# Ontology-Based Knowledge Representation for Self-governing Systems

Elyes Lehtihet[1,3], John Strassner[2], Nazim Agoulmine[3], and Mícheál Ó Foghlú[1]

[1] Telecommunications Software & Systems Group -
Waterford Institute of Technology
Waterford, Ireland
{elehtihet, mofoghlu}@tssg.org
[2] Autonomic Research Lab - Motorola Labs,
Schaumburg, IL 60010, USA
John.Strassner@motorola.com
[3] Networks and Multimedia Systems Group - University of Evry-Val d'Essonne
Evry Courcouronnes, France
Nazim.Agoulmine@iup.univ-evry.fr

**Abstract.** Self-governing systems need a reliable set of semantics and a formal theoretic model in order to facilitate automated reasoning. We present an ontology-based knowledge representation that will use data from information models while preserving the semantics and the taxonomy of existing systems. This will facilitate the decomposition and validation of high level goals by autonomous, self-governing components. Our solution reuses principles and standards from the Semantic Web and the OMG to precisely describe the managed entities and the shared objectives that these entities are trying to achieve by autonomously correlating their behavior. We describe how we created UML2, MOF, OCL and QVT ontologies, and we give a case study using the NGOSS Shared Information and Data model. We also set the requirements for integrating existing information models and domain ontologies into a unique knowledge base.

## 1 Introduction

The representation of knowledge at the autonomic manager level is a critical issue for designing and deploying self-governing systems. We see an autonomic manager as a set of software agents that use an expressive and dynamically updateable knowledge base to represent the relationships between managed entities, the system configuration, the objectives of the administrators (policies) and the explicit semantics of the goals of the system. For autonomic networks, this flexible knowledge base enables the system to dynamically adjust to the changing demands of its users, as well as changing environmental conditions.

Self-governing systems can be described as very reactive systems, with a precise modeling structure, data description and behavior. The basis for structuring is hierarchical decomposition and type hierarchies. Data description is based on data types of values and objects. The basis of behavior description is extended finite state machines communicating by messages [1].

A knowledge base requires an expressive, unique representation. Arguably, this is best realized using a knowledge representation language with well defined semantics and a practicable inference algorithm for reasoning on a system specification. A system specification, in a broad sense, is the specification of both the behavior and a set of general parameters of the system [1].

The integration of the structural and the dynamic aspects of a self-governing system is a difficult task that is even harder to achieve because of: 1) the complexity of describing and refining a high level goal into lower level objectives that can be enforced by autonomous system entities, and 2) the lack of interoperability between existing modeling standards. Interoperability is a major issue for system integrators and tools vendor; several initiatives are trying to solve the problem by unifying knowledge representation without loosing the associated semantics of the data. Until this is accomplished, it will be impractical to effectively share models among heterogeneous (competing) modeling tools.

In this paper, we present an approach for unifying the representation of the information needed by an autonomic manager, and preserving the inherent semantics of a domain — this is a crucial step toward automating the reasoning process and integrating heterogeneous knowledge bases.

### 1.1 Using Software Engineering Principles to Specify Self-governing Systems

An autonomic manager is a software agent, which must be developed by applying technologies and best practices from computer science, applications domains and other fields. Due to the complexity of describing a high level goal and its refinement into low level objectives, we argue that an agile (adaptive) development framework, as opposite to a plan-driven (predictive) framework, is the most appropriate methodology for designing, deploying and testing self-governing systems. The Agile methodology is quite simple to understand: *Instead of creating extensive models before writing source code, you start by developing and testing agile models with a concise and a precise kernel.*

Another approach for defining IT system specification is the OMG Model Driven Architecture (MDA). The aim of MDA is not to replace the existing system with something completely different; the objective is to make that system more efficient by incrementally automating the parts of it that can easily be automated, so it will accelerate the integration of new technologies and automate their integration into the existing solution.

Many would argue that principles of Agile Modeling and OMG Model Driven Architecture are disjointed (not compatible), but we see the two methodologies as complementary: First, to accelerate the (agile) development, the behavior of a system must be derived from the specification. It is only this way that we will ensure that the business goals (desired behavior) meet the implementation (effective behavior). Second, as software architectures grow in size and complexity, the need to subsequently incorporate software models into the system development automation stream grows even faster.

In the white paper comparing the OMG-MDA and the TMForum Next Generation Operation Support System (NGOSS) [2], the authors argue that: *"While following an approach similar to MDA, NGOSS has chosen to focus on building a framework for identifying and specifying well-defined business and system views on a modeled OSS/BSS solution"*. NGOSS principles and standards will be discussed more in detail in section 4 and 5. Note that our work extends the work of NGOSS.

The OMG Unified Modeling Language (UML) is the software industry's dominant modeling language. The current standard is UML 2.0., a major rewrite on the previous 1.5 version. This 1.5 version, used by the NGOSS Shared Information and Data model (SID), will continue to be the official current version until all four components of UML2 (Infrastructure, Superstructure, Diagram Interchange and Object Constraint Language) are completed and ratified.

In section 8.2 of the OMG Ontology Definition Metamodel (ODM) specification [3], it is mentioned that: *"The lack of reliable set semantics and model theory for UML prevents the use of automated reasoners on UML models. Such a capability is important to applying Model Driven Architecture to systems integration ... UML lacks a formal model theoretic semantics, OCL also has neither a formal model theory nor a formal proof theory, and thus cannot be used for automated reasoning (today)"*.

The same arguments can be used against the OMG Meta Object Facility (MOF), and since every modeling language used in MDA "must be" described in terms of the MOF language, the OMG Vision is missing a formal, explicit specification of concepts and relationships for its modeling languages (UML2, MOF, OCL and QVT). Thus, any information model that "only" relies on UML would not fulfill the requirements of a decidable knowledge base and will prevent the use of automated reasoning.

Reasoning on a precise and computer-processable semantic is the ultimate objective of the Semantic Web vision. The essential principles, standards and technologies are discussed in the following section.

### 1.2 Semantic Web Technologies

While trying to ensure the long term growth of the web, the World Wide Web Consortium (W3C) issued the Web Ontology Language (OWL) — a markup language for publishing and sharing data using ontologies on the Internet. Ontologies are agreements about shared conceptualization [4], and hence a basis for information exchange by putting documents with machine-readable meaning (semantics) on the Web. OWL permits varying degrees of reasoning depending on the expressivity of the Description Logic subset that is used. Description Logics, sometimes called terminological systems or concept languages, are a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way [5].

OWL$\mathcal{DL}$ is based on the description logic category known as $\mathcal{SHOIN}(\mathcal{D})$. Its subset OWL Lite is based on the less expressive logic category $\mathcal{SHIF}(\mathcal{D})$. OWL

Full is the most expressive level but, because of that, *can* lead to infinite loops — not recommended for automated reasoning. The Semantic Web Rule Language (SWRL) extends the set of OWL axioms in order to include conditional rules (Horn clauses).

As noted by Ushold and Menzel in [4], the strength of the W3C standard for representing ontologies, in addition to its soundness and unique implementation, is the ability to express logical equivalence and other relationships between concepts, properties and individuals in different ontologies. One main weakness is the lack of support for procedural functions (e.g. arithmetic, string manipulation/comparison) that are, in our opinion too, essential for mapping between real-world ontologies.

The following section will give a summary of existing approaches for representing knowledge for Autonomic systems. In section 3, we describe the methodology that we have used to create a precise modeling language for information models. Then, we will demonstrate how our solution can be applied to the TMF NGOSS set of principles. Finally, we will discuss the results of our experiments and describe the future orientation of our research.

## 2 Related Works

As stated in [6], Information models alone are not enough to capture the semantic and behavior of managed network entities. The ideal solution is to use an ontology language to precisely formalize a domain problem. In this section, we will give a short overview on existing approaches and their use of information modeling.

### 2.1 Autonomic Network Management

In 2004, IBM issued a toolkit that includes components, tools and scenarios for designing and deploying Self-managing systems. In [7], the importance of ontologies in the design and the implementation of autonomic computing systems is described: *without an explicit meaning, the resolution of a problem is not possible.* As shown in Figure 1, IBM uses the DMTF Common Information Model (CIM) as a reference model to infer properties about distributed systems. The system behavior is expressed using the Simplified Policy Language (SPL). Common Base Event (CBE) is the IBM standard for exchanging messages between autonomic management engines (implementation of the autonomic-manager). The CBE model provides a basis for sounder problem determination and is a cornerstone of automatic computing system management. This profusion of formats creates a semantic gap between the specification of the system and its behavior; since they are not expressed in the same language and do not share the same semantic, how can they be integrated in a unique knowledge base? Furthermore, as noted in [6], the DMTF, IETF and ITU do not produce UML compliant models, and thus cannot reuse off-the-shelf UML tools to represent their concepts.

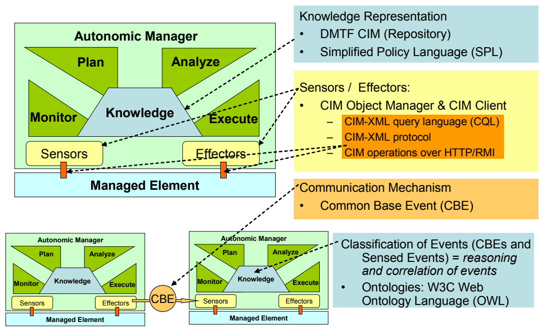
Autonomic Manager
Plan
Analyze
Monitor
Knowledge
Execute
Sensors
Effectors
Managed Element
Knowledge Representation
• DMTF CIM (Repository)
• Simplified Policy Language (SPL)
Sensors / Effectors:
• CIM Object Manager & CIM Client
– CIM-XML query language (CQL)
– CIM-XML protocol
– CIM operations over HTTP/RMI
Communication Mechanism
• Common Base Event (CBE)
CBE
Classification of Events (CBEs and Sensed Events) = reasoning and correlation of events
• Ontologies: W3C Web Ontology Language (OWL)

**Fig. 1.** IBM Autonomic Computing Vision

As shown in Figure 2, the advantage of using UML notations is to capture the specification of the system and its behavior. We believe that the implementation of Self-governing systems will rely on a specification of Executable Models [8] — with a formal model theoretic and a precise semantic, necessary to implement a reliable model compiler (Reasoner). This model compiler will allow automated reasoning and help in discovering hidden inconsistencies in the system specification.

Guerrero et al., in [9], proposed the utilization of OWL+SWRL for the definition of the management behavior. SWRL would replace the conventional policy language while the CIM-to-OWL mapping will enable system properties to be inferred. However, we think that this approach has two important limitations:

1. The fundamental difference between Object Oriented (OO) models and Ontologies is the representation of semantics: Ontologies use a declarative approach (Constraints, Axioms and Rules), but OO models only represent imperative semantics (operations). Therefore, it is not possible to map OO models into OWL without loosing semantics (e.g., for the CIM, as used in the IBM toolkit, semantics attached to the CIM Methods and Qualifiers are lost).
2. The DMTF uses their own proprietary Managed Object Format (DMTF-MOF), which is not compatible with the OMG-MOF. Hence, CIM Models will *not* produce valid UML models and, as discussed in section 1.1, it will not be possible to reuse any software development methodology (e.g., Agile or MDA) for the design, deployment and testing of a self-governing system.

In [10], López de Vergara proposed to refine and extend the CIM Metaschema by using the Object Constraint Language (OCL); however, this necessary formalization was not incorporated into the DMTF specification. Recently, the DMTF realized the advantage of aligning its model with the OMG standards. This key work will enable the use of off-the-shelf UML tools for CIM development. However, the first draft will not be available before the $3^{rd}$ quarter of 2006 [11]. Until then, we will not consider any CIM-based approach as a possible solution for representing knowledge for self-governing systems.

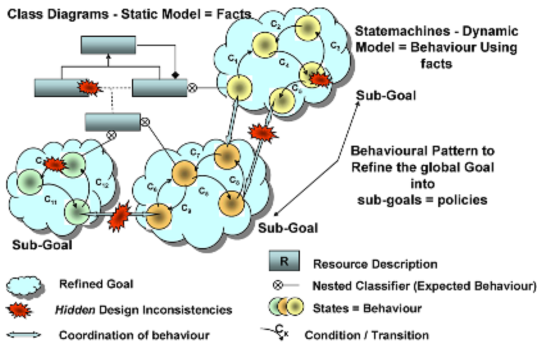
Class Diagrams - Static Model = Facts
Statemachines - Dynamic Model = Behaviour Using facts
Sub-Goal
Behavioural Pattern to Refine the global Goal into sub-goals = policies
Sub-Goal
Sub-Goal
Refined Goal
Hidden Design Inconsistencies
Coordination of behaviour
Resource Description
Nested Classifier (Expected Behaviour)
States = Behaviour
Condition / Transition

**Fig. 2.** Systems specification using UML

## 2.2 Ontologies, Information Models and Constraint/Rule Languages

Ontologies and Information models (Object Oriented Languages) have very similar approaches for the declaration of static structures, namely classes (concepts), class hierarchies (using inheritance), attributes, relationships, and instances [12]. However, an ontology only describes concepts and their inter-relationships; it does not provide support for behavioral features (e.g., operations, parameters, and state machines). Therefore, trying to represent a UML-based information model in OWL will lead to major inconsistencies and loss of valuable semantics.

As previously noted, SWRL extends OWL with Horn-like Clauses. SWRL provides formal semantics and thus allows computation, unlike the OMG-OCL which does not provide a formal proof theory (see section 1.1). We are working on a mechanism to map existing OCL Expressions into SWRL axioms, but this is beyond the scope of this paper and hence, will be developed in future work.

An advantage of using an ontology, in addition to the computation guarantee, is the uniqueness of the representation, since the implementation of the W3C specification guarantees interoperability between different ontology tools and repositories. On the other hand, for all OMG specifications, the informal definition of the concrete syntax is not given in the semantics document, but in the notation guide. There is no mapping between the concrete syntax and the abstract syntax. This lead to an implementation problem: there is no way to check that the output of a tool conforms to the language specification (UML, MOF, OCL or QVT). This is because the OMG only produces the specifications — documents that precisely describe what something should do, and how it should act. The implementations of the specifications (UML Modeling tools, Transformation Engine, Model Compiler/Checker) are not, and will never be, produced by the OMG [13]. Thus, the lack of constraints in the specification of the language (metamodel) and the heterogeneous implementations create informal (not precise) models.

We investigated the work of Cranefield [14] and Knublauch [12], who specified a mapping from UML models to RDF and OWL, respectively. However, these approaches are not appropriate for capturing the semantics attached to behavioral diagrams, as well as some semantic aspects of the class diagrams.

## 3 Ontology Based Metamodel for UML2, MOF, OCL and QVT

Our approach is to represent the UML, MOF, OCL and QVT *metamodels* into OWL models. Therefore, any UML *model* can be checked against the precise specification of its metamodel and thus, all the concepts from our system model (structure and behavior) will be instantiated in a unique format that will surely comply to the OMG specification previously captured by an ontology.

In this way, we have built a knowledge base using an expressive and unique language (OWL) with well defined semantics (Description Logic) and practicable inference algorithms ($\mathcal{DL}$ Reasoners) for reasoning on a system specification (UML structural and behavioral models).

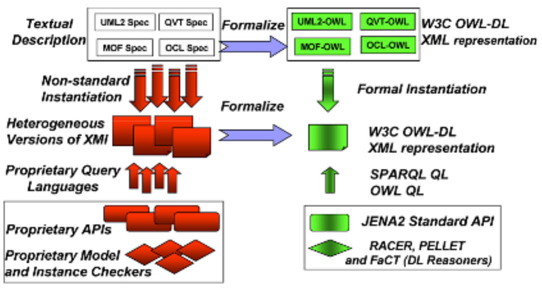
Textual Description
UML2 Spec
QVT Spec
MOF Spec
OCL Spec
Formalize
UML2-OWL
QVT-OWL
MOF-OWL
OCL-OWL
W3C OWL-DL XML representation
Non-standard Instantiation
Formal Instantiation
Formalize
Heterogeneous Versions of XMI
W3C OWL-DL XML representation
Proprietary Query Languages
SPARQL QL OWL QL
Proprietary APIs
Proprietary Model and Instance Checkers
JENA2 Standard API
RACER, PELLET and FaCT (DL Reasoners)

**Fig. 3.** Ontology-based knowledge specification of reactive systems

### 3.1 Transformation Principles

We have implemented transformations from the main OMG specifications (MOF, UML2, OCL and QVT) to W3C-OWL. UML2, MOF1.4 and MOF-QVT are available in Rational Rose format on the OMG website. UML and MOF reuse the InfrastructureLibrary; MOF and OCL are decomposed into two main packages: Essential and Complete, while QVT extends EssentialMOF and EssentialOCL. However, we were not able to find any reference to CompleteOCL. As a result, every OMG sublanguage is a distinct package that has dependencies (import/merge) with other packages.

In Figure 4, we show an example of dependencies between UML top level packages and the UML subset that the SID Business View employs. In this paper, we outlined the transformation rules and detail the implemented parser for the XMI output of the Rational Rose Unisys Add-In. Every package was transformed into a separate OWL ontology (file).

For UML2, we generated 82 ontologies that import each others depending on their dependencies (import/merge) and the inter-references between elements in

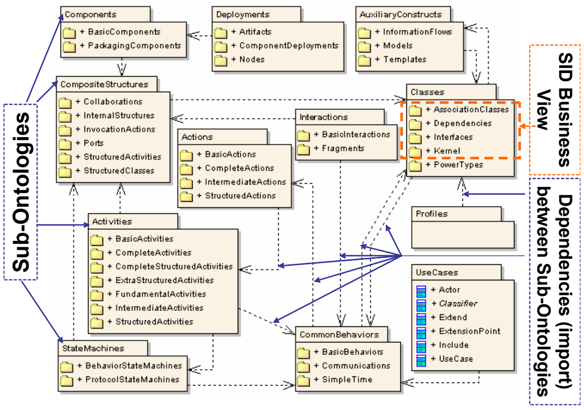

**Fig. 4.** UML2 top level packages and the SID Business View Subset

separate packages. Every class, owned by a package, has a unique name and thus map to an `owl:Class`. For the range of the attributes (String, Boolean, Integer and UnlimitedNatural), we did not want to use the predefined xsd datatypes (used by Protégé), the reason is that they are not supported by the current version of reasoners — this limitation should be addressed in the next version of OWL. Therefore, all the data types were represented as an `owl:Class`; the attributes were all mapped to `owl:ObjectProperty` with the following pattern : `<ClassName>.<AttributeName>`. The same mechanism was used for the AssociationEnds, where the role name of every *navigable* association end was mapped to an object property of the source class. We treated the cardinalities of the object properties, respectively `lower` and `upper`, as follow:

- [1..1] ⇒ `FunctionalProperty`;
- [0..n] ⇒ Default Cardinality;
- [x..n] ⇒ `minCardinality(x)`;
- [0..x] ⇒ `maxCardinality(x)`;
- [x..x] where $x \neq 1$ ⇒ `cardinality(x)`;
- [x..y] where $x > 0 \wedge x \neq y \wedge y \neq n$ ⇒ `minCardinality(x)` ∧ `maxCardinality(y)`.

The Enumeration Classes (AggregationKind, VisibilityKind, etc.) were mapped to an `owl:Class`. Their possible values, previously represented as attributes, were mapped to OWL instances of the owning class and explicitly made disjoint. Example: UML VisibilityKind enumerated attributes {package protected private public} were made disjoint by applying an `owl:allDifferents` construct.

We created our specific URI to identify the ontologies, which is also their online repository. The results of the transformations can be found at : http://www.tssg.org/public/ontologies/*org*/*spec*/*year*/*PackageName*.owl; where:

- UML2 : BaseUri + omg/uml/2004/UML2-Super-MDL-041007.owl
- QVT : BaseUri + omg/qvt/2005/QVT.owl
- MOF : BaseUri + omg/mof/2004/MOF.owl

The ontologies (which import their dependent ontologies), can be loaded in Protégé. When performing the Classification Hierarchy, the Reasoners will remove the unnecessary superclasses. For example, *Transition* and *RedefinableElement* are subclasses of *NamedElement*; and *Transition* is also a subclass of *RedefinableElement*. Therefore, the inheritance between *Transition* and *NamedElement* is superficial and thus can be removed without altering the consistency of the model — This should be considered as an optimization of the language.

## 3.2 Limitation of the Transformation

As noted in section 2.2, ontologies do not support behavioral features; thus the operations (and parameters) present in the UML2 metamodel were mapped into annotation properties (`rdfs:Comment`) of the owning class. UML2 encloses 107 private operations. All the operations specify an OCL expression as Text — not XML constructs. The automated mapping of OCL to SWRL is not supported by our tool as yet, a solution to this will be proposed in our future work.

OWL does not apply the Unique Name Assumption (UNA) by default: every concept is not, by default, necessarily distinct from the others. In our case, this implies that all the constructs of the OMG metamodel are not necessarily distinct. This in turn implies that an `Association` is not by default different from a `Class`. To precisely describe the metamodel, we implemented an automatic generation of disjointment between classes in the same package; however, we had to abandon this solution because it created too many inconsistencies related to multiple inheritances. Example: an `AssociationClass` is a `Class` and an `Association` at the same time, so if a `Class` is semantically disjoint from an `Association`, it implies that an `AssociationClass` is disjoint with itself. However, this can be solved in the instantiation of the model by explicitly creating an `owl:allDifferents` construct between instances or specifying an `owl:disjointWith` between subclasses.

There is another issue with the cardinality restrictions. Because ontologies use the Open World Assumption (OWA) for reasoning, it means that what is given to the reasoner is not necessarily complete. The reasoner will only generate an error when there are more than the allowed `owl:maxCardinality` (or `owl:cardinality`) instances associated with an object property. However, if there are fewer instances than the `owl:minCardinality` restriction then the reasoner will assume that it could be defined elsewhere and therefore infer that the ontology is consistent. The solution is to implement a pre-compiler to check the cardinalities and corresponding instances before using the reasoner to check to overall consistency of the model.

Another limitation concerns Package naming. There are only two cases where the name of the Package is duplicated in the specification. An algorithm can look for such conflicts and change the name of the package (sub-ontology); we used the convention `<owningNamespace>.<PackageName>`. Example: `InfrastructureLibrary.`*Profiles* and `UML.`*Profiles*.

## 4 Mapping the TMF NGOSS SID to OWL

Once the UML2 ontology was defined, we could check the consistency of any UML2 model (structure and behavior) against the precise specification of the language. In this section we will describe how we applied a mapping from the SID (UML Model) to an Ontology, without loosing the semantics of the modeling language (behavioral features).

As noted in section 1.1, the SID uses UML version 1.5. After a review of the UML specifications, we noticed that the main differences between the two versions concern the behavioral aspect of the language. For the object structure, with the exception of the NestedClassifier pattern that is not used in the actual version of the SID Business View, there is no difference between UML 1.5 and 2.0. Therefore, the UML subset used by the SID is fully compatible with the UML2 ontology since it does not use any behavioral diagram. The transformation rules are described in the following paragraphs.

The SID is the *"lingua franca"* for all TMF work. It defines a common set of concepts, in the form of an object-oriented information model, that all other TMF programs can use. The mapping from the SID to OWL, having a UML2 ontology, is quite straightforward. Every language construct used by the model: *Class*, *Association*, *Property*, *Operation*, *Attribute*, *Stereotype*, *DataType*, *AssociationClass*, *Dependency*, etc. has a direct equivalence in the ontology model. The interrelation between entities, *ownedAttributes*, *ownedOperations*, *ownedParameters*, *stereotypes*, *datatypes*, etc. is inherently present and constrained (cardinalities, domain and range) in the imported UML2 sub-ontologies: *Kernel*, *AssociationClasses*, *Dependencies* and *PrimitivesType*.

There are two different ways of representing a UML model: by instantiating the element of the model from their specification in the ontology (i.e., represent the model as a set of individuals), or by subclassing every element of the model and expressively constraining the values of its object properties (`owl:allValuesFrom`). The choice will not make any semantic difference for the reasoner. However, we preferred the second approach, as it offers a better visualization with the Protégé editor.

If the approach for representing the model is to subclass and constrain the corresponding language constructs, then all the `owl:Class` elements in a sub-ontology must be disjointed. But, if the representation mechanism is to create an instance of the model, then all the individuals must be distinguished by adding an `owl:allDifferents` axiom.

The mapping principles are also quite simple; every SID package will be represented in a separate file and imports the required UML2 sub-ontologies, in addition to the other SID packages referenced by its elements (datatype, constrainedElement, etc.). Every sub-ontology extend the imported *Kernel:Package* to create an `owl:Class` with the name of the package as an ID. The same mechanism is applied to every language construct. The algorithm for naming the sub-ontologies is similar to the one described in section 3.1.

One minor limitation is the representation of the *AssociationEnd* and *Constraint*. We implied that every element in the model must be properly named, because it is more readable than the *xmi:id* that distinctively identify the elements. However, it is not the case with all the SID elements, so we had to manually update the model.

## 5 Discussion and Future Work

Our approach can be used for any UML2-based model, and the UML2 ontology could be refined/extended to capture additional artifacts not present in the actual specification. Two important aspects of the NGOSS Methdology that can benefit from our solution are the NGOSS Contracts and the NGOSS Metamodel specifications.

NGOSS Contracts extend the *Design by Contract* paradigm to ensure that all information that is exchanged between components is done so in a consistent way. All NGOSS Contracts have a view-specific portion (NGOSS contains views that are used to represent the needs of different constituencies, such as business analysts vs. programmers and architects). The view specific model part contains various types of models (UML and others) tailored to support the specific view of the contract, i.e. Business, System, Implementation and Deployment views. All these views need to be integrated in order to provide a coherent mapping between NGOSS views: reasoning on Contracts models. Therefore, our solution could be fully reused to represent the model part of the Contracts expressed at different level; the Reasoning mechanism would allow their automatic validation.

The NGOSS Metamodel (TMF053D) extends the UML metamodel in order to introduce specific concepts, building blocks and artifacts that are required to represent telecom needs. Thus, TMF053D is a necessary reference document to support the creation of NGOSS-based models of software system solutions. These models will capture specific aspect of the telecom world and will need to be stored in a unique and precise knowledge base. Now, for the same reason described in section 2.2, a solution based on the extension of the UML2 Ontology with the NGOSS artifacts will ensure the different NGOSS stakeholders that the semantic of their systems will be fully captured in the model and they will have a unique, open repository for all their views of the system based on the "homogeneously" implemented W3C OWL specification.

## 6 Conclusion

Autonomic systems require knowledge from different sources to be represented in a common way. While conflicting attribute and datatype definitions present problems, semantic dissonance is a far more difficult problem to solve — one that requires an extensible, common representation of knowledge that does not lose its associated semantics. This paper has introduced an ontology-based knowledge representation to solve this problem. We have used the algorithms described in this paper to construct an OWL representation of the TMF NGOSS SID, which we are using in other autonomic computing works. The OWL mapping provides a *machine-readable* representation of the SID managed entities and concepts

— this enables semantics to be properly captured and associated with model elements.

Future work will include mapping OCL into SWRL axioms and building a reasoner that can use the OWL SID mapping (which includes NGOSS Contracts) to build a reliable reasoner for contract-based interactions and workflows. These two work items will then be used to continue our research in semantics for autonomic computing.

## Acknowledgement

This research has been funded by SFI *"Autonomic Management of Communications Networks and Services"* PI Cluster Award: 04/IN3/I404C.

## References

1. International Telecommunication Union (ITU-T): Specification and Description Language (SDL), Recommendation Z.100, August 2002.
2. Strassner J., Fleck J., Huang J., Faurer C., Richardson T.: TMF White Paper on NGOSS and MDA, TMForum, April 2004.
3. Object Management Group (OMG): Ontology Definition Metamodel, Fourth Revised Submission, November 2005.
4. Ushold M., Menzel C.: Achieving Semantic Interoperability & Integration Using RDF and OWL, W3C Draft, January 2006.
5. F. Baader, D. Calvanese, D. L. McGuiness, D. Nardi, P. F. Patel-Schneider: The Description Logic Handbook: Theory, Implementation, Applications. Cambridge University Press, Cambridge, UK, 2003
6. Strassner J., Agoulmine N., Lehtihet E.: FOCALE A Novel Autonomic Networking Architecture, in Latin American Autonomic Computing Symposium (LAACS), July 18-19, 2006, Campo Grande, MS, Brazil.
7. Stojanovic L., Schneider J., Maedche A., Libischer S., Studer R., Lumpp Th., Abecker A., Breiter G., Dinger J.: The role of ontologies in autonomic computing, Published in IBM Systems Journal, Volume 43, Issue 3, 2004.
8. Mellor S. J., Balcer M. J.: Executable UML: A Foundation for Model Driven Architecture, Addison-Wesley Longman Publishing Co., Inc., 2002.
9. Guerrero A., Villagrá V. A, López de Vergara J. E, Berrocal J.: Ontology-Based Integration of Management Behaviour and Information Definitions Using SWRL and OWL. DSOM 2005, October 24-26, 2005, Barcelona, Spain: pp 12–23.
10. López de Vergara J. E, Villagrá V. A, Berrocal J.: On the formalization of the Common Information Model metaschema. DSOM 2005, October 24-26, 2005, Barcelona, Spain, pp 24-26.
11. DMTF Newsletter : can be found at `http://www.dmtf.org/newsroom/newsletter/2006/05/page4`, May 2006.
12. Knublauch H. : An Agile Development Methodology for Knowledge-Based Systems Including a Java Framework for Knowledge Modeling and Appropriate Tool Support, Dissertationsschrift (PhD thesis), University of Ulm (2002)
13. Object Management Group - Specifications and Process, can be found at `http://www.omg.org/gettingstarted/specsandprods.htm#SpecProd`, May 2006.
14. Cranefield S. : Networked Knowledge Representation and Exchange using UML and RDF, Journal of Digital Information, Volume 1 Issue 8 Article No. 44, 2001.