Towards a Data Consistency Modeling and Testing Framework for MOF Defined Languages

Jan Pettersen Nytun[1,2] Christian S. Jensen[1,3] Vladimir A. Oleshchuk[1]

[1]Faculty of Engineering and Science, Agder University College, Norway
[2]Faculty of Engineering, University of Oslo, Norway
[3]Department of Computer Science, Aalborg University, Denmark

Abstract

The number of online data sources is continuously increasing, and related data are often available from several sources. However accessing data from multiple sources is hindered by the use of different languages and schemas at the sources, as well as by inconsistencies among the data. There is thus a growing need for tools that enable the testing of consistency among data from different sources.

This paper puts forward the concept of a framework, that supports the integration of UML models and ontologies written in languages such as the W3C Web Ontology Language (OWL). The framework will be based on the Meta Object Facility (MOF); a MOF metamodel (e.g. a metamodel for OWL) can be input as a specification, the framework will then allow the user to instantiate the specified metamodel.

Consistencies requirements are specified using a special modeling technique that is characterized by its use of special `Boolean` class attributes, termed consistency attributes, to which OCL expressions are attached. The framework makes it possible to exercise the modeling technique on two or more legacy models and in this way specify consistency between models. Output of the consistency modeling is called an integration model which consist of the legacy models and the consistency model. The resulting integration model enables the testing of consistency between instances of legacy models; the consistency model is automatically instantiated and the consistency attribute values that are false indicates inconsistencies.

1 Introduction

The Semantic Web [1] aims at giving well-defined meaning to web content, in this way allowing automatic reasoning about, and processing of, web content. An important aspect of supporting this is the provisioning of appropriate knowledge representation [2] languages, which remains an active area of research. Prominent examples of such languages include the Resource Description Framework, RDF [3], the DAML+OIL [4] language, which integrates the US DARPA Agent Markup Language and the European OIL effort and is an extension of the RDF Schema, and DAML+OIL's successor, the World Wide Web Consortium's Web Ontology Language (OWL) [5].

Somewhat unrelated to this, the Unified Modeling Language (UML) is being used widely for conceptual modeling in the development of software systems. It may be noted that UML has substantial semantic overlap with knowledge representation languages such as those just mentioned, although there are also differences [6, 7].

The Object Management Group (OMG) recently issued a request for proposals [6] that seeks:

- A standard, Meta Object Facility (MOF) 2.0 [8] compliant metamodel for Ontology Definition (ODM).
- A UML 2.0 [9] (UML for short) Profile that supports reuse of UML notation for ontology definition.
- A mapping from the ODM to the profile.

The OMG request also seeks a language mapping for the ODM to OWL. There are good reasons to reuse the UML notation for ontology definition [10]. For example, the graphical notation of UML is well tested and tools exist that support UML.

There is a trend towards the use of languages that are tailored for special problem domains and also towards integration of different languages (as indicated by the latest request for proposals from OMG); in an OMG context this can be done by using the extension mechanisms of UML, definitions of UML profiles, and also definition of new MOF metamodels. Tools that support definition and application of this type of languages are largely missing.

This paper takes the first steps towards defining a framework for experimenting with the integration of UML and knowledge representation languages. The framework should contain components that can be assembled to form different tools.

Our selected application is consistency testing of user data; the objective is to ensure consistency among semantically related data, but with different models (schemas) that might have been expressed in different languages like UML and OWL. For data sources with semantically related models, one simple consistency rule could be: *two objects (entities) with the same identity must have the same values stored for corresponding attributes; otherwise, they are not consistent with each other* (e.g., one data source claims that Bob has income of 10.000 an another lists earnings of 50.000). Figure 1 offers an overview of our approach to consistency testing of user data.

Given models `M1` and `M2` for two data sources, a consistency model is defined manually. The consistency model explicitly states constraints that must be fulfilled in order for instances of the two models to be consistent with each other. The consistency is defined at the “model level”; automatic consistency testing is done on the user data with help of the consistency model (which is instantiated automatically). The user data, depicted as `:M1` and `:M2`, must be instances of model `M1` and model `M2`, respectively. As can be seen from Figure 1, we need a *consistency modeling tool* and a *consistency testing tool*.

This main body of this paper offers a more detailed description of the consistency modeling and testing in a pure UML context, a more complex example would involve usage of both UML and OWL in defining the consistency model.

Some metamodeling tools are available in the literature [11, 12]. The consistency modeling and testing approach espoused in this paper are based on the results presented in [13].

This paper is structured as follows. Section 2 specifies what the framework should support together with preliminary design considerations. In Section 3, a consistency modeling technique is described in the context of UML; and the section also briefly

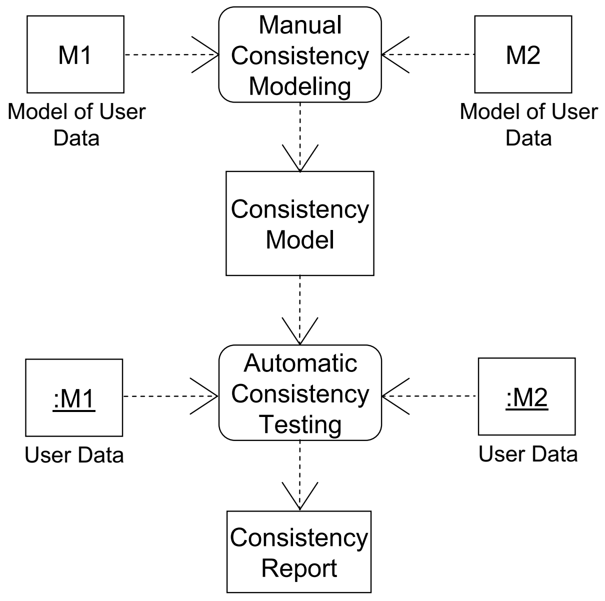

Figure 1: Consistency Modeling Overview

describes how consistency testing of user data can be performed. Finally, Section 4 offers a short summary and conclusions.

2 Data Integration Framework

In this section, we first describe the OMG metamodel architecture and how to represent user data and model. Then a non-exhaustive list of requirements to the framework is given, and finally initial design and implementation considerations are presented.

Use of the OMG Meta-Model Architecture

The OMG advocates a four-layer metamodel architecture [14] where MOF constitute the top level (level M3). The UML metamodel resides on the next highest level (level M2) and can be seen as an instance of MOF. When system's developers design a model using UML (level M1), the developers instantiate the metaclasses of the UML metamodel. In our context, only the small subset of the UML metaclasses that typically get instantiated on class diagrams are of interest. The run-time instances (user data) are found at the lowest level (level M0). The user-defined model has been instantiated to obtain these instances.

The OMG recommendations [6] state that the Ontology Definition Metamodel (ODM) should be an instance of MOF; this places the ODM at the same level as the UML metamodel—see Figure 2(a).

There is a semantic overlap between the UML metamodel and the ODM, but they are not subsets of one another, and a combination of the two might be worth investigating. Figure 2(b) illustrates a situation where the UML metamodel and the ODM are combined.

Our aim is to establish a framework where different types of languages and mixtures of languages can be investigated. The focus will be on languages that are defined by metamodels or, more specifically, MOF defined languages. If successfully implemented, the framework might be characterized as a framework for integration of MOF-based languages.

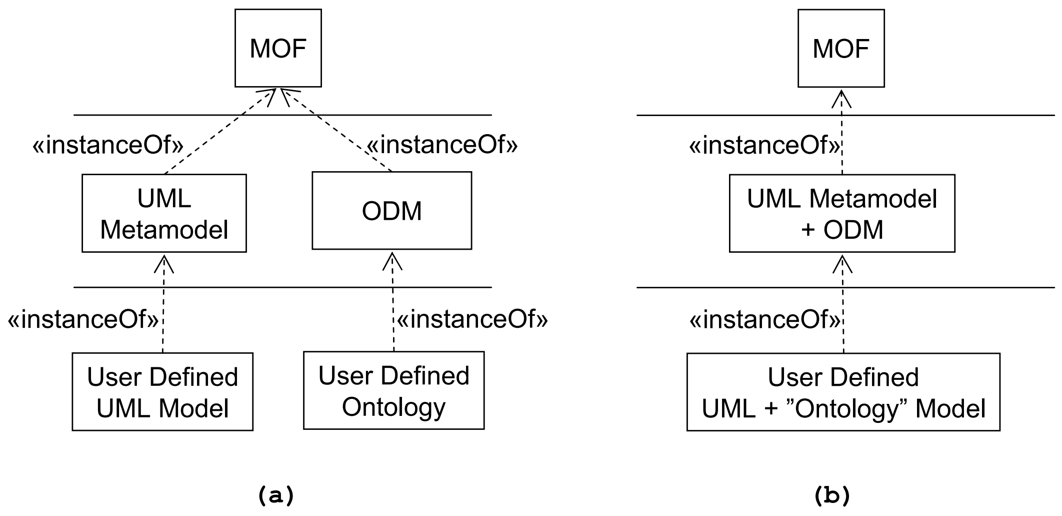

Figure 2: ODM Relative to the Metamodel Architecture

Representation of Model and Model Instance

UML 2.0 introduces the metaclass `InstanceSpecification`, which can be used to model an instance of another model element. An instance of `InstanceSpecification` can for example be used to illustrate an instance of a class (an object) or an instance of an association (a link between objects). As a concrete example, given a class `Person` (an instance of metaclass `Class`), `InstanceSpecification` can be instantiated to illustrate an instance of class `Person`; this is sometimes referred to as a snapshot (a run-time instance at a specific time) of the object. An `InstanceSpecification` will have a reference to the classifier that is the classifier of the represented instance. Consequently, it is possible to have a model (at level M1) that describes both a snapshot of user data and the corresponding metadata. When the user defines the consistency model, only metadata (models) matters; when the consistency testing is performed, both data and metadata must be present.

XML Metadata Interchange (XMI) [15] is an interchange format that can be used on models/data from all the four levels of the OMG metamodel architecture; XMI is a natural choice when it comes to storing and exchanging models and data.

The proposed framework should be able to visualize instances of models, e.g., visually pinpoint inconsistencies exposed by the consistency testing. The mentioned use of `InstanceSpecification` will make this possible.

What the Framework Should Support

The list that follows briefly states central functionality expected from the proposed framework.

- Support definition of MOF metamodels, e.g., guide the combination of two metamodels and resolve possible conflicts. The definition of MOF metamodels can also be done using tools such as UML2MOF [16], which transforms UML models (conforming to UML Profile for MOF) into MOF metamodels. Also, standard tools from IBM [17] have plug-ins that allow this.
- Offer users the possibility to load an MOF-specified metamodel.
- Offer users the possibility to instantiate the loaded metamodel. For example, if the loaded metamodel is the UML metamodel then the user is given the possibility to

make UML models (which is done by instantiating the loaded UML metamodel); or if the loaded metamodel is ODM, the user is given the possibility to define ontologies.

- Import and export of models based on XMI and the UML Diagram Interchange Specification [18].
- A UML model is typically represented as an instance of the UML metamodel, but an SQL schema is not. Transformation of an SQL schema to an UML model is rather straightforward . A transformation that is even more likely to be necessary concerns the user data that have to be represented as instances of metaclass `InstanceSpecification`.
- Specific features that support the modeling of consistency and automatic consistency testing. Section 3 offers more detail.

Implementation of the Framework

The implementation will be a set of components that can be assembled to form different tools. Figure 3 shows a set of interconnected components that together form a modeling tool. For example, if the component named *Metamodel Defined with MOF* is the ODM, the component named *MOF Based Modeling Tool* will give the user the ability to define an ontology that is made persistent with the help of the *Model Repository* component. Since modeling is to be done visually, the tool needs to know how to display the specific ontology elements. The *Concrete Syntax Definition* component will support this, although how this is to be achieved has yet to be investigated.

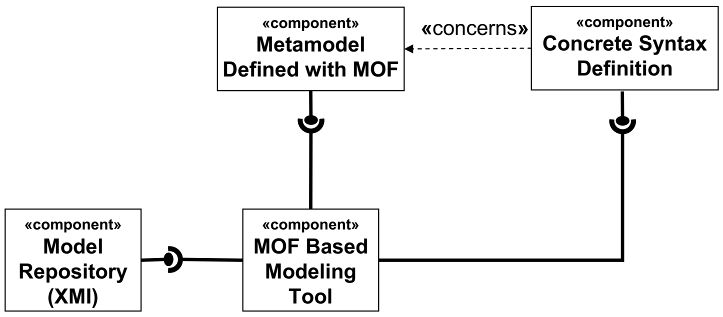
«component» Metamodel Defined with MOF
«concerns»
«component» Concrete Syntax Definition
«component» Model Repository (XMI)
«component» MOF Based Modeling Tool

Figure 3: A UML Component Diagram Showing a General MOF Based Modeling Tool

An important implication of the framework being based on the four-layer metamodel architecture is that formal Meta Object Facility descriptions of abstract syntaxes must be a understood; this understanding is incorporated (hard coded) into the the component named *MOF Based Modeling Tool*. Figure 3 only offers an abstract picture, and further investigation is necessary.

The Eclipse Platform [19, 20] is designed for building integrated development environments; it has a plug-in architecture that makes it suitable for extensions, and several useful plug-ins are already present. The UML tool Rational XDE from IBM [17] is built on the Eclipse Platform. Our framework could be built by making the right plug-ins for the Eclipse Platform. NetBeans IDE [21] is a similar framework and is also a candidate for use in implementing such frameworks.

3 Example Application of the Modeling Framework

Our consistency modeling and testing approach is presented in [13]. This section presents an example and propose a component architecture to achieve the desired functionality.

Figure 1 offers an overview of our approach. The consistency modeling is to be done with the Object Constraint Language (OCL) [14] and a selected subset of UML modeling elements:

- Association
- OCL constraint
- Association class
- Class
- Class attribute

The output of the *consistency modeling* is an *integration model* where the two legacy (in this context, "legacy" means "pre-existing") models (`M1` and `M2`) have been integrated and the desired consistency has been expressed explicitly. We term the part of the integration model that is not part of any legacy model the *consistency model*—see Figure 4.

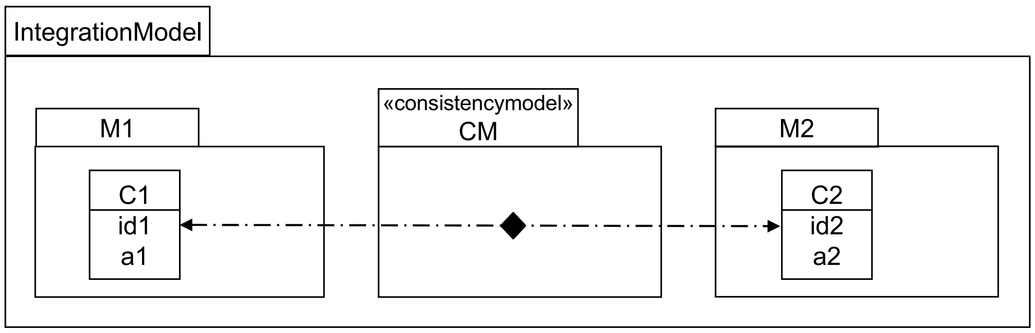

Figure 4: Integration Model Encompassing Legacy Models and a Consistency Model

We assume that the modeling activity is manual. Next, *consistency testing* is done automatically with the consistency model and legacy data (`:M1` and `:M2`) as inputs. Some processing of the user data might be necessary since they are to be represented with the help of metaclass `InstanceSpecification`. The output of the consistency testing activity is a report describing the consistency violations that were encountered.

Consistency Modeling Example

Figure 5 visualize three legacy models. Legacy model `M2` relates pictures to persons, legacy model M3 concerns observations done at different observation posts and legacy model `M1` concerns information about police investigations.

From the perspective of the police, the following question is of interest:: *Has the suspect lied about his whereabouts?* A suspect is exposed as lying if he claims to have been in one place, but has been observed at the same time from an observation post located elsewhere..

In Figure 6, a consistency model has been inserted. The dashed-dotted line between class `Person` and class `Suspect` represents an association—we term it a *consistency association*. This association is used for linking a suspect with a picture of the suspect.

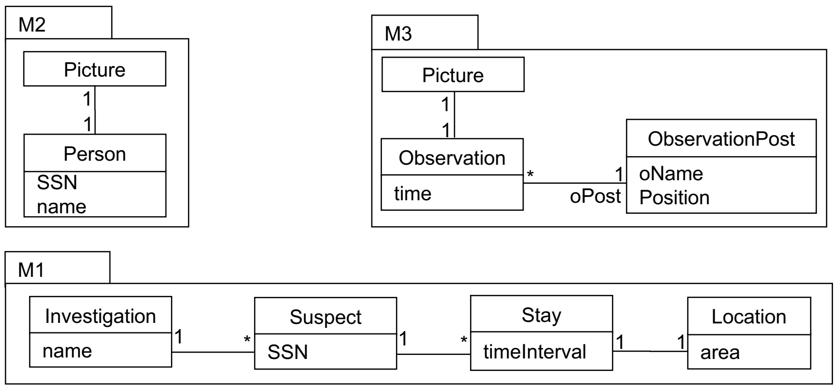

Figure 5: Three Legacy Models

The OCL constraint attached to the association: `person.SSN = Suspect.SSN` ensures that an object of class `Suspect` can only be linked to a correct object of type `Person` (the two objects must represent the same person).

Assume that classes `TimeSupport`, `AreaSupport`, and `PictureSupport` are part of the framework; the operations of these classes are class scoped and can be used inside OCL expressions.

The class `Stay` is used to record where a suspect claims to have been during a specific time interval. The consistency association between `Stay` and `Observation` is used to link a "stay" with observations done at the same time at different locations.

Class `ObservedAtOtherLocation` is a *consistency class* (stereotype `c-class`). The model prescribes that each `Stay` object must be linked to an `ObservedAtOtherLocation` object (multiplicity one-to-one). The constraint on the attribute `cNotExposed` prescribe the value `true` if no inconsistency has been exposed regarding where the suspect claims to have been and observations done; if there is an inconsistency then `cNotExposed` must be `false`. Attribute `cNotExposed` is an example of what we call a *consistency attribute*.

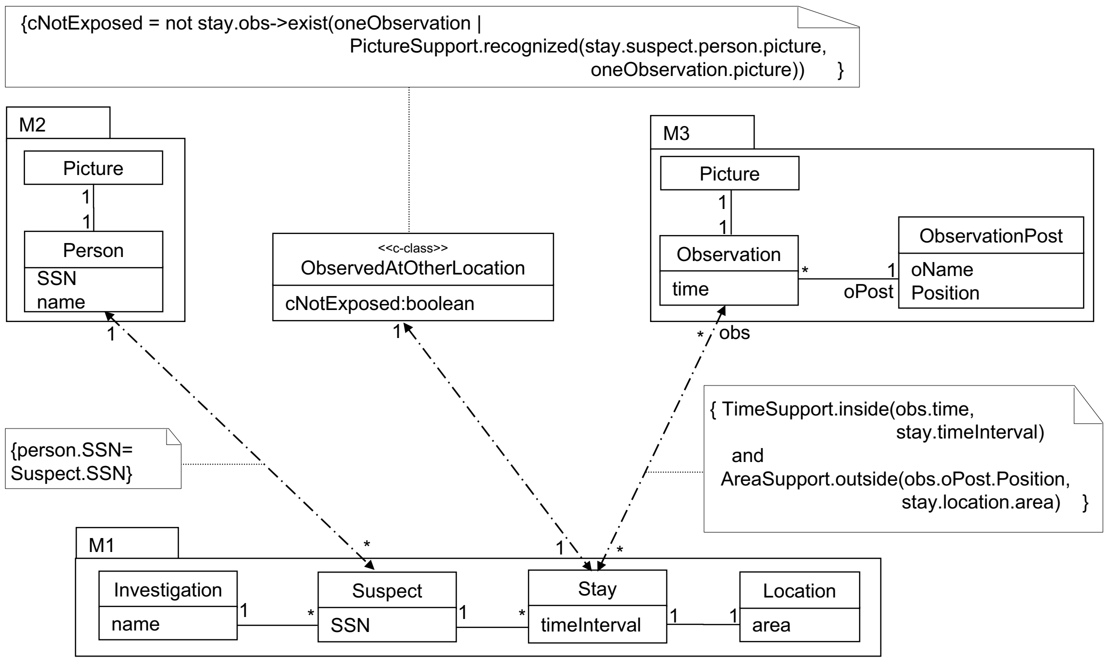

Figure 6: Consistency Model: Has the Suspect Been Somewhere Else?

The integration model can be made with an ordinary UML tool (except for the use of the dashed-dotted line, a stereotype `c-association` might be used instead).

At *consistency test time* an instance of the integration model will be instantiated. Instances of legacy models (the user data) are prefabricated and will be inserted as parts of the integration model instance. The test tool then automatically instantiates the consistency model. The consistency model can be seen as a declaration: instances of consistency model elements are in a sense derived from the legacy instances and the declaration. The constraint `person.SSN = Suspect.SSN` can function as a sort of production rule: for each pair of a `Person` object and a `Suspect` object, the constraint can be evaluated; and if it is fulfilled, a link can automatically be created. The rest of the consistency model can be instantiated in the same way.

A closer look at the constraint on `cNotExposed` reveals navigations through all the consistency associations. As a consequence, instantiation of this attribute must be performed last. The constraints, the consistency associations, and the attributes of the consistency classes must not be dependent on each other in a cyclic way—if they are, it might not be possible to do the automatic instantiation. The order of instantiation can be decided by building a dependency graph, see reference [13] for details.

The attributes of the consistency classes are used when the consistency report is generated, e.g. if `cNotExposed` equals `false` then there is a consistency violation.

Consistency Test Tool

A preliminary design of the consistency test tool is presented in Figure 7. The component named *UML Model and Data Repository* provides the integration model and the legacy data.

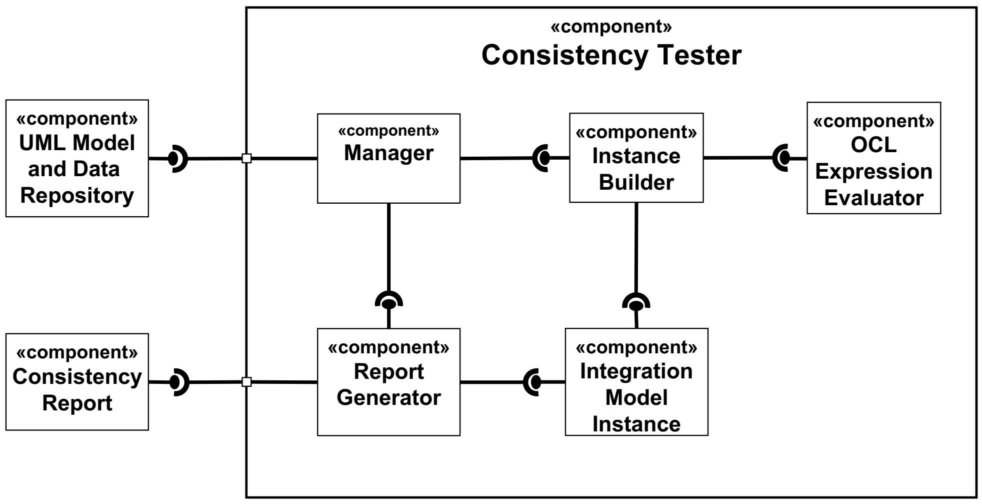

Figure 7: UML Component Diagram Showing the Consistency Tester

Looking inside the *Consistency Tester* component, we find the subcomponent *Instance Builder* that builds the consistency model instance which is represented by subcomponent *Integration Model Instance*. To build the instance, evaluation of OCL-expressions are necessary; this is done with the help of the *OCL Expression Evaluator* subcomponent.

The subcomponent *Report Generator* uses the subcomponent *Integration Model Instance* and produces a consistency report.

As mentioned above the integration model can be made with an ordinary UML tool, but an ordinary UML tool will allow cyclic references; a more sophisticated tool could prevent this. Obviously the framework presented above (section 2) is a candidate for making such a tool. In [13] a metamodel for consistency models is proposed. This metamodel could be input to the general MOF-based modeling tool presented above.

4 Summary and Conclusions

There is momentum in industry and academia towards the integration of UML and knowledge representation languages. A recent request for proposals issued by the Object Management Group is a clear indication of this (e.g. [6] and [22]). We have started the development of a tool (or framework) that will support such an integration: if successfully developed, the tool can be used to define diagrams that simultaneously incorporate both UML and "ontology features." The tool is meant as a MOF metamodeling tool, meaning that a MOF metamodel can be input as a specification. The tool then allows the user to instantiate the specified metamodel.

This paper has demonstrated how the full power of OCL as a declarative language can come to play in a setting where the consistency of partially overlapping data sources is to be specified and checked. The modeling technique proposed in the paper for consistency specification is based on standard OCL and a small subset of UML's graphical modeling notation. In future work, the reasoning possibility typically offered by knowledge representation languages will be included.

The proposed tool may be seen as representing a step in the direction towards the creation of a language that possesses the full power of UML and knowledge representation languages. The described consistency modeling and testing, which is an application of the framework will function as a practical demonstration.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[2] J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., 2000.

[3] D. Brickley and R. V. Guha. Resource Description Framework Schema Specification 1.0. Technical Report, W3C Consortium, http://www.w3.org/TR/2000/CR-rdf-schema-20000327, March 2000.

[4] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web. In *18th Nat. Conf. on Artificial Intelligence*, 2002.

[5] D. Brickley and R. V. Guha. OWL Web Ontology Language: Overview. Technical Report, W3C Consortium, W3C Working Draft: http://www.w3.org/TR/owl-features/, March 2003.

[6] OMG Editor. *Ontology Definition Metamod.RFP, OMG Document:ad/2003-03-40*. http://www.omg.org/techprocess/meetings/schedule/Ontology_Definition_Metamod.RFP.html, 2003.

[7] P. Kogut, S. Cranefield, L. Hart M. Dutra, K. Baclawski, M. Kokar, and J. Smith. Extending UML to Support Ontology Engineering for the Semantic Web. volume 2185, pp 342–360, 2001.

[8] OMG Editor. *Meta Object Facility (MOF) 2.0 Core Proposal, OMG Document:ad/2003-04-07*, 2003.

[9] OMG Editor. *OMG Unified Modeling Language Specification, Version 2.0*. OMG Document. OMG, http://www.omg.org, 2003.

[10] K. Baclawski, M. K. Kokar, P. A. Kogut, L. Hart, J. Smith, W. S. Holmes III, J. Letkowski, and M. L. Aronson. UML for Ontology Development. *Knowledge Engineering Review*, 2001.

[11] Honeywell Inc. *DOME (the DOmain Modeling Environment)*. http://www.htc.honeywell.com/dome/description.htm Accessed June 2003.

[12] MetaCase Consulting. *MetaEdit+*. http://www.metacase.com, Accessed June 2003.

[13] J. P. Nytun and C. S. Jensen. Modeling and Testing Legacy Data Consistency Requirements. In *UML 2003*, October 2003.

[14] OMG Editor. *OMG Unified Modeling Language Specification, Version 1.5*. OMG Document. OMG, http://www.omg.org, March 2003.

[15] OMG Editor. *XML Metadata Interchange (XMI) Specification v1.2*. OMG Document. OMG, http://www.omg.org, January 2002.

[16] netBeans.org. *UML2MOF Tool*. http://mdr.netbeans.org/uml2mof, Accessed June 2003.

[17] Rational Software Corporation. http://www.rational.com, Accessed June 2003.

[18] M. Boger, M. Jeckle, S. Müller, and J. Fransson. Diagram Interchange for UML. In *UML 2002*, pp. 398–367, October 2002.

[19] eclipse.org. *Eclipse IDE*. http://www.eclipse.org, Accessed June 2003.

[20] S. Shavor, J. D'Anjou, S. Fairbrother, D. Kehn, J. Kellerman, and P. McCarthy. *The Java Developer's Guide to Eclipse*. Addison-Wesley, 2003.

[21] netBeans.org. *NetBeans IDE*. http://mdr.netbeans.org, Accessed June 2003.

[22] OMG Editor. *Production Rule Representation.RFP, OMG Document:br/2003-09-03,* 2003.