UML as an Ontology Modelling Language *

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

Abstract

Current tools and techniques for ontology development are based on the traditions of AI knowledge representation research. This research has led to popular formalisms such as KIF and KL-ONE style languages. However, these representations are little known outside AI research laboratories. In contrast, commercial interest has resulted in ideas from the object-oriented programming community maturing into industry standards and powerful tools for object-oriented analysis, design and implementation. These standards and tools have a wide and rapidly growing user community. This paper examines the potential for object-oriented standards to be used for ontology modelling, and in particular presents an ontology representation language based on a subset of the Unified Modeling Language together with its associated Object Constraint Language.

1 Introduction

In recent years a number of subfields of artificial intelligence have been aiming to increase the ability of their systems to interact with humans and other external agents by developing and sharing *ontologies* — formally specified models of bodies of knowledge defining the concepts used to describe a domain and the relationships that hold between them. Research areas investigating the design of ontologies include agent-based software interoperability [Genesereth and Ketchpel, 1994], knowledge acquisition [SMI, 1998] and natural language processing [Bateman *et al.*, 1995].

Various formalisms have been developed for expressing ontologies, notably the Knowledge Interchange Format (KIF) [NCITS, 1998] and knowledge representation languages descended from KL-ONE [Brachman and Schmolze, 1985]. In this paper we examine the use of an alternative formalism for representing ontologies: a subset of the Object Management Group's Unified Modelling Language (UML) together with its associated Object Constraint Language (OCL). Object-oriented analysis, design and implementation is a maturing field with many industry standards emerging for distributed computation. The large user community and commercial support for object-oriented standards warrants the investigation of standard object modelling techniques for ontology development.

This work is motivated primarily by consideration of the role that ontologies play in agent-based infrastructures for supporting queries over open and dynamic collections of heterogeneous and distributed information sources. Systems such as SIMS [Knoblock and Ambite, 1997], Infosleuth [Bayardo *et al.*, 1997] and Observer [Mena *et al.*, 1999] use ontologies to model the semantic structure of individual information sources, as well as to describe models of a domain that are independent of any particular information source. The challenges for these systems are to support the construction of user queries using domain ontologies that may be initially unfamiliar to the user, and to allow queries to span multiple information sources by representing and computing the mappings between domain ontologies and the ontologies supported by individual information sources.

2 Common Ontology Modelling Languages

The most common formalisms used to represent ontologies are the Knowledge Interchange Format (KIF) [NCITS, 1998] and KL-ONE style knowledge representation languages [Brachman and Schmolze, 1985].

KIF provides a Lisp-like syntax for expressing sentences of first order predicate logic and also provides extensions for representing definitions and metaknowledge. KIF is a highly expressive but low-level language for representing ontologies; however, the Stanford University Knowledge Sharing Laboratory's ontology editing tool, Ontolingua [Farquhar *et al.*, 1996], allows users to build KIF ontologies at a higher level of description by importing predefined ontologies defining concepts such as sets, standard units, time and simple geometrical functions. In particular, the *frame ontology* [KSL, 1994] allows ontologies to be defined in terms of relations, classes (and subclasses), functions and sets.

Much of the research on ontology design and use is performed by researchers using knowledge representation tools descended from KL-ONE [Brachman and Schmolze, 1985]. KL-ONE was the basis for much work in the field of knowledge representation. It implemented "structural inheritance

*This paper will appear in the on-line proceedings of the IJCAI-99 Workshop on Intelligent Information Integration

networks": networks containing descriptions of named concepts with generalisation/specialisation links between them. Descendants of KL-ONE include LOOM [ISI, 1998] and a family of logics called *description logics* or *terminological logics* [Donini *et al.*, 1996; Owsnicki-Klewe, 1990]. The KIF frame ontology discussed above also allows this type of specification to be used in conjunction with more general KIF sentences.

In a description logic, concepts can be introduced by simply naming them and specifying where they fit in the generalisation/specialisation hierarchy. The following examples are adapted from Nebel [1990]:

$$\texttt{Human} \doteq \texttt{Anything}$$
$$\texttt{Set} \doteq \texttt{Anything}$$

where $\doteq$ represents concept specialisation and `Anything` is a predefined concept representing the class of all things.

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:

$$\texttt{Male-student} \doteq \texttt{(and Man Student)}$$

New *roles* may be introduced to represent possible relationships that may hold between instances of a concept and other individuals in the world, for example:

$$\texttt{member} \doteq \texttt{anyrelation}$$

where `anyrelation` represents the class of all relations.

Concepts may be specialised by operations such as *value restriction*, where the operator `all` is used to restrict a role's possible values to be instances of a certain class, and *number restriction*, where the operators `atleast` and `atmost` are used to restrict the possible number of values that a given role may have. The following example states that a team is a set for which all values for its "member" role are instances of the `Human` concept with the cardinality of the member role being at least two.

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

Systems such as KL-ONE and LOOM structure their knowledge bases to allow certain types of inferences to be performed efficiently on the user-defined concepts, such as the following list paraphrased from Owsnicki-Klewe [1990]:

- Subsumption: Is a given concept description more general or more specific than another, or can no such relation be established?
- Coherence: Is a concept description logically coherent, i.e. can there be an instance of this term?
- Identity: Do two concept descriptions refer to the same concept?
- Compatibility: Can two concept descriptions have common instances?
- Common specialisation: What are the properties of the common specialisation of two concept descriptions?

These types of deduction are designed to help the user in incrementally designing a coherent set of concepts and instances to describe a domain. Description logics provide a formal characterisation of the representational and deductive capabilities of KL-ONE style systems and allow their computations to be studied in terms of completeness, computational complexity, etc. Although domain knowledge could be represented using first order predicate logic, the benefit of using a specialised representation is that special-purpose data structures and algorithms can be used to support efficient reasoning. In addition, the structured knowledge base supports efficient processing of declarative queries about the defined concepts.

3 UML for Ontology Modelling

Knowledge representation (KR) systems such as LOOM are large and complex systems with a steep learning curve and are little known outside AI laboratories. Instead of using such technology, the authors are investigating the more mainstream and rapidly growing arena of object-oriented technology to construct a distributed information retrieval and processing system. Currently there is no counterpart for the deductive capabilities of KR systems in current object-oriented technology; however, for distributed information systems these capabilities are not necessarily needed. Many of the benefits of KR systems occur during the process of designing an ontology. This support is undoubtedly useful, but in the object-oriented world there is also much support available for the design of models, with mature and commonly used languages, methodologies and tools available.

The other function of KR systems — to store highly structured data and answer queries about it — is not an issue in distributed information systems. The point of systems such as SIMS, Infosleuth and Observer is to allow disparate databases and other information sources to be integrated. Nothing can or should be assumed about the underlying databases and information storage systems. In particular, it cannot be assumed that the information sources will be implemented using KR systems. While systems such as LOOM can be used to implement key components of a distributed information system infrastructure (such as query planning agents), it is certainly possible to use other reasoning engines. In the authors' view, unless a system that uses ontologies is constructed around a tool such as LOOM, there seems to be nothing inherently intuitive or appealing in the use of a description logic formalism to represent ontologies.

The ontology representation formalism presented in this paper is a subset of the Unified Modeling Language (UML) [Rumbaugh *et al.*, 1998] from the Object Management Group (OMG) [OMG, 1998], together with its associated Object Constraint Language (OCL) [OMG, 1997b; Warmer and Kleppe, 1998]. Benefits of using UML and OCL include the following:

- UML has a very large and rapidly expanding user community. Users of distributed information system infrastructures will be more likely to be familiar with this notation than KIF or description logics. This issue should not be overlooked for its importance in gaining acceptance

of distributed information systems technology amongst new end-user communities.

- Unlike description logic formalisms, there is a standard graphical representation for models expressed in UML. Such a graphical representation is important to allow users of distributed information systems to browse an ontology and discover concepts that can appear in their queries. In contrast, a description logic has a linear syntax but no standard graphical representation. Although UML currently has no standard linear syntax, the OMG is in the process of adopting XMI (XML Model Interchange) as a standard for stream-based model interchange [DSTC, 1999].
- The Object Constraint Language (OCL) is powerful and allows the expression of constraints that cannot be described using description logic. Of course, there is a trade-off between the expressive power of a language and the computational complexity of reasoning about it. This issue is discussed in Section 3.5.

3.1 An Overview of UML and OCL

UML defines several types of diagram that can be used to model the static and dynamic behaviour of a system. We have chosen to model an ontology as a static model consisting of a class diagram to depict the classes in the domain and their relationships, and an object diagram to show particular named instances of those classes. A sample class diagram appears in Figure 1. Section 3.1 explains the classes and relationships shown in this diagram. In this section we describe the UML notation used in Figure 1.

In a *class diagram*, classes are represented by boxes with three parts: the name of the class, the attributes of the class (specified by their name, type and visibility) and the operations of the class (specified by name, argument list, return type and visibility). For the purposes of representing ontologies, all attributes can be considered to have public visibility — an ontology is a shared public view of a domain. At present we do not use operations in our ontologies, although these could be used in conjunction with OCL postcondition constraints that specify the result of the operation. If operations are included, it is possible to declare that they are *queries*, i.e. they will not change the state of the object the operation is invoked on.

Figure 1 shows three types of relationship that may be used between classes:

- *generalisation*, represented by lines with large hollow arrow heads pointing to the super class (e.g. see classes `Role` and `InterpretiveRole` at the top of the figure);
- *association*, represented by solid lines between two classes with optionally named ends, or *roles* (e.g. class `Realisation` in the middle of the figure has an association with class `Work` to its right);
- *aggregation*, an association with a diamond at the aggregate end of the link (e.g. class `CD` on the left of the figure has an aggregation relationship with `ItemOnCD` to its right). UML includes a stronger type of aggregation (composite aggregation, notated by a solid black diamond) which implies ownership of the parts by the aggregate. We do not make a distinction between the two types of aggregation in our ontologies at present.

The ends of association and aggregation relationships may be annotated with multiplicity indicators giving a range of numbers (with '$*$' representing infinity) denoting how many instances of the class at that end of the relationship can be associated with each instance of the class at the other end. Also, a small barbed arrow head may be used to specify that an association or aggregation relationship may only be navigated in one direction (this feature is not used in Figure 1).

Several other constructs of UML are used in Figure 1. Class `CreativeAct` in the top right corner is an *association class*: a class attached to an association. These can be used for associations that require attributes (e.g. an association between two classes `student` and `assignment` might have a grade attribute). In the case of Figure 1, association classes are used for associations that themselves participate in an association with another class.

Finally, the large rectangles with folded corners are *notes*: uninterpreted pieces of text that may be anchored with dashed lines to model elements to provide informal clarification. In this case, however, the notes are used to attach OCL constraints to classes and associations. This is necessary as the diagram was produced using Rational Rose 98 which does not provide a general facility for placing OCL constraints on a model.

A UML *object diagram* depicts objects and *links* between objects — instances of the relationships that hold between the linked objects' respective classes. The class of each object included in the diagram must be specified and the object may optionally be named. The values of the object's attributes must be shown. UML itself does not define a standard set of primitive types for attribute and operation declarations; however, the Object Constraint Language does, and it is proposed that these be used for ontology modelling with UML.

In a class diagram, OCL may be used to constrain attribute values and possible instances of the relationships. It is beyond the scope of this paper to give a comprehensive discussion on OCL, but a brief overview follows.

An OCL expression is written in the context of an instance of a specific type. The name 'self' is used to refer to that instance. The value of an instance's attribute can be expressed by following the expression naming the instance with a dot and the attribute's name. The dot notation can also be used to traverse an association or aggregation relationship. In this case, the dot is followed by either the name of the class at the far end of the relationship (with the initial letter changed to lower case) or by the name of the role at that end of the relationship (if it is named). The resulting expression can represent a single instance (if the multiplicity of that role has an upper limit of 1), a set of instances (when traversing roles with other multiplicity indicators), or a sequence of instances (for roles labelled with the constraint "ordered"). Given an expression representing a collection (a set, sequence or bag), the arrow operator `->` can be used to invoke one of a number of standard functions and predicates on that collection, e.g. `collection->size`.

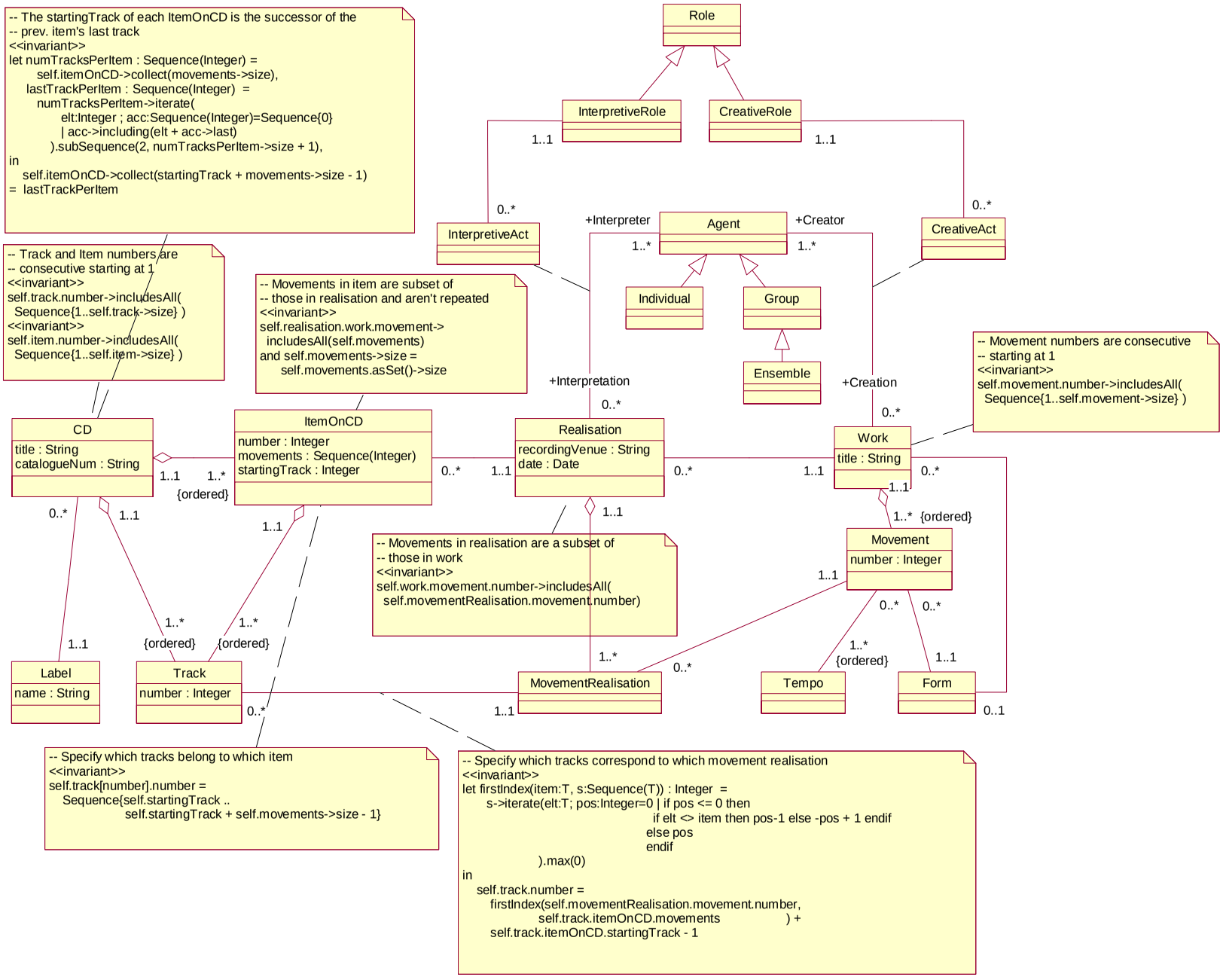

Figure 1: UML class diagram for a CD publisher's catalogue

3.2 Example

Figure 1 shows an example UML class diagram modelling the concepts and relationships in the catalogue system for a classical music compact disc publisher. This model concentrates on three classes:

Work
This class (located centre right) represents a piece of music, i.e. the work of art itself. It is an aggregation of `Movement` objects (which are ordered sequentially starting at 1) and will have one or more `CreativeAct` relationships associating it with an `Agent` and a `Role` (instances will include roles with name "composer", "librettist", etc.). `CreativeAct` might be better represented as a ternary relationship rather than an association class with an association to `Role`, but Rational Rose 98 does not support the UML n-ary relationship notation.

Realisation
This class (located to the left of `Work`) represents a particular recording of a work (or part of a work) made by the recording company. It consists of `MovementRealisation` objects that are in turn associated with `Movement` objects. It has one or more `InterpretiveAct` relationships that associate it with an agent in some `InterpretiveRole` such as conductor, performer or narrator.

CD
This class (centre left) consists of an ordered sequence of objects of class `ItemOnCD`. An `ItemOnCD` object represents a whole or partial instance of a work appearing on a `CD`. Note that a `Realisation` (in whole or part) may occur on more than one `CD`. A `CD` is an aggregate of (ordered) `Track` objects, and so is an `ItemOnCD`. The OCL constraint in the lower left corner specifies which of a `CD`'s tracks belong to each item on the `CD` (i.e. this constraint restricts the possible instances of the aggregation relationship between `ItemOnCD` and `Label`). A `CD` is also associated with a recording `Label` (e.g. Naxos).

Some of the classes in the diagram are incomplete (containing no attributes) and a full version of the ontology would show more details such as `name` attributes for the classes `Agent`, `Role`, `Tempo` and `Form`.

3.3 Required Extensions to UML and OCL

UML allows ends (or "roles") of association and aggregation links to be annotated with the constraint "ordered", meaning that navigating that role from an object results in a sequence of objects rather than a set. However, there is no syntax defined in an object diagram to specify the actual ordering on the instances of that relationship. This would be simple to include by allowing a new constraint type "precedes" to relate two association links in an object diagram.

OCL contains some predefined functions on collections of objects, as well as a simple "mapping" function on bags, sets and sequences called "iterate". This iterates over the collection, using an expression involving the current element to modify a single accumulator value at each step. However, this function is highly frustrating to use due to its support for only a single accumulator value. This problem could be solved if a tuple type were introduced to OCL. An accumulator could then be a tuple of several different values.

OCL lacks the facility to use temporary variables and functions to avoid having to repeat subexpressions in an expression. The example in Figure 1 uses a non-standard "let" construct to solve this problem. The need for a "let" construct has also been noted by Hamie *et al.* [1998a], along with a number of other shortcomings of OCL and some proposed solutions.

3.4 Semantics of UML

As an ontology is a formal model of a domain, it is important that the language used to describe it has formal semantics. Unfortunately the official OMG document 'defining' the semantics for UML gives an informal description in English [OMG, 1997a]. This shortcoming is currently being addressed by a number of researchers who have proposed various different forms of semantics for UML, including a direct mathematical model of the system being described in UML [Breu *et al.*, 1997], a description using the specification language Z [Evans *et al.*, 1998] and operational semantics describing how a UML model evolves as new elements are added to it [Övergaard, 1998].

Semantics for OCL, which necessarily include semantics for class diagrams, have been proposed by Richters and Gogolla [1998] and Hamie *et al.* [1998b].

3.5 Reasoning about Ontologies in UML

When choosing an ontology representation language, it is not sufficient only to consider the ease with which the language can be used to describe the domain. It is also necessary to consider the types of automated reasoning about ontologies that may be required. There is a well-known tradeoff between the representational power of a formalism and the tractability (and even the solvability) of reasoning with it [Levesque and Brachman, 1985].

For example, KIF provides all the expressive power of first order predicate logic, but reasoning about ontologies in plain KIF requires general theorem-proving capabilities. In contrast, description logic provides a much more structured and less general language for describing ontologies, and therefore specialised inferences can be performed on ontologies described using description logic. Much research has been undertaken to investigate the computational properties of various types of inferences on different variants of description logic [Nebel, 1990].

The ontology representation language used in this paper — a UML class diagram (containing OCL constraints) in conjunction with an object diagram — contains both a highly structured model that could support automated reasoning (the basic class and object model, ignoring the OCL constraints) and an expressive language that it would not be practical to attempt general-purpose reasoning reason with. Further research is needed to clarify what types of inference it would be desirable and possible to support for ontologies represented in UML. This partly depends on the type of system the ontologies are intended for. We do not suggest that UML be considered as an alternative to description logic formalisms in all situations. For example, although Haimowitz et al. [1988] found a KR tool to be inadequate for ontology modelling in a medical expert system, UML would not provide a straightforward alternative for modelling ontologies such as this that

form part of a deductive system. It would either be necessary to express the semantics of UML class diagrams within the deductive system's logic (which would increase the complexity and length of its deductions) or a hybrid system would have to be constructed so that inferences that can be made due to the (implicit) semantics of the ontology can be integrated with the explicit deductive reasoning of the system.

For systems where the required type of reasoning about ontologies can be restricted to answering specific specialised questions, UML is a stronger candidate. However, it remains to identify the questions we would like answered about our ontologies. Consider the example of a distributed information retrieval system — there are several stages at which particular inferences about ontologies may be needed:

- The initial construction of the ontology. This is the area well supported by description logics which provide inference mechanisms for checking the integrity of the ontology as it is constructed. Would similar capabilities be useful for object-oriented modelling with UML and is there a reason why current object-oriented modelling methodologies have not included the use of such mechanisms?
- Assisting users to form queries within an ontology. For example, it may be useful for the system to help users discover concepts that can appear in queries, e.g. by finding and displaying all shortest navigation paths from a given class to classes or attributes with names matching a user-supplied pattern.
- Decomposing and translating queries expressed in one or more high-level domain ontologies into a query plan involving ontologies for specific data sources. This requires both a representation for the relationships between ontologies and a mechanism for reasoning about them.

We expect that the sort of reasoning required for distributed information systems could be performed using the class and object diagrams alone. In many cases the OCL constraints can be regarded as extra detail specifying how systems that implement the ontology should behave. For example, the class diagram in Figure 1 states that a CD contains ItemOnCD objects as well as tracks. Each ItemOnCD object also contains a subset of the CD's tracks. An OCL constraint specifies which of the CD's tracks are associated with each item. This constraint is an important part of the ontology when viewed as a *specification*. Any implemented system that claims to support this ontology must respect this constraint. However, for the purposes of information retrieval, this constraint can be ignored as an implementation detail.

Alternatively, it may be possible to define a set of standard OCL constraints forming a language that can be supported by automated reasoning, such as the types of slot constraints provided by description logic. This would be equivalent to using the frame ontology with Ontolingua: KIF plus the frame ontology can be seen as a higher level language that can be translated to other structured formalisms such as LOOM (provided that other, plain KIF sentences, are not also included in the ontology). This is an important subject for future research.

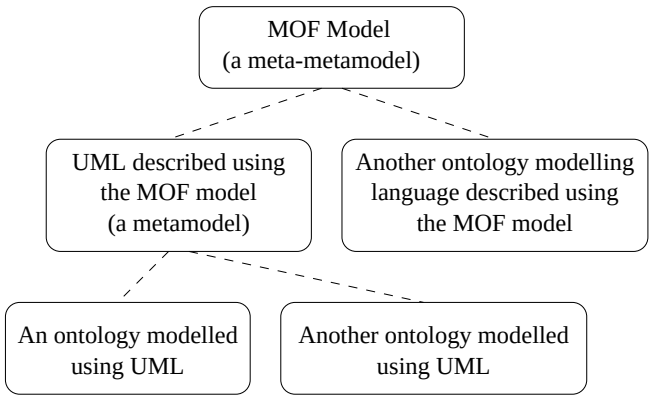

Figure 2: A MOF-based ontology repository

4 Supporting Multiple Ontology Languages

A single ontology representation language is not necessarily convenient for modelling all domains. It may be useful to have several ontology representation languages available to the ontology designer. The Infosleuth project has an interesting approach to supporting multiple modelling languages [Bayardo *et al.*, 1997]. A simple frame-based language is used to define specific ontology representation languages such as object models and entity-relationship diagrams. The actual ontologies are then expressed as instances of these languages. This is a three layer model, with the frame layer acting as a meta-metamodel, the definitions of the ontology representation languages being metamodels and the ontologies themselves being models.

A similar facility is offered by the OMG's Meta Object Facility (MOF) [OMG, 1997c; Crawley *et al.*, 1997; DSTC, 1998]. The MOF defines a standard for CORBA-based services to manage meta-information in a distributed environment. It defines a model (in fact a meta-meta model) that can be used to describe modelling languages such as UML. It also defines interfaces that can be used to populate and query repositories of models defined using various languages. We intend to use this framework to build an ontology server agent with similar capabilities to those of the Infosleuth project. Figure 2 shows the structure of a MOF-based ontology server.

The OMG is currently selecting a standard "Stream-based Model Interchange Format" [OMG, 1999] for the interchange of MOF-based models and metamodels. XMI (XML Model Interchange) is likely to be adopted [DSTC, 1999].

5 Conclusion

We have investigated the use of UML and OCL for the representation of information system ontologies and have constructed an example ontology in the domain of a cataloguing system for classical music compact discs. UML and OCL show promise for representing the kinds of relationships and constraints that are familiar to systems builders. Future research includes investigating the potential for reasoning about ontologies expressed using UML — either ignoring the OCL constraints, or by recognising specific forms of constraints that are amenable to automated reasoning.

References

[Bateman *et al.*, 1995] John A. Bateman, Renate Henschel, and Fabio Rinaldi. The generalized upper model 2.0. http://www.darmstadt.gmd.de/publish/komet/genum/newUM.html, 1995.

[Bayardo *et al.*, 1997] R. J. Bayardo, Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. Infosleuth: agent-based semantic integration of information in open and dynamic environments. In Joan Peckham, editor, *Proceedings of the ACM SIGMOD international conference on management of data*, SIGMOD Record 26(2), pages 195–206, June 1997.

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

[Breu *et al.*, 1997] Ruth Breu, Radu Grosu, Franz Huber, Bernhard Rumpe, and Wolfgang Schwerin. Towards a precise semantics for object-oriented modeling techniques. In Haim Kilov and Bernhard Rumpe, editors, *Proceedings ECOOP'97 Workshop on Precise Semantics for Object-Oriented Modeling Techniques*, pages 53–59. Technische Universität München, TUM-I9725, 1997.

[Crawley *et al.*, 1997] Stephen Crawley, Simon McBride, and Kerry Raymond. Meta-Object Facility tutorial (draft). http://www.dstc.edu.au/Meta-Object-Facility/Tutorial.html, 1997.

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

[DSTC, 1998] Distributed Systems Technology Centre. Meta Object Facility frequently asked questions. http://www.dstc.edu.au/Meta-Object-Facility/MOFAQ.html, 1998.

[DSTC, 1999] Distributed Systems Technology Centre. XMI spec recommended. News item on Meta-Object Facility Information Web Page, http://www.dstc.edu.au/Meta-Object-Facility/, January 1999.

[Evans *et al.*, 1998] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. Developing the UML as a formal modelling notation. In Pierre-Alain Muller and Jean Bézivin, editors, *Proceedings of UML'98 International Workshop, Mulhouse, France, June 3 - 4, 1998*, pages 297–307. ESSAIM, Mulhouse, France, 1998.

[Farquhar *et al.*, 1996] Adam Farquhar, Richard Fikes, and James 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.

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

[Haimowitz *et al.*, 1988] Ira J. Haimowitz, Ramesh S. Patil, and Peter Szolovits. Representing medical knowledge in a terminological language is difficult. In *Proceedings of the Symposium on Computer Applications in Medical Care*, pages 101–105. IEEE Computer Society Press, 1988.

[Hamie *et al.*, 1998a] Ali Hamie, Franco Civello, John Howse, Stuart Kent, and Richard Mitchell. Reflections on the Object Constraint Language. In Pierre-Alain Muller and Jean Bézivin, editors, *Proceedings of UML'98 International Workshop, Mulhouse, France, June 3–4, 1998*, pages 137–145. ESSAIM, Mulhouse, France, 1998.

[Hamie *et al.*, 1998b] Ali Hamie, John Howse, and Stuart Kent. Interpreting the Object Constraint Language. In *Proceedings of the 5th Asia Pacific Software Engineering Conference (APSEC'98)*. IEEE Press, 1998.

[ISI, 1998] Information Sciences Institute. Loom project home page. http://www.isi.edu/isd/LOOM/LOOM-HOME.html, 1998.

[Knoblock and Ambite, 1997] C. A. Knoblock and J. L. Ambite. Agents for information gathering. In J. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, 1997.

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

[Levesque and Brachman, 1985] Hector J. Levesque and Ronald J. Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised version). In Ronald J. Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*, pages 42–70. Morgan Kaufman, 1985.

[Mena *et al.*, 1999] E. Mena, A. Illarramendi, V. Kashyap, and A. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Distributed and Parallel Databases*, 1999. (to appear).

[NCITS, 1998] National Committee for Information Technology Standards, Technical Committee T2 (Information Interchange and Interpretation). Draft proposed American national standard for Knowledge Interchange Format. http://logic.stanford.edu/kif/dpans.html, 1998.

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

[OMG, 1997a] Object Management Group. UML semantics, version 1.1. ftp://ftp.omg.org/pub/docs/ad/97-08-04.pdf, September 1997.

[OMG, 1997b] Object Management Group. Object Constraint Language specification. ftp://ftp.omg.org/pub/docs/ad/97-08-08.pdf, September 1997.

[OMG, 1997c] Object Management Group. MOF specification. http://www.omg.org/techprocess/meetings/schedule/Technology_Adoptions.html#tbl_MOF_Specification, 1997.

[OMG, 1998] Object Management Group. OMG homepage. http://www.omg.org/, 1998.

[OMG, 1999] Object Management Group. Stream-based model interchange Web page. http://www.omg.org/techprocess/meetings/schedule/Stream-based_Model_Interchange.html, 1999.

[Övergaard, 1998] Gunnar Övergaard. A formal approach to relationships in the Unified Modeling Language. In Manfred Broy, Derek Coleman, Tom S. E. Maibaum, and Bernhard Rumpe, editors, *Proceedings PSMT'98 Workshop on Precise Semantics for Modeling Techniques*. Technische Universität München, TUM-I9803, 1998.

[Owsnicki-Klewe, 1990] Bernd Owsnicki-Klewe. A general characterisation of term description languages. In K.-H. Bläsius, U. Hedtstück, and C. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, number 418, pages 183–189. Springer-Verlag, 1990.

[Richters and Gogolla, 1998] Mark Richters and Martin Gogolla. On formalizing the UML Object Constraint Language OCL. In Tok Wang Ling, Sudha Ram, and Mong Li Lee, editors, *Proc. 17th Int. Conf. Conceptual Modeling (ER'98)*. Lecture Notes in Computer Science, number 1507, Springer-Verlag, 1998.

[Rumbaugh *et al.*, 1998] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[SMI, 1998] Stanford Medical Informatics. The Protégé project. http://smi-web.stanford.edu/projects/protege/, 1998.

[Warmer and Kleppe, 1998] Jos B. Warmer and Anneke G. Kleppe. *The Object Constraint Language: Precise Modeling With UML*. Addison-Wesley, 1998.