Figure 13 shows a UML object diagram depicting a message built using the ACL classes from Figure 5 and the family ontology specific definite description classes from Figure 12. Note that the top-level message object (centre top) is "multiply classified" to be both a `QueryRef` object and a `CompleteMessage`. This is due to the design decision discussed in Section 4.4 to use two independent dimensions of specialisation in the UML model for the ACL. The object on the left of the figure is intended to represent a reference to an ontology by name – its class `BasicCL::Identifier` is assumed to be predefined to be an implementation of the `CL::Reference` interface.

7 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

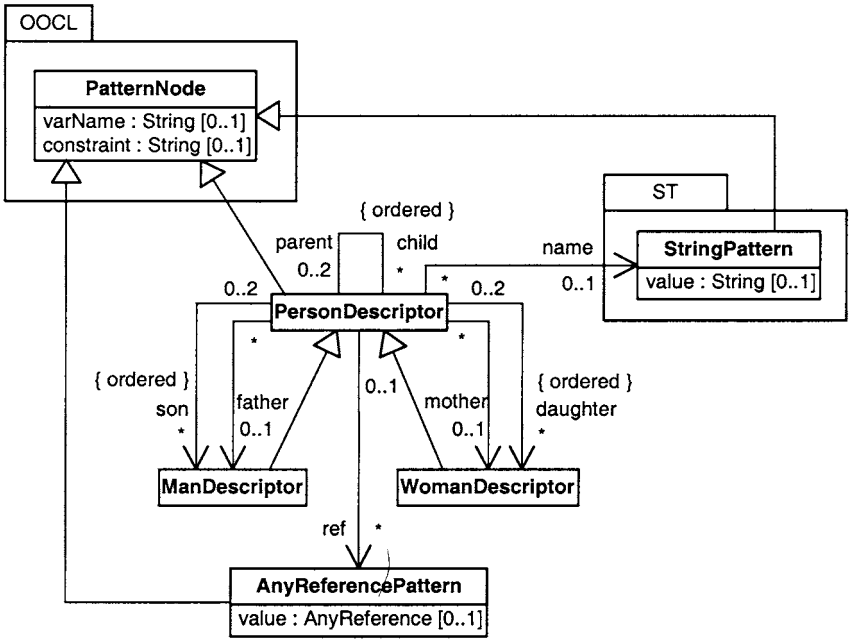

Figure 12 Generated descriptor classes for the family ontology

descriptions and (for value types) terms. It remains to formally define and implement these mappings. In previous work, a "UML data binding scheme" has been developed (Cranefield, 2001a, 2001b). This allows agent messages to be visualised as object diagrams and serialised using the XML-based Resource Description Framework (Lassila & Swick, 1999). The serialisation is performed with reference to RDF schemas that are generated from the UML definitions of the ACL content languages used. 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 (W3C, 1999) starting from encodings of the UML models in the XML Model Interchange format (XMI).

At present, classes in ontologies are mapped directly to Java classes and RDF resources rather than being just mapped to corresponding classes in on ontology-specific content language. This generation process will be extended to implement the mappings described in the previous section in order to

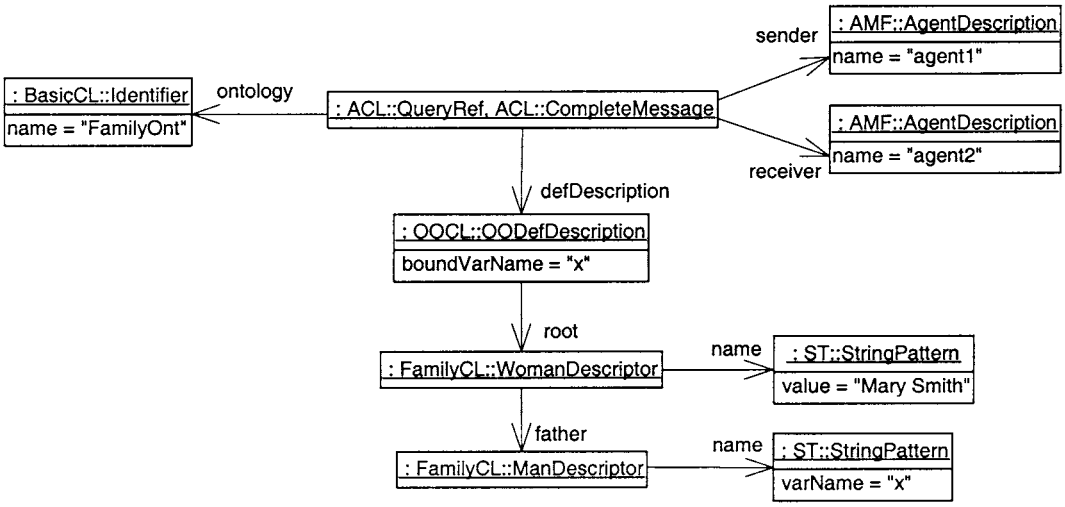

Figure 13 An example message

provide agent programmers with an automatically generated Java application programmer interface for including domain-specific objects within messages – but only in a type-safe manner as representions of propositions, definite descriptions or value type terms. This extension will require simulating multiple classification in Java (although RDF supports it directly), but is otherwise straightforward.

There is one feature of the UML data binding scheme that has not yet been addressed in the present work. The generated proposition classes allow attribute values and links to be omitted if they are not of relevance to the discussion – otherwise an agent would 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 cases where there really is no value and cases 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 (Cranefield, 2001b). 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.

8 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: the objects must be intended to represent propositions, values or definite descriptions. The ability to create such ontology-specific content expressions can be provided in a principled way by generating an object-oriented ontology-specific content language from an ontology modelled using UML.

It has also been argued that ontologies should distinguish between concepts with a notion of identity and those without, and a UML profile has been developed to support the explicit inclusion of this information in ontologies.

The mapping from ontologies to ontology-specific content languages provides a principled basis for the generation of an application programmer interface (API) allowing agent developers to include domain-specific objects within messages. In future work we will extend a FIPA agent platform we are developing to allow such ontology-specific APIs to be plugged in with minimal programmer effort.

References

Baclawski, K, Kokar, M, Kogut, P, Hart, L, Smith, J, Holmes, W, Letkowski, J and Aronson, M, 2001, "Extending UML to support ontology engineering for the Semantic Web" UML 2001 – *The United Modeling Language, Lecture Notes in Computer Science 2185*, Springer. Editors: M. Goyolla and C. Kobryn 342–360.

Bergenti, F and Poggi, A, 2000 "Exploiting UML in the design of multi-agent systems" in A Omicini, R Tolksdorf and F Zambonelli (eds) *Engineering Societies in the Agents World*, Lecture Notes in Computer Science 1972, Springer pages 106–113.

Berners-Lee, T, 1997, "Metadata architecture", World Wide Web Consortium Discussion Document, http://www.w3.org/2000/01/sw/.

Booch, G, Jacobson, I and Rumbaugh, J, 1998, *The Unified Modeling Language User Guide* Addison-Wesley.

Cattell, R, Barry, D, Berler, M, Eastman, J, Jordan, D, Gamerman, S, Russell, C, Schadow, O, Stanienda, T and Velez, F (eds), 2000, *The Object Data Standard: ODMG 3.0* Morgan Kaufmann.

Cranefield, S, 2001, "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/.

Cranefield, S, 2001b, "UML and the Semantic Web" *Proceedings of the International Semantic Web Working Symposium* (SWWS) http://www.semanticweb.org/SWWS/program/full/paper1.pdf.

Cranefield, S, Haustein, S and Purvis, M, 2001a, "UML-based ontology modelling for software agents" *Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents* http://CEUR-WS.org/Vol-52/oas01-cranefield-1.pdf.

Cranefield, S, Nowostawski, M and Purvis, M, 2002 "Implementing agent communication languages directly from UML specifications" Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems, ACM Press. Forthcoming.

Cranefield, S and Purvis, M, 1999, "UML as an ontology modelling language" *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence* (IJCAI-99) http://CEUR-WS.org/Vol-23/cranefield-ijcai99-iii.pdf.

Cranefield, S and Purvis, M, 2000, "Extending agent messaging to enable OO information exchange" *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), Vienna, 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.

Cranefield, S, Purvis, M and Nowostawski, M, 2000, "Is it an ontology or an abstract syntax? Modelling objects, knowledge and agent messages" *Proceedings of the Workshop on Applications of Ontologies and Problem-Solving Methods, 14th European Conference on Artificial Intelligence* (ECAI 2000) http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/16.pdf.

Finin, T, Labrou, Y and Mayfield, J, 1997 "KQML as an agent communication language" in JM Bradshaw (ed.) *Software Agents* MIT Press.

FIPA, 2000a, "FIPA ACL message representation in string specification" Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00070/.

FIPA, 2000b, "FIPA communicative act library specification" Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00037/.

FIPA, 2000c, "FIPA SL content language specification" Foundation for Intelligent Physical Agents, http://www.fipa.org/specs/fipa00008/.

Genesereth, MR and Ketchpel, SP, 1994 "Software agents" *Communications of the ACM*, **37**(7) 48–53.

Grand, M, 1998, *Patterns in Java*, *Volume 1* Wiley.

Lassila, O and Swick, RR, 1999, "Resource Description Framework (RDF) model and syntax specification" World Wide Web Consortium, http://www.w3.org/TR/REC-rdf-syntax.

NCITS, 1998, "Draft proposed American national standard for Knowledge Interchange Format" National Committee for Information Technology Standards, http://logic.stanford.edu/kif/dpans.html.

OMG, 2000, "UML profile for CORBA" Object Management Group, http://doc.omg.org/ptc/00-10-01.

OMG, 2002, "CORBA/IIOP 2.6.1 specification" Object Management Group", http://www.omg.org/technology/documents/formal/corba_iiop.htm.

Russell, B, 1956, "On denoting" in *idem*, *Logic and Knowledge: Essays, 1901–1950* (ed. RC Marsh) Allen and Unwin. Also at http://www.santafe.edu/~shalizi/Russell/denoting/.

Sadek, M, 1990, "Logical task modelling for man-machine dialogue" *Proceedings of the Eighth National Conference on Artificial Intelligence* (AAAI Press-90) 970–975.

Softeam, 1999, "UML profiles and the J language: totally control your application development using UML" http://www.softeam.org/pdf/us/uml_profiles.pdf.

Tveit, A, 2001, "A survey of agent-oriented software engineering" *Proceedings of the First Norwegian University of Science and Technology (NTSU) Computer Science Graduate Student Conference* http://csgsc.idi.ntnu.no/2001/pages/papers/atveit.pdf. 1–28.

W3C, 1999, XSL Transformations (XSLT) version 1.0. World Wide Web Consortium, http://www.w3.org/TR/xslt.

Warmer, JB. and Kleppe, AG, 1998 *The Object Constraint Language: Precise Modeling With UML* Addison-Wesley.

Willmott, SN, Dale, J, Burg, B, Charlton, C and O'Brien, P, 2001, "Agentcities: a worldwide open agent network" *Agentlink News* **8** 13–15; http://www.AgentLink.org/newsletter/8/AL-8.pdf.

Wooldridge, M and Ciancarini, P, 2001, "Agent-oriented software engineering: the state of the art" in P Ciancarini and MJ Wooldridge (eds) *Agent-Oriented Software Engineering*, Lecture Notes in Computer Science 1957, Springer 1–28.