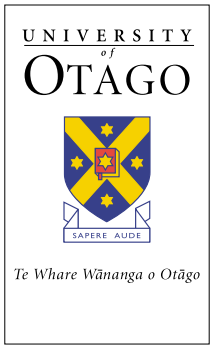

UML-Based Ontology Modelling for Software Agents

Stephen Cranefield
Stefan Haustein
Martin Purvis

The Information Science Discussion Paper Series

Number 2001/07
May 2001
ISSN 1172-6024

University of Otago

Department of Information Science

The Department of Information Science is one of six departments that make up the School of Business at the University of Otago. The department offers courses of study leading to a major in Information Science within the BCom, BA and BSc degrees. In addition to undergraduate teaching, the department is also strongly involved in postgraduate research programmes leading to MCom, MA, MSc and PhD degrees. Research projects in spatial information processing, connectionist-based information systems, software engineering and software development, information engineering and database, software metrics, distributed information systems, multimedia information systems and information systems security are particularly well supported.

The views expressed in this paper are not necessarily those of the department as a whole. The accuracy of the information presented in this paper is the sole responsibility of the authors.

Copyright

Copyright remains with the authors. Permission to copy for research or teaching purposes is granted on the condition that the authors and the Series are given due acknowledgment. Reproduction in any form for purposes other than research or teaching is forbidden unless prior written permission has been obtained from the authors.

Correspondence

This paper represents work to date and may not necessarily form the basis for the authors' final conclusions relating to this topic. It is likely, however, that the paper will appear in some form in a journal or in conference proceedings in the near future. The authors would be pleased to receive correspondence in connection with any of the issues raised in this paper, or for subsequent publication details. Please write directly to the authors at the address provided below. (Details of final journal/conference publication venues for these papers are also provided on the Department's publications web pages: http://www.otago.ac.nz/informationscience/pubs/publications.htm). Any other correspondence concerning the Series should be sent to the DPS Coordinator.

Department of Information Science
University of Otago
P O Box 56
Dunedin
NEW ZEALAND

Fax: +64 3 479 8311
email: dps@infoscience.otago.ac.nz
www: http://www.otago.ac.nz/informationscience/

UML-Based Ontology Modelling for Software Agents

Stephen Cranefield*, Stefan Haustein† and Martin Purvis*

* Department of Information Science
University of Otago
PO Box 56
Dunedin, New Zealand
{scranefield,mpurvis}@infoscience.otago.ac.nz

† University of Dortmund
Computer Science VIII
Baroper Str. 301
D-44221 Dortmund, Germany
stefan.haustein@udo.edu

ABSTRACT

Ontologies play an important role in defining the terminology that agents use in the exchange of knowledge-level messages. As object-oriented modelling, and the Unified Modeling Language (UML) in particular, have built up a huge following in the field of software engineering and are widely supported by robust commercial tools, the use of UML for ontology representation in agent systems would help to hasten the uptake of agent-based systems concepts into industry. This paper examines the potential for UML to be used for ontology modelling, compares it to traditional description logic formalisms and discusses some further possibilities for applying UML-based technologies to agent communication systems.

1. INTRODUCTION

Ontologies play an important role in defining the terminology that agents use in the exchange of knowledge-level messages and therefore the choice of an ontology representation language is a significant issue when designing a multi-agent system. Traditional approaches to ontology representation use modelling formalisms developed by the artificial intelligence knowledge representation community—in particular, frame-based languages descended from the KL-ONE system [7] and their formalisations as various forms of *description logic*. However, these languages were developed for use in monolithic knowledge representation systems that are very different in character from distributed multi-agent systems.

One of the potentially significant advantages of multi agent systems is that agents built using different technologies can be combined to form a society and can collectively solve problems that none of the agents could on their own. In particular, it cannot be assumed that all agents will be implemented on top of a KL-ONE style knowledge representation system. Therefore, the choice of a ontology modelling language should depend more on its suitability for the initial construction of the ontology than any support it may offer for specialised run-time reasoning. In fact, we believe that the reasoning requirements of future multi-agent applications are not likely to be predominant design criteria and other issues such as coping with design complexity are more important.

In addition, there are other things to consider when deciding what representation to use for ontology modelling. If agent-based architectures are to find their way from the research laboratory into wide-spread use in industry and business, we would do well to consider the degree to which a candidate ontology modelling language would gain aceptance in the wider community outside the research laboratories. As object-oriented modelling, and the Unified Modeling Language (UML) in particular, have built up a huge following in the field of software engineering and are widely supported by robust commercial tools, the use of UML for ontology representation in agent systems would help to hasten the uptake of agent-based systems concepts into industry. This paper therefore examines the potential for UML to be used for ontology modelling, compares it to traditional approaches (description logic in particular) and discusses some further possibilities for applying UML-based technologies to agent communication and reasoning systems.

The paper is structured as follows: First, we discuss traditional approaches to ontology modelling, focusing in particular on description logic. We then explain why we consider UML to be suitable as a knowledge representation language before making a comparison between the features of description logic and UML. We then discuss UML in the wider context of agent communication.

2. TRADITIONAL APPROACHES

The most widely used traditional approaches for ontology modelling are the Knowledge Interchange Format (KIF) [26] and description logic.

KIF is a language based on first-order predicate logic with extensions for representing definitions and metaknowledge. While first-order logic is a low-level language for expressing ontologies, the Ontolingua tool [19] allows users to build KIF ontologies at a higher level of description by importing predefined ontology definitions. In particular, the frame ontology [24] allows ontologies to be described at a level similar to description logics.

Description logics are a formalisation of the representations and processes underlying frame-based knowledge representation systems in the tradition of KL-ONE [7]. These systems support the definition of concepts by simply naming them and specifying where they fit in the generalisation/specialisation hierarchy of existing concepts. New concepts can also be defined in terms of existing concepts using the operations of *concept conjunction*: the `and` operator can be used to specify that the new concept is a common specialisation of a number of other concepts. New *roles* may be introduced to represent possible relationships that may hold between individuals in the domain being modelled, and concept definitions may include restrictions on the possible values, number of values, or type of values that a role may have for the concept being defined.

The following example, adapted from Nebel [27], illustrates these features of description logic. In the notation used, `Anything` is a predefined concept representing the class of all things and the class of all relations is denoted by `anyrelation`. The symbol $\doteq$ rep-

resents concept definition and concept specialisation is represented by $\dot{\leq}$.

```
Human ≐ Anything
Set ≐ Anything
Man ≐ Human
Woman ≐ Human
member ≐ anyrelation
Team ≐ (and Set
            (all member Human)
            (atleast 2 member))
```

Description logics are designed to support certain types of inferences over user-defined concepts and instances of concepts and roles that may be stored in a knowledge base (see Section 4.4). These types of deduction are designed to help the user in incrementally designing a coherent set of concepts and instances to describe a domain.

3. UML FOR ONTOLOGY MODELLING

The Unified Modeling Language is a language and associated graphical notation for object-oriented analysis and design. The object-oriented modelling paradigm has become the mainstream technique in the software industry based on the widely accepted view that object-oriented modelling fits well with people's intuitive models of the world [4].

UML is a standard from the Object Management Group (OMG) [29]. The OMG is a consortium of around 800 member companies and institutions involved in software engineering. Therefore, UML has a very large and rapidly expanding user community and the language is widely taught in universities. There are also many tools available for creating and editing models in UML using direct manipulation of the models' graphical presentation[1].

The use of UML is now widespread in industry and its rapid acceptance (even for the design of mission-critical applications) suggests that it provides an effective and scalable approach to conceptual modelling, and therefore warrants serious consideration as an ontology modelling language [12, 1]. However, UML has a rather different character from the logic-based formalisms traditionally used for ontology modelling. UML is defined in terms of a graphical syntax as well as a metamodel defining the types of structure that may appear in a user's model and the ways in which they can be related to each other. The metamodel plays the same role that a context-free grammar does in defining the legal combinations of terminal symbols in a string-based language, but without the limitations caused by the use of a linear textual format. There is currently no text-based syntax designed for human use (although work is in progress to design one [30]), but there is a standard XML-based format—the XML Model Interchange format (XMI)—that allows models to be encoded as instances of the UML metamodel. Despite these differences from traditional knowledge representation languages, we believe UML models have a number of features

[1] Although there are graphical notations and/or model editors existing for some AI knowledge representation languages (e.g. Semantic Nets and Conceptual Graphs), UML is unique in having both a compact, high-level and standardised notation and a high degree of tool support.

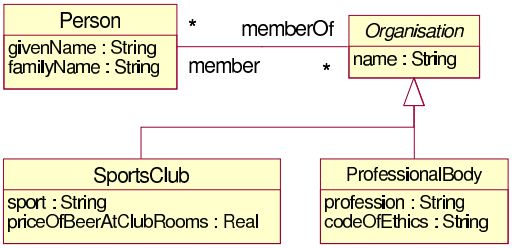

Figure 1: A simple UML class diagram example

commonly regarded as characteristic of the declarative knowledge representation paradigm [21, 13]:

- Knowledge expressed in UML is directly accessible for human comprehension (via its standard graphical presentation) and for machine processing (via the XMI model interchange format and associated software libraries or the application programmer interface defined by the OMG's Meta Object Facility).
- Knowledge in a UML model can be changed easily due to the modular nature of object-oriented modelling. Changes to one feature in the model do not generally affect other features.
- UML models can be used for purposes that were not anticipated at the time of model creation. In other words, UML is an abstract modelling language, not tied to any particular application.
- New knowledge can be derived from UML models by reasoning about their contents. In particular, UML has an associated constraint language—the Object Constraint Language (OCL)—that can be used to define derived model elements (those that can be computed from other elements) and to assert arbitrary constraints on the possible instances of a model. This aspect of UML has not been well supported by tools in the past, but a number of OCL-aware tools [3, 35] and code libraries [34] are beginning to appear.

With this viewpoint, UML can be regarded as a suitable candidate for knowledge representation. In particular, we refer to UML *class diagrams* which provide a rich notation for defining classes, their attributes and the relationships between them. They can therefore be used to define ontologies in an object-oriented fashion.

Figure 1 shows a simple class diagram specifying the structure of and relations between classes representing people and organisations. The diagram also shows two specialisations of the organisation class: sports club and professional body. In order to understand this UML 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.
- Lines between classes represent association relationships.

  - Association ends may be labelled with "role names".
  - Association ends may be annotated with numbers indicating how many objects may have this association with instances of the class at the other end. '*' means "zero or more".

- A line with a closed arrowhead represents generalisation, with the arrow pointing to the more general class.

UML contains many other modelling constructs besides those mentioned here [5, 28], such as n-ary associations, association classes (associations with their own attributes and the ability to take part in associations with other objects), associations navigable in one direction only, and "ordered" constraints on association ends (specifying a sequence-based rather than set-based semantics). UML also includes the Object Constraint Language (OCL) that can be used to constrain the possible models of a specification in ways that cannot be achieved using the UML structural elements alone.

Examples of more complex ontologies expressed in UML can be found elsewhere [12, 10, 11].

4. DESCRIPTION LOGIC AND UML: A COMPARISON

Description logic and UML have resulted from different fields of research. Description logic was produced by the knowledge representation community as a formalisation of the conceptual modelling support provided by systems such as KL-ONE. In particular, description logic is optimised to allow automated consistency checking and the classification of new concepts with respect to existing concepts as the model is constructed. UML stems from research into object-oriented modelling and is designed to allow the direct expression of analysis and design models of problem domains and software systems using OO concepts. In this section we will look at these differences from an agent system viewpoint.

4.1 Modelling paradigm

Defining an ontology in DL involves introducing terms naming or describing concepts and combining them to create new terms describing more complex concepts. Consider the DL definition of Team shown earlier:

```
Team ≐ (and Set
            (all member Human)
            (atleast 2 member))
```

This definition states that the concept described by the symbol Team is the conjunction of three other concepts: `Set`, the unnamed concept of anything that has a member role having a value that is any instance of the concept Human, and the unnamed concept of anything that has a member role having at least two values. Although these latter two concepts are highly unlikely to be of interest to agents, it is unavoidable in DL that mentioning these role restrictions for the `Team` concept introduces them as concepts with the same status as any others.

In UML, a class diagram is a direct specification of the structure of a number of classes and the relationships between them. A class is defined by firstly an optional parent class (or a set of parent classes), from which it inherits all attributes and associations, and secondly a set of additional attributes and associations. Each class can be seen as the template for all instances of that class. Only the classes that are of interest to the modeller need to be defined.

4.2 Roles vs. Attributes and Associations

One significant difference between DL and object-oriented modelling notations, such as UML, is the style in which the structure of concepts and relationships between concepts are defined. In DL, there is a single construct to model these: the role. Roles are first-class entities: a binary role represents a binary relation on the universe of discourse U, i.e. a subset of U^2. A given role can potentially be used to make an assertion about any individual in U.

A concept can be defined in terms of restrictions on the possible value types, number of values, or possible values for a role when applied to instances of that concept. Apart from any type restrictions declared for particular concepts, roles are untyped. Also, if an ontology makes no mention of a role in relation to a given concept, this does not prevent a knowledge base from containing assertions that an instance of a concept has a value for that role. Consider the `Set` concept defined earlier. This has no mention of any roles, although it is clear from the definition of `Team` (which is a specialisation of `Set` with restrictions on `member`) that the intended way of stating that instances are members of a set is to use the `member` role. However, if the knowledge base contained the assertions `(Set s)` and `(element s a)` (where the second assertion uses the wrong vocabulary to refer to members of a set), this would not be considered to be an error (assuming that the role `element` was introduced for some other concept in the ontology, e.g. `PeriodicTable`).

In fact, some forms of description logic would allow the ontology designer to state that `member` is defined for `Set` by a statement of the following form, which means that `Set` is a special case of things that have values for the `member` role:

```
Set ≤̇ (exists member Anything)
```

Ignoring the fact that this does not allow for empty sets, this now allows the absence of any `member` assertions about set `s` to be detected (which may result in the mistaken use of `element` to be discovered).

However, the fact that such declarations are not *required* before referring to `member` in a sub-concept means that developers (or tools supporting DL) must exercise a certain discipline when building ontologies. This could become a problem when attempting to scale the use of DL large-scale team development of ontologies.

While the above declaration now indicates that the `member` role is associated with the concept `Set`, and therefore provides guidance for users of the ontology, it still does not help to directly detect the mistaken assertion of `(element s a)` in the knowledge base. The following declaration would help:

```
(exists element Anything) ≤̇ PeriodicTable
```

This restricts the domain of `element` to the concept `PeriodicTable` and therefore it can no longer be used to make assertions about sets. However, there is now a more serious problem: Two designers cannot work on independent parts of the ontology without reserving unique names for every role they need to define. This is clearly not a viable approach.

It could be argued that it is not desirable to prevent the simultaneous use of different terms such as `member` and `element` in the

same database. This could be compared with Tim Berners-Lee's notion of the Semantic Web where "anyone can say anything about anything" [2]. However, this requirement does not seem to apply to ontology-based agent communication. The nature of inter-agent communication using an agent communication language is that agents agree on an ontology to use. They know what the applicable roles are for each individual they wish to talk about and can be assumed to converse in it correctly. While it will no doubt be necessary to translate between ontologies, this is probably better handled as a separate process involving specialised translation agents.

In contrast, in UML, the internal structure and relationships between classes are represented by attributes and associations, with attributes being used to represent slots of simple types and associations being used to express relationships between classes. Attributes and associations are not defined globally, but are defined in the context of the particular classes with which they are concerned. Thus, since an attribute of a class is only known in the context of that class, it may be expressed as a simple identifier, such as "name", "member", etc., without the necessity of defining its global semantics for the whole model.

One of the significant features that UML offers is assistance at design time in the management of complexity. This is accomplished by the way in which modelling elements are localised in scope. This feature of design locality makes it more likely that the model designer will be able to construct a more complex design by means of manageable 'chunks', whose complexity is encapsulated within the scope of a class or a few classes. Although a DL may provide more advanced mechanisms for detecting inconsistencies among modelling elements that the designer has constructed, it doesn't offer the guidance in managing complexity that UML does. The design locality that UML inherently offers by means of its structure assists the design in "getting it right from the outset", and the value of this feature should not be underestimated.

4.3 Formal Properties

Much research has been done on the semantics of description logics and the computational complexity of reasoning with them to answer various types of questions about models. This is the main advantage of using DLs for ontology modelling: their well-understood properties. This depth of research has also led to the definition of a wide array of different DLs with varying degrees of expressiveness; however as these generally share a common core of modelling concepts, interoperability shouldn't be a problem.

In contrast, UML currently lacks a formal definition. The semantics of UML are defined by a metamodel, some additional constraints expressed in the Object Constraint Language (essentially a form of first-order logic with an object-oriented syntax), and descriptions of the various elements of the language in English. This shortcoming is being addressed by the Precise UML Group [32] who are actively working on formal semantics for UML.

4.4 Inference Capabilities

There are various possible types of inference that a knowledge representation system could support. This section compares DL and UML in terms of a number of different categories of reasoning.

The first four categories are listed by Bucheit *et al.* [8] as the minimum features that should be offered by a DL-based knowledge representation system:

- **Concept Satisfiability**

  Can a concept C have a non-empty set of instances?

  Detecting when a concept is unsatisfiable can be useful during ontology design to alert the user when an invalid model is being defined.

- **Subsumption**

  Is a given concept description more general or more specific than another, or can no such relation be established?

  The ability to answer this question allows DL systems to perform automatic classification of concepts. When an ontology designer defines a new concept, the system can determine how this concept relates to other existing ones.

- **Knowledge-Base Satisfiability**

  Are the model and the set of recorded instances consistent with each other?

  This check can be used to validate a model using sample instance data. It can also potentially be used during normal agent operation to check that meaningful information is being provided by other agents.

- **Instance Checking**

  Is a an instance of concept C in any model of the knowledge base?

  This is closely related to the previous question and is also concerned with the validation of information.

The first two categories above are applicable during the design of an ontology and are well supported by description logic. In contrast, there are no corresponding analysis capabilities for UML. However, object-oriented modelling is a mature field with well-established and proven methodologies for the design of models. This suggests that such inference abilities are not critical to the process of designing complex models.

It could be possible to equip UML design tools with similar capabilities. However, the structural modelling elements of UML together with the Object Constraint Language provides a formalism that is more expressive than DL and the usual trade-off between representational power and tractability applies. As OCL is "essentially a variant of [first order predicate logic] tuned for writing constraints on object structures" [9], concept satisfiability is only semi-decidable for UML.

It can also be argued that automatic classification based on structural properties of a concept is not always desirable. The subsumption inference leads to problems where the distinction between two concepts cannot be modelled explicitly in the language, but the ontology designer wishes to have a "strongly typed" system in which these concepts are distinct. For example, an ontology may contain two specialisations of collections: sets and lists. What is the subsumption relation between the three concepts if DL does not provide means to describe the differences?

The latter two categories above don't have precise counterparts for UML. In UML an object always has a specific class associated with it and it is (conceptually at least) a structure that must conform to the structural definition of that class. However, in the presence of OCL constraints in the ontology that relate to that class it is

necessary to check these in order to be certain that an object is valid. There are now UML tools emerging [3, 35] that can store instances and check that they conform to any relevant OCL constraints.

The following two additional categories of reasoning are also likely to be of interest to agent systems.

- **Reasoning with Metaproperties**

  Some knowledge representation languages allow slots, roles or properties (depending on the terminology used) to have certain 'metaproperties'. For example, in OIL [31], properties can be declared to be transitive, symmetric or functional. These declarations provide information that an agent could potentially use for deduction.

- **General Rule-Based Reasoning**

  Ontologies expressed in lower-level languages such as KIF can include general axioms. The use of the frame ontology with KIF allows ontologies to be defined using the concept of frames while still retaining the ability to include general logical axioms. In many cases these axioms may simply be considered to be for specification and certification purposes only: any agent advertising that it can communicate using the ontology should always generate conforming instances. However, it is also possible for ontologies to include axioms that agents might plausibly use to perform inference, particularly if–then rules that could be used in backwards chaining inference to answer queries, or in forward chaining inference to generate new information.

Metaproperties can generally be defined using lower-level modelling constructs. In the case of OIL, their meaning is defined by expressions in an expressive description logic. While the underlying description logic is powerful enough to encode these notions using more primitive concepts such as role subsumption, agents are likely to want to make explicit deductions using metaproperties, rather than just having this information included in the more generic deductions described above. Although some DL-based systems can use satisfiability tests to perform more general inference by refutation, this is unlikely to be very efficient. It would be more useful for agents to explicitly take note of metaproperties such as transitivity that might occur in an ontology and use it to generate new knowledge from old. Description logic has nothing to offer in this regard: knowledge representation systems based on DL typically have a separate "ABox" (the "assertional" component) that contains instance data and may have application-specific inference mechanisms that are not formally connected with the DL inference mechanisms.

For UML models, OCL constraints can be used to declare association properties such as transitivity. However, describing such constraints may involve writing moderately complex OCL expressions that are not immediately understandable to a human reader (or an agent). Furthermore, there may be several different expressions encoding the same constraint. One way around this would be to use one of UML's extension mechanisms, stereotypes, to define specialised types of association that are constrained (by OCL) to have certain metaproperties such as transitivity. An agent could then recognise the appearance of (e.g.) a 'transitive' stereotype and make inferences based on this knowledge, even if it could not understand OCL.

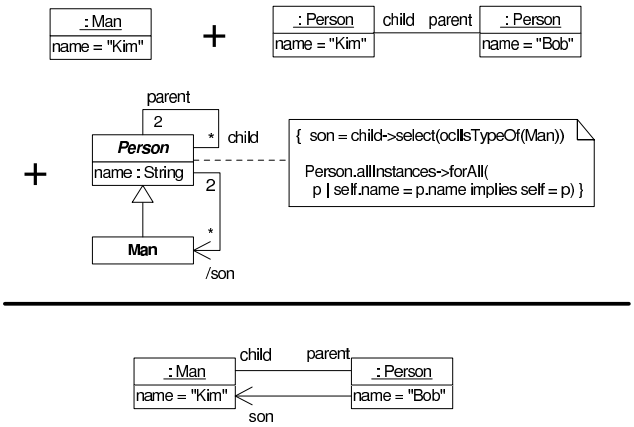

Figure 2: An example of inference over knowledge in UML

Description logics also have no notion of rules, although an ABox component in a knowledge representation system may support them. OIL 1.0 ontologies could include a 'rule-base' but no specification was given for the meaning of the rules or how they relate to the conceptual modelling part of the language. Later versions of OIL and DAML+OIL [17] do not support this, but the definition of a "DAML-L" language (combining conceptual models with semantically-integrated rules) is a long-term aim of the DAML project.

Although UML does not resemble a traditional logic-based language recent research has shown how inference rules in UML can be expressed as graph transformations on the UML metamodel [18, 22]. To give a taste of what inference with UML might look like, Figure 2 shows how new knowledge in the form of an object diagram can be generated by combining existing knowledge and information about the ontology. In this example, one agent has communicated to another that there is an object of class `Man` with "Kim" as the value of its `name` attribute. The other agent knows that there is a `Person` object with name Kim and that this object is the child of a `Person` object with name Bob. The ontology for this domain states that `Man` is a specialisation of `Person`, and includes two OCL constraints: one defining the derived (indicated by a '/') role `son` (a son is a child that is a man), and the other stating (rather unrealistically) that the `name` attribute uniquely identifies objects of class `Person`. Over several steps of inference the agent can conclude that the two objects with name Kim are the same and therefore Kim is a male child, i.e. a son. Implementing this style of deduction in UML is a subject for future research.

5. UML AND AGENT COMMUICATION

In the previous sections we focused on the benefits of using UML from the conceptual modelling perspective. However, in distributed agent systems, besides the ontology, agents need a concrete knowledge exchange language to be able to communicate with each other.

While the standards of the Foundation for Intelligent Physical agents (FIPA) [20] are widely accepted as providing a good foundation for agent communication—ranging from the basic transportation protocol to the speech act level—the encoding of the message content itself is still an open issue.

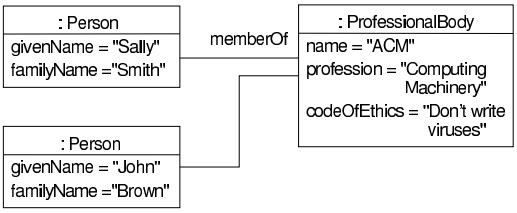

Figure 3: A simple UML object diagram example

The use of an object-oriented ontology representation language for ontology modelling raises interesting questions about the form in which knowledge should be stored within an agent and encoded within inter-agent messages. If an agent developer is using an object-oriented model of the problem domain and is also using a agent development toolkit based on an object-oriented language (such as Java), it may be most convenient to encode domain knowledge using Java objects and to include these within messages to play the role of propositions. There is evidence for a desire amongst agent developers to have this ability: the FIPA content language SL (for "Semantic Language") includes the ability to use functional terms built from object 'constructors' that appear in the ontology, the JADE agent platform [16] allows application-specific Java classes to be associated with concepts in an ontology, and members of the FIPA agentcities mailing list have been discussing the inclusion of ontology-specific objects within XML encodings of messages.

Given a set of ontologies described using UML class diagrams, knowledge about the domains described in these ontologies can be expressed as instances of the classes in the ontologies. This knowledge can therefore be formalised as a UML *object diagram*. In object diagrams, rectangles denote objects, specifying their class and the object's attribute values. The lines between objects show 'links': instances of associations between classes.

Figure 3 shows a simple object diagram showing instances of the classes defined in Figure 1. This can be considered to be a declarative representation of knowledge. It states that there are Person objects with names "Sally Smith" and "John Brown" and that these are both members of the ProfessionalBody object with name "ACM". It also records the profession and code of ethics of this latter object (note: this diagram in no way represents the official policy of the ACM).

For a simple, domain-specific agent, this diagram (or, more accurately, a Java or OODBMS encoding of it) may be the most convenient way to represent knowledge, rather than using a knowledge base containing separate facts such as `class(object1, person)` and `givenName(object1, "Sally")`. The choice between these representations may depend largely on how much (if any) reasoning the agent is expected to do. If general deductive capabilities are required, then a traditional logic-based representation of knowledge is likely to be more suitable. However, this will not be required for all agents.

5.1 Message serialisation via RDF

If two agents represent domain knowledge as object diagrams, it would be most convenient for them to communicate information to each other in this form as well. To provide this ability requires:

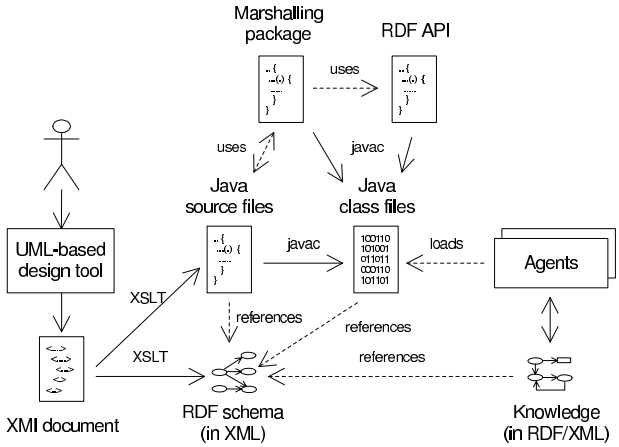

Figure 4: UML "data binding" via an RDF serialisation

1. a concise and convenient representation for object diagrams within an agent (e.g. as networks of Java objects)
2. a concise and convenient serialisation format for object diagrams
3. a marshalling framework to make it easy for agents to convert between 1 and 2.

Requirement 1 can be achieved (for agents developed using Java) by generating from a UML class diagram a set of corresponding Java classes.

A candidate for requirement 2 is the XML Model Interchange (XMI) format which is an XML-based standard for exchanging models. In particular it provides an XML DTD corresponding to the concepts in the UML metamodel. This provides a convenient format for exchanging a model such as an ontology. However, it does not provide a concise and convenient way to encode object diagrams. This is because the UML metamodel has concepts such as Instance, Link and LinkEnd rather than domain-specific concepts such as Person, Organisation, etc. Therefore an XMI encoding of an object diagram has separate cross-referenced entities for each instance, 'attribute link', link and link end and an application has to piece together the application-specific objects from these.

A better solution is to generate a domain-specific schema from the class diagram. One possibility is to generate an XML schema and then Requirement 3 could be satisfied by one of the various Java XML binding [33] implementations under development. However, an alternative approach has been taken based on the simpler data model underlying the Resource Description Framework (RDF) [25]. Figure 4 shows an implemented mechanism [10, 11] for generating RDF schemas and sets of Java classes corresponding to ontologies in UML. The generation is performed using Extensible Stylesheet Language Transformations [36]. The generated classes in conjunction with some additional marshalling support classes allow object diagrams to be converted between in-memory networks of Java objects and RDF serialisations with a single method call.

This technique can be extended to allow complete agent messages (not just the content expression) to be conceptualised as UML ob-

ject diagrams and serialised using RDF. Agent communication languages and content languages can be viewed as an ontologies for communication and knowledge representation, and their abstract syntax can be defined using UML. This approach to language specification could be less error-prone than the grammar-based approach used currently, and would provide a uniform framework for modelling and investigating the relationships between agent communication languages and ontologies [15, 14].

5.2 Message serialisation via SOAP

The RDF serialisation provides a machine-readable and ontology-specific encoding of object-oriented knowledge that can be used for encoding knowledge within messages or for publishing information on the "Semantic Web". However, the RDF format is not ideal for this purpose. One problem is that RDF (like DL) has universal 'properties' which must be used to represent class attributes and association ends. The solution chosen was to generate unique property names within each class by prepending the class name to each UML attribute and rolename. However, this makes the RDF serialisation verbose for a human to read. In addition there are a number of other factors that complicate human reading and machine parsing of RDF files [23].

An alternative is to use the serialisation format of the Simple Object Access Protocol (SOAP) [6]. SOAP is a specification covering remote procedure calls over HTTP. It contains an object serialisation format that can be compared to the Resource Description Format (RDF) to some extent, although RDF is more than just an object serialisation format.

SOAP is supported by computer industry leaders like Microsoft, IBM and SUN. The simplicity of SOAP together with the strong support from the industry suggests that many SOAP-based services will be available in the near future. Although SOAP is an relatively young technology, implementations for many programming languages are available. SOAP is suited for all kinds of automated internet services like weather forecasts, traffic services, and logistics coordination, to give just a few examples.

The main difference from RDF is that SOAP does not rely on globally unique slot names, thus the inclusion of the class name in the property name is not neccessary.

Figure 5 shows the SOAP serialisation of the object diagram shown in Figure 3.

```
<Person id="123">
  <givenName>Sally</givenName>
  <familyName>Smith</familyName>
  <memberOf>
    <element href="#789" />
  </memberOf>
</Person>

<Person id="456">
  <givenName>John</givenName>
  <familyName>Brown</familyName>
  <memberOf>
    <element href="#789" />
  </memberOf>
</Person>

<ProfessionalBody id="789">
  <name>ACM</name>
  <profession>Computing Machinery</profession>
  <codeOfEthics>Don't write viruses</codeOfEthics>
  <member>
    <element href="#123" />
    <element href="#456" />
  </member>
</ProfessionalBody>
```

Figure 5: SOAP encoding of the object diagram

6. CONCLUSION

This paper has discussed the application of the Unified Modeling Language to ontology modelling for agent systems. The characteristics of UML and description logic—the most widely used traditional ontology modelling language—were compared. Although UML may not currently have the support for reasoning that is offered by description logic, we believe that UML's strengths in many other areas make it suitable as a candidate for general adoption for ontology representation. In particular its features that assist in the modelling of complex systems and its association with many other standard business technologies suggest that there is a significant advantage to using UML in comparison with other approaches currently in use.

The paper also discussed the use of UML for representing instance information within agent messages and presented some technologies to support this style of knowledge representation.

7. REFERENCES

[1] 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/confs/ESAW00/pdf/ESAW13.pdf).

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

[3] BoldSoft. ModelRun product Web page. http://www.boldsoft.com/products/modelrun/, 2001.

[4] G. Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2nd edition, 1994.

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

[6] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. Note, World Wide Web Consortium, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[7] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April 1985.

[8] M. Buchheit, F. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[9] T. Clark, A. Evans, S. Kent, S. Brodsky, and S. Cook. A feasibility study in rearchitecting UML as a family of

languages using a precise OO meta-modeling approach. Report, Precise UML Group, September 2000. http://www.cs.york.ac.uk/puml/mml/mmf.pdf.

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

[11] S. Cranefield. UML and the Semantic Web. Discussion Paper 2001/04, Department of Information Science, University of Otago, 2001. http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2001-04.pdf.gz.

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

[13] S. Cranefield and M. Purvis. Extending agent messaging to enable OO information exchange. In R. Trappl, editor, *Cybernetics and Systems 2000: 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, 2000. Austrian Society for Cybernetic Studies. An earlier version is available at http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2000-07.pdf.gz.

[14] S. Cranefield and M. Purvis. Generating ontology-specific content languages. In *Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents*, 2001. http://autonomousagents.org/2001/oas.

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

[16] CSELT. The JADE agent platform Web site. http://sharon.cselt.it/projects/jade/, 2001.

[17] DARPA Technology Integration Center. DAML project Web site. http://www.daml.org, 2001.

[18] A. S. Evans. Reasoning with UML class diagrams. In *Proceedings of the Workshop on Industrial Strength Formal Methods (WIFT'98)*. IEEE Press, 1998. http://www.cs.york.ac.uk/puml/papers/evanswift.pdf.

[19] A. Farquhar, R. Fikes, and J. Rice. The Ontolingua Server: a tool for collaborative ontology construction. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[20] Foundation for Intelligent Physical Agents. FIPA web site. http://www.fipa.org/specs/fipa00023/XC00023F.pdf.

[21] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[22] M. Gogolla. Graph transformations on the UML metamodel. In *Proceedings of the ICALP Workshop on Graph Transformations and Visual Modeling Techniques (GVMT'2000)*, pages 359–371. Carleton Scientific, 2000. ftp://ftp.informatik.uni-bremen.de/%2Flocal/db/papers/Gogolla_2000_GraGra.ps.

[23] S. Haustein. Semantic Web languages: RDF vs. SOAP serialisation. In *Proceedings of the Second International Workshop on the Semantic Web*, pages 14–22, May 2001. http://semanticweb2001.aifb.uni-karlsruhe.de/semanticWebWorkshop2001.pdf.

[24] Knowledge Systems Laboratory. The Frame Ontology. ftp://ftp.ksl.stanford.edu/pub/knowledge-sharing/ontologies/html/frame-ontology/frame-ontology.lisp.html, 1994.

[25] O. Lassila and R. R. Swick. Ressource Description Framework (RDF) model and syntax specification. Technical report, World Wide Web Consortium, 1999. http://www.w3.org/TR/1999/REC-RDF-SYNTAX-19990222.

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

[27] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Number 422 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1990.

[28] Object Management Group. OMG Unified Modeling Language Specification, version 1.3. http://www.omg.org/technology/documents/formal/unified_modeling_language.htm, 2000.

[29] Object Management Group. OMG Web site. http://www.omg.org/, 2001.

[30] Object Management Group. UML textual notation RFP. http://www.omg.org/techprocess/meetings/schedule/UML_Textual_Notation_RFP.html, 2001.

[31] On-To-Knowledge project. OIL web site. http://www.ontoknowledge.org/oil, 2000.

[32] Precise UML Group. The Precise UML Group home page. http://www.puml.org, 2001.

[33] M. Reinhold. XML data binding specification. Java Specification Request JSR-000031, Sun Microsystems, 1999. http://java.sun.com/aboutJava/communityprocess/jsr/jsr_031_xmld.html.

[34] Technische Universität Dresden. Dresden OCL toolkit. http://sourceforge.net/projects/dresden-ocl/, 2000.

[35] University of Bremen. A UML-based specification environment. http://www.db.informatik.uni-bremen.de/projects/USE/, 2001.

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