Using the CIM Conceptualization in Autonomic System Management: the System Management Ontology Project

Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Giuseppe Santucci
Dipartimento di Informatica e Sistemistica
Università di Roma La Sapienza
lastname@dis.uniroma1.it

Antonio Perrone, Guido Vetere
IBM SWG Rome Lab
firstname.lastname@it.ibm.com

1. Introduction

The administrator of current IT environments is facing a continuous increase in the complexity that s/he needs to handle. S/he can be compared to a power jet pilot sitting in front of a console with a plenty of gouges, leds, counters, each of them measuring or showing a particular part of the overall aircraft status. The pilot looks to all these indications, correlate each other these data sources, provide them a context (the air craft is landing, is taking off, and so forth) and, on the light of his knowledge and experience, he can understand if the system is working within acceptable parameters or experiencing a problem or can anticipate if the system is going to face a crisis state. The previous example suggest us two considerations:

- You need data collection about the several components of the system to implement system management.
- Collected data, in almost any case, are useless, without the knowledge and experience (we will refer to this ensemble as best practices) of the "pilot".

To address the autonomic computing challenge, systems management has to shift from a data driven approach to a knowledge-based approach able to alleviate the problem of making sense of the collected data. It can be difficult to draw any conclusion about a single system or about an application from the raw data provided by most systems management tools. A graphical performance monitor and an event viewer tool can be useful, but only an experienced administrator can detect if a problem exists or not. In the mind of the administrator exist the right domain knowledge that allows him to opportunely correlate a set of data, metrics and events obtained from the tools into information useful to answer relevant questions such as *'Is there a bottleneck?' 'What is the root cause of this bottleneck?' 'How do I can fix it?'* In this context, best practices are the way to correlate data about system resources to identify problems and trends and maintain the systems in a status that allows them fulfilling their business objective.

The first step in making system management tools able to own, deploy and actuate best practices in an autonomic computing environment is building an appropriated conceptualization in the system management domain. This conceptualization layer, or ontology, would provide the appropriate vocabulary to express, manipulate and share knowledge about the autonomic environment.

The Common Information Model developed by the Distributed Management Task Force organization is an approach to the modeling of systems for management that applies the basic structuring and conceptualization techniques of an object-oriented paradigm. The approach uses a uniform formalism that supports the cooperative development of an object-oriented schema across multiple organizations. The CIM standard

schemas, on which system databases on CIM-enabled machines (CIMOM) are based, is having a wider acceptance and support year by year. They provide an interesting set of classes and relationships among classes, based on UML, that can be considered an appropriate starting point for the development of a system management domain conceptualization, to be used for supporting automatic reasoning tasks.

The System Management Ontology project is a joint research initiative of IBM SWG Rome Laboratory and the Department of Informatic Engineering of University of Rome "La Sapienza", Italy. The project has been recently outlined in [Lanfranchi et al, 2003], with emphasis on the relevance in the field of Autonomic Computing. This paper aims at giving readers an insight in the knowledge representation and reasoning methods at the basis of the project, and illustrating the way the DMTF Common Information Model can be used as ontology of the system management domain, with the purpose of modeling state descriptions to be exploited in automatic system management tasks.

The first part of this paper illustrates the adoption of Description Logics as a formalism for representing system management knowledge suitably for automatic reasoning. Then, an overlook on the architecture of a first prototype is given, along with a picture of some basic technical options. Finally, a discussion of ontological issues concerning the CIM conceptualization is presented.

2. Knowledge Representation and Reasoning for System Management

As mentioned in the introduction, there is a general consensus on expressing the ontology of a domain of interest for System Management in terms of classes and relationships between classes, by making use of class based formalisms such as UML [Rumbaugh et al, 1998]. In particular, CIM is a standardized conceptualization of System Management that is extensionally supported by a wide range of automatic systems. Interestingly, it is possible to re-express such UML-like formalizations into specific forms of formal logic, called *Description Logics*, which are well behaved from a computational point of view. This can be profitably exploited by system management tools to provide systems with an autonomic behavior based on automated reasoning in such logics.

Description logics (DLs) have been introduced in the early 80's to provide a formal ground to class-based knowledge representation formalisms such as Semantic Networks and Frames. Since then they have evolved into powerful logics that are able to capture virtually all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases [Calvanese et al, 1999-1, 1999-2]. DLs are also at the base of state-of-the-art ontology languages like OIL and DAML+OIL [Fensel et al, 2001], which in fact can be viewed as dialects of DLs. One of the distinguishing features of the work on these logics is the detailed computational complexity analysis both of the associated reasoning algorithms, and of the logical implication problem that the algorithms are supposed to solve. By virtue of this analysis, most of these logics have optimal reasoning algorithms, and practical systems implementing such algorithms are now used in several projects [Horrocks, 2001].

Generally speaking, in DLs, the domain of interest is modeled by means of *concepts* (unary predicates), which denote classes of objects, and *relations* (n-ary predicates), which denote relationships among classes. A DL is characterized by three basic components:

1. A *description language*, which specifies how to construct complex concept and relation expressions, by starting from a set of atomic symbols and by applying suitable constructs. Typical constructs include, besides Boolean constructs on concepts and relations, restricted forms of quantification, which allow for expressing typing constraints on relation components,and numerical restrictions, which allow for expressing participation constraints as well as more general cardinality constraints.
2. A *knowledge specification mechanism*, which specifies how to construct a DL *knowledge base*, in which properties of concepts and relations are specified by means of suitable *assertions*. Typical kinds of assertions allow for expressing inclusions between concepts and relations respectively, forms of concept/relation definitions, and universal constraints of various kinds.
3. A set of *reasoning procedures* provided by the DL, which allow one to carry out suitable inferences from the knowledge expressed in the knowledge base, such as logical implication, knowledge base satisfiability, concept/relation satisfiability, concept/relation subsumption.

The expressive power of a DL depends on the generality of the constructs allowed in the description language and on the kind of assertions allowed in knowledge bases. In general there is a tradeoff between the expressive power of a DL and the computational complexity of the associated reasoning tasks. Such a tradeoff has been thoroughly investigated and DLs that are "optimal" with respect to such a tradeoff have been devised. That is, such DLs have sufficient expressive power to express most constructs of class-based formalisms (e.g., they can express the UML class diagram constructs) while admitting sound, complete and terminating inference procedures. A notable example of such DLs is $\mathcal{DLR}$ [Calvanese et al, 1998-1, 1998-2, 2001], which is at the base of the work described here.

Automated reasoning tools for expressive DLs, including $\mathcal{DLR}$, have been developed. Such tools are highly optimized in order to cope with the inherently high computational complexity of such logics, and in practice can deal with quite large knowledge bases, such as those resulting from modeling real world application domains [Berardi et al, 2001]

2.1. The Basic Formalism

SMO components, including CIM schemas and user models are expressed using UML class diagrams with specific conventions. Since $\mathcal{DLR}$ is able to fully capture UML class diagrams together with the specific conventions adopted in SMO, it can be used to equip SMO with automated reasoning capabilities. More specifically:

- SMO, including the CIM Core Model and Resource Models, are expressed in terms of $\mathcal{DLR}$ knowledge bases.
- New classes describing additional concepts of interest are translated to $\mathcal{DLR}$ concepts and, by exploiting the automated reasoning capabilities of $\mathcal{DLR}$, are automatically classified with respect to the existing classes.
- Queries posed to SMO are translated into queries to the corresponding $\mathcal{DLR}$ knowledge base, and are answered by evaluating the corresponding concepts on the available data.

Next we illustrate the main ideas behind the encoding of SMO into $\mathcal{DLR}$ knowledge bases. We introduce for each class *C* an atomic concept `C`, for each attribute *a* an atomic binary relation `a`, for each aggregation *G* an atomic binary relation `G`, and for each association *A* with roles $r_1,\ldots,r_n$ an atomic concept `A` and *n* atomic binary relations $r_1,\ldots,r_n$. The latter allows for dealing with associations with and without corresponding association class in a uniform way. Built-in predicates denoting arithmetic comparisons may also be used. Such predicates are suitably evaluated in query answering, but are treated as ordinary atomic binary relations during classification. Note that more sophisticated forms of reasoning based on the special meaning of such predicates over concrete numerical domains could also be used [Lutz, 1999]. However, state-of-the-art implemented description logics reasoning systems still do not support concrete domains.

We introduce suitable assertions that capture the semantics of the various UML constructs. For each attribute *a* of type *T* for a class *C*, we introduce a $\mathcal{DLR}$ inclusion assertion encoding the typing of the attribute:

```
(C ISA (FORALL [1] (a IMPLIES (2:T))))
```

Such an assertion literally expresses that each instance of concept `C` participating as first component to the binary relation `a`, is related by `a` only to objects/values that are instances of `T`.

The multiplicity *[i..j]* of attribute *a*, expressing that for each instance of the class *C* there are at least *i* and at most *j* values of *a*, is captured by means of the assertion:

```
(C ISA ((ATLEAST i [1] a) AND (ATMOST j [1] a)))
```

Each association *A* relating classes $C_1,\ldots,C_n$ via roles $r_1,\ldots,r_n$ respectively is represented by the atomic concept `A` together with the inclusion assertion

```
(A ISA ((ATMOST 1 [1] r1) AND … AND (ATMOST 1 [1] rn) AND
        (SOME [1] (r1 AND (2:C1))) AND … AND (SOME [1] (rn AND (2:Cn))))
```

For each aggregation *G* between a containing class C_1 and a contained class C_2, and represented by the atomic binary relation G, we encode the typing of *G* by means of the inclusion assertion:

(G **ISA** ((1:C_1) **AND** (2:C_2)))

Multiplicities on binary associations and on aggregations can be represented in a similar way as multiplicities on attributes.

A generalization between a class *C* and a derived class *D* can be represented using the inclusion assertion

(D **ISA** C)

Given generalizations (D_1 **ISA** C),…, (D_n **ISA** C), the covering constraint asserting that each instance of *C* must appear among (at least) one of $D_1,\ldots,D_n$, can be expressed as

(C **ISA** (D_1 **OR** … **OR** D_n))

Analogously, disjointness constraints among the derived classes $D_1,\ldots, D_n$ can be formalized as

(D_i **ISA** (**NOT** D_j)), for each $i \neq j$

Additional forms of constraints on the class diagram can also be expressed in $\mathcal{DLR}$, such as

- More general forms of negative information, e.g., to enforce that no instance of a class has a given attribute, or that no instance of a class participates in a given association or aggregation;
- More general forms of disjunctive information, e.g., to express that a given attribute is present only for a specified set of classes;
- Subset constraints on attributes, associations, and aggregations.

More generally, the expressive power of $\mathcal{DLR}$ allows to formalize several types of constraints that allow to better represent the application semantics and that are typically not dealt with in a formal way.

$\mathcal{DLR}$ reasoning tasks allow for automated support in dealing with SMO, fully taking into account the semantics of the various constructs and the additional constraints specified in SMO. In particular, the following tasks can be reduced to $\mathcal{DLR}$ reasoning tasks:

- *Consistency of the class diagram*, i.e., whether the diagram admits an instantiation. If this is not the case, there is no set of instances that satisfies the diagram, which indicates that the definitions altogether are inconsistent.
- *Class consistency (association/aggregation consistency)*, i.e., whether there is an instantiation of the diagram such that a given class (association/aggregation) has a nonempty extension. Observe that, if this is not the case, then there is an inconsistency in the class specification, or at the very least the class is inappropriately named since it is a synonym for the empty class.
- *Class subsumption (association/aggregation subsumption)*, i.e., whether the extension of one class (association/aggregation) is a subset of the extension of another class (association/aggregation) in every instantiation of the diagram. This property suggests the possible omission of an explicit generalization. Alternatively, if all instances of the more specific class are not supposed to be instances of the more general class, then something is wrong in the rest of the diagram, since it is forcing an undesired conclusion.
- *Classification*, i.e., determining class subsumption between all pairs of named classes.
- *Detection of redundancies*, e.g., if two classes (associations/aggregations) subsume each other, then one of them is redundant.
- *Refinement of properties*, e.g., when the properties of various classes and associations/aggregations interact to yield stricter multiplicities or typing than those explicitly specified by the designer. The simplest cases arise with multiple inheritance.

By making use of state-of-the-art reasoning tools supporting $\mathcal{DLR}$, all of the above reasoning tasks can be effectively realized.

2.2. The Extension Formalism

With respect to a database system like CIM, where classes are primitive constructs whose instances are created by processes running on singular machines, SMO gives the possibility to introduce new classes described in terms of necessary and sufficient constraints, whose instances can be dynamically evaluated without changing, supplementing, or populating the primitive databases.

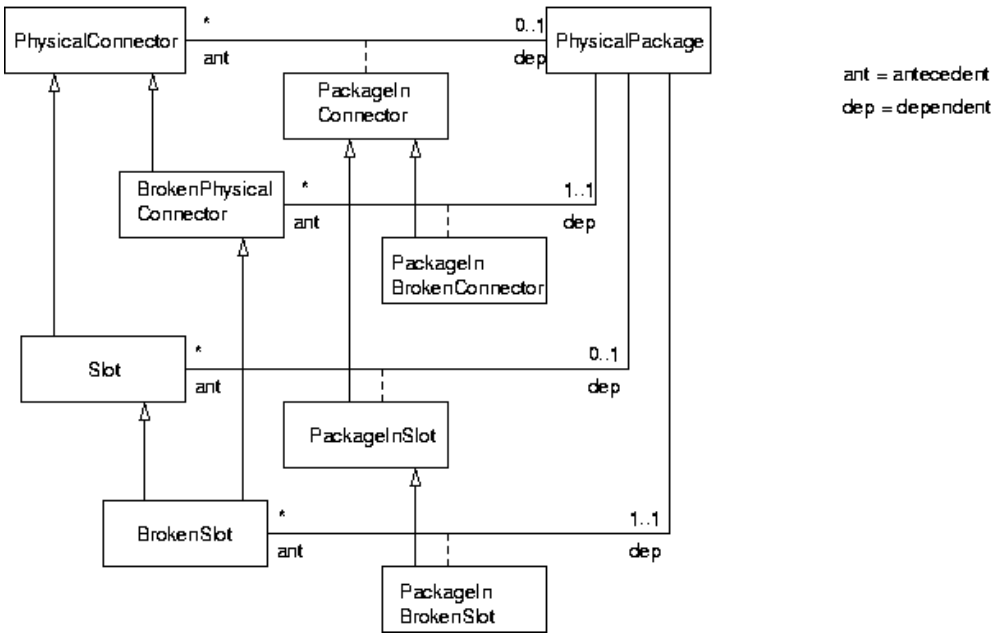

Figure 1

Figure 1 shows a UML class diagram obtained from the Physical Specification, which is part of the CIM Schema, version 2.7 (released in 21/04/2003) available at *http://www.dmtf.org/standards/cim_schema_v27.php*.

The CIM schema above models the relations between physical connectors, physical packages and slots, i.e., between hardware devices such as cards (referred to as physical packages) that can be inserted into suitable supports such as slots (referred to as physical connectors).

With respect to the CIM Specification, we have added the class `BrokenPhysicalConnector` and a subclass `BrokenSlot` with the purpose of modeling, respectively, a broken physical connector and a broken slot. Possibly we may define such classes in terms of the values of attributes of classes/association in CIM. For example, one may define a broken physical connector as a physical connector that has reached an error status from which there is no recovery mechanism. This may be encoded by considering the `OperationalStatus` attribute of class `ManagedSystemElement` from which `PhysicalConnector` is derived, and introducing the following $\mathcal{DLR}$ assertion:

```
(BrokenPhysicalConnector DEFN
    (PhysicalConnector AND
     (SOME [1] (OperationalStatus AND (2: "Non-Recoverable Error"))))
```

where `"Non-Recoverable Error"` is a particular value that the attribute `OperationalStatus`, inherited from `ManagedSystemElement`, may assume. For the considerations that follow, however, we do not need to enforce any particular definition for the classes `BrokenPhysicalConnector` and `BrokenSlot`; we only enforce certain properties (see below).

Apart from the above two classes we have also introduced two associations PackageInBrokenConnector and PackageInBrokenSlot, relating respectively instances of BrokenPhysicalConnector and BrokenSlot to instances of PhysicalPackage. Note that, a physical connector contains at most one physical package. Broken physical connectors are particular kinds of physical connectors. Since we are not interested in monitoring broken connectors in which no package is inserted, we assume that in each physical connector a package is inserted (multiplicity 1..1). A slot is a particular physical connector that contains at most one physical package. A broken slot is a slot and a broken physical connector, which contains exactly one physical package. As before, we want to monitor broken slots in which a package is inserted, therefore the class BrokenSlot participates to association PackageInBrokenSlot with multiplicity 1..1. Note that each package that is contained in a slot is also contained in a connector, each package that is contained in a broken connector is also contained in a connector, and each package that is contained in a broken slot is also contained in a slot. Finally, a physical package is contained in zero or more physical connectors, broken physical connectors, slots and broken slots. For brevity, we have not detailed attributes in this example.

According to the discussion made so far, the classes BrokenPhysicalConnector and BrokenSlot are expressed in $\mathcal{DLR}$ as follows.

```
(BrokenPhysicalConnector ISA
    (PhysicalConnector AND
     (ATLEAST 1 [2] (antecedent AND (1:PackageInBrokenConnector))) AND
     (ATMOST 1 [2] (antecedent AND (1:PackageInBrokenConnector)))))

(BrokenSlot ISA
    (Slot AND BrokenPhysicalConnector AND
     (ATLEAST 1 [2] (antecedent AND (1:PackageInBrokenSlot))) AND
     (ATMOST 1 [2] (antecedent AND (1:PackageInBrokenSlot)))))
```

The associations PackageInBrokenConnector and PackageInBrokenSlot are captured by the following DL formulas:

```
(PackageInBrokenConnector ISA
    (PackageInConnector AND
     (ATMOST 1 [1] antecedent) AND (ATMOST 1 [1] dependent) AND
     (SOME [1] (antecedent AND (2:BrokenPhysicalConnector))) AND
     (SOME [1] (dependent AND (2:PhysicalPackage)))))

(PackageInBrokenSlot ISA
    (PackageInSlot AND
     (ATMOST 1 [1] antecedent) AND (ATMOST 1 [1] dependent) AND
     (SOME [1] (antecedent AND (2:BrokenSlot))) AND
     (SOME [1] (dependent AND (2:PhysicalPackage)))))
```

2.3. Reasoning on the Ontology

By reasoning on the part of the class diagram described above one can infer that the association PackageInBrokenSlot is contained in the association PackageInBrokenConnector, i.e. that if a package is inserted into a broken slot, it is also inserted into a broken connector. The class diagram shows that each physical connector contains at most one physical package, and that each broken slot is also a slot and a physical connector. Therefore, if a physical connector participates to the association PackageInConnector, it participates to at most one tuple of the association. Hence, if this physical connector is actually a broken slot, the same tuple is also contained in the association PackageInBrokenSlot. Additionally, since a broken slot is a broken physical connector, it participates to the association PackageInBrokenConnector. Therefore, since the multiplicity is 1..1, there cannot be two different tuples, one in PackageInBrokenConnector and one in PackageInSlot, relating the same broken slot to two different physical packages. This implies that each tuple

of PackageInBrokenSlot is also a tuple of PackageInBrokenConnector. This inference is easily drawn by state-of-the-art DLs automated reasoning tools from the $\mathcal{DLR}$ knowledge base corresponding to the above class diagram.

Observe that this kind of reasoning allows one to make explicit knowledge that is only implicit in the ontology. Apart from detecting properties of interest, such as inconsistencies and redundancies, this allows for avoiding the explicit introduction in the ontology of concepts which can be inferred, thus keeping the ontology simpler to manage and to maintain.

2.4. Query Answering

Each $\mathcal{DLR}$ concept expressed over the class diagram can be directly translated into a query on the database corresponding to an extension of the diagram, by taking into account the first-order semantics of $\mathcal{DLR}$. In particular, for a SMO concept that is either a CIM class or a definition based on CIM classes, one can automatically obtain an SQL query to fetch the extension of the SMO concept from a CIMOM database. We illustrate this on a simple example. Consider the concept above used for defining the class BrokenPhysicalConnector:

```
(PhysicalConnector AND
 (SOME [1] (OperationalStatus AND (2: "Non-Recoverable Error"))))
```

The SQL query retrieving the extension of the class BrokenPhysicalConnector is then as follows:

```
(SELECT PC.id
 FROM   PhysicalConnector PC, ManagedSystemElement ME,
 WHERE  PC.id = MC.id AND MC.operationalstatus = "Non-Recoverable Error")
```

where we have assumed that: *(i)* each class is represented by means of a relational table in which the attribute id is the primary key and all the (single value) attribute of the class are represented by attributes of the relational table itself; *(ii)* each ISA is represented by means of foreign key in the subclass to the id of the superclass.

As another example, consider the following concept denoting the set of PhysicalPackages that are related via the association PackageInConnector to a BrokenPhysicalConnector:

```
(PhysicalPackage AND
 (SOME [2] (dependent AND
            (1:(PackageInConnector AND
                (SOME [1] (antecedent AND (2:BrokenPhysicalConnector)))
                ))
            )))
```

The SQL query retrieving the extension of the above concept is as follows:

```
(SELECT PP.id
 FROM   PhysicalPackage PP, PackageInConnector PC,
        BrokenPhysicalConnector BC
 WHERE  PP.id = PC.dependent AND PC.antecedent = BC.id)
```

where we have assumed that each association is represented by means of a relational table containing one attribute for each role of the association, having as foreign key the id of the class to which the role is related. Observe that the first join could be avoided (and so could be the first conjunct of the corresponding concept). However, one is in general interested in retrieving not only the id of the instances of the class, but also values of attributes (which we have not considered in this example).

Interestingly, by taking into account the constraints imposed by the class diagram, one can show that the above query is equivalent to the following one, in which the last join is avoided:

```
(SELECT PP.id
 FROM   PhysicalPackage PP, PackageInBrokenConnector PC,
 WHERE  PP.id = PC.dependent)
```

Observe that the equivalence between this query and the one above holds only due to the constraints in the class diagram. In particular, the equivalence holds since each instance of `BrokenPhysicalConnector` necessarily participates to the association `PackageInBrokenConnector`, which is a specialization of `PackageInConnector`, and each `PhysicalConnector`, including `BrokenPhysicalConnector`s, participates at most once in `PackageInConnector`. Again, automated reasoning support may allow for automatically detecting such kinds of optimizations.

3. The SMO Development Environment

This section provides the reader with a general idea of the SMO development environment. SMO has client-server architecture, as shown in Figure 3, and is mainly intended for an Intranet usage. The client component, which has the role of ontology browser/editor, is developed as a Plug-in in the Eclipse environment [ECLIPSE]. The server component is a J2EE Web Service, which delegates most of the functionalities to specific Java modules. Communications between client and server are therefore carried on the HTTP/SOAP protocol. An LDAP directory supplied with a suitable schema acts as database for ontologies, dictionaries, and other specific data. The reasoning module is based on FaCT [Horrocks 1998]. At the current stage, the reasoner doesn't have a direct access to the LDAP repository of ontology data, and import/export files are used for data exchange.

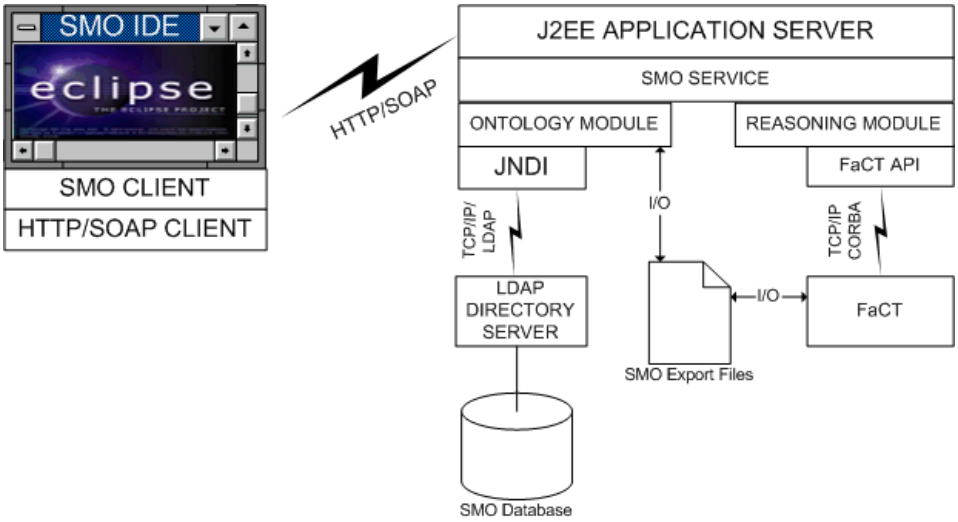

Figure 2

The user interface is available as a Perspective in the Eclipse runtime shell, i.e. a predefined set of windows associated with a specific task. This has been developed by integrating SWT widgets with the JFace library and the GEF (Graphical Editor Framework) plug-in for the graph drawing.

The Eclipse Plug-in is composed of five views on the ontology the user is interacting with. Four of them are intended for querying and browsing the ontology and one devoted to allow defining new ontology concepts.

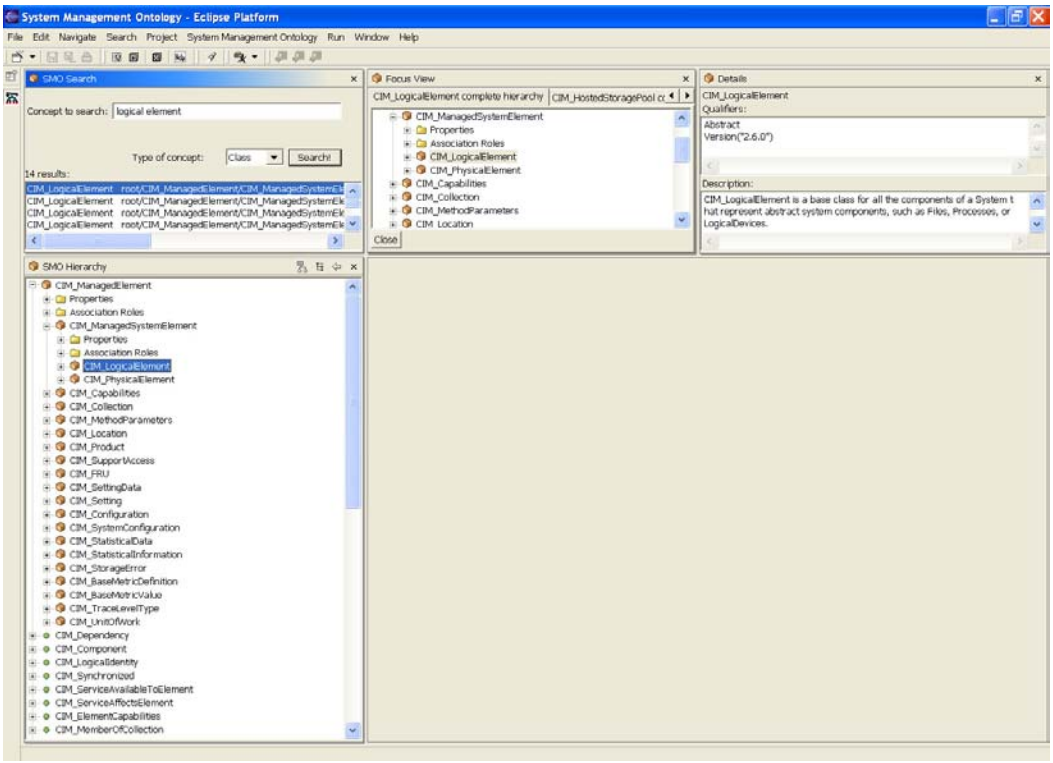

Figure 3

In Figure 3, a typical user session is depicted. Through the SMO Search window the user can search the concept(s) s/he is interested in. The search is driven by a controlled vocabulary, extracted from CIM class and attributes names (e.g. `CIM_LogicalElement -> logical, element`). An auto-completion mechanism allows the user for easily writing down terms that actually belong to the ontology. Also, the user can exploit some Meta information to narrow the query answer (e.g., the user can specify that the term s/he is looking for refers to a class). The user is presented a list of either class or attributes matching the search criteria. Then, s/he can double-click the intended one and thus be pointed to the corresponding node of the SMO Hierarchy, which is presented in a Tree View. Such a window contains the topmost ontology concepts and automatically reacts to the user selection in the SMO search windows, opening a path down to the selected concept. Tree nodes corresponding to concepts can be expanded to show hierarchy, attributes, and association roles. The other two windows (Focus View and Details) provide further pieces of information about the explored concept. Note that the search activity is not mandatory: the user can just browse the tree presented in the SMO Hierarchy, exploring the ontology content by hand.

Once the user finds a relevant concept, s/he can drag it in the Editor Window with the purpose of constructing a new ontology concept. The idea is that the Editor Window shows the classes the user focused on, allowing for defining different types of relationships among them, either exploiting the ontology associations or declaring new ones based on comparison operators. The user interacts with the classes graphically, by drawing links between classes. Then, a Wizard drives the user in specifying the intended relationship, by allowing semantically consistent constraints to be set. The expressive power of the graphical query interface is limited to *conjunctive queries,* which trade-off expressiveness and simplicity.

By interacting with the reasoning module, it is possible to activate a consistency check that validates the user definition comparing it with other previously defined indications. Redundancies and or inconsistencies are graphically outlined on the drawing, giving a visual feedback of the reasoning outcomes.

4. Beyond the CIM Conceptualization

This paper mainly deals with knowledge representation formalisms and reasoning. In this section, however, we want to start a discussion about giving SMO models a valuable content, i.e. making them effective in describing real things, events, relationships, states of affairs, and so on. At the time being, the SMO basic content is provided by CIM standard schemas, but it is important to make clear that this choice is not compelling. In fact, providing a mapping of CIM concepts to a different ontology would allow the adoption of a different conceptual backbone, while keeping the bindings to the CIM extensional layer.

Generally speaking, a good model should provide an empirically adequate conceptualization that captures the essence of the target domain, facilitates the knowledge sharing, prevents misuses and interpretation errors [Guarino, 1994]. Coming to ontological adequacy is not an easy task. On one hand, modelers would benefit of the possibility to apply strong a-priori structuring principles, on the other hand they cope with large and complex real-world domains, where legacy knowledge corpora already exist and must be taken into account. A promising approach to this kind of complexity is based on the adoption of general-purpose, standardized top-level ontologies such as [IEEE SUO], i.e. a small number of highly abstracted concepts valid for any domain. Based on this kind of ontological foundation, the development of domain-specific models comes out a process of particularization, while refactoring existing knowledge results in working out a sort of conceptual mapping. Even if a generally agreed reference ontology has not been released yet, a number of basic and widely accepted ontological distinctions can be already exploited, both for driving the development of high quality domain models and for reforming available conceptualizations with the purpose of increasing their adequacy.

With respect to some of generally accepted ontological principles mentioned above, CIM schemas show a number of deviations that could limit somehow their usage for the purposes of our work. For this reason, our future plans include a complete refactoring of the CIM conceptualization, by remapping some of the most general CIM classes to a principled top-level ontology. Even if this work is just at the beginning, the awareness of these problems can help using CIM, as is, more effectively, and therefore it is worth to briefly discuss them here.

First, we have considered the commonly accepted distinction of *continuants* (e.g. objects) from *occurrents* (e.g. events) [Simons 1987]. Continuant concepts are abstracted from time, in that they don't have temporal parts (i.e. they are all present if they are present at all). Occurrents, on the other hand, flow in time and do have temporal parts. We have noticed that the CIM conceptual root represented by the class `LogicalElement` is mostly intended to account for continuants (e.g. `FileSystem`), and yet contains concepts like `CIM_Process` that would rather fall in the category of occurrents, because they do have temporal parts. Another important time-related concept like "state", which is essential for describing system dynamics, is spread over the entire CIM class hierarchy by a variety of attributes (e.g. `EnabledLogicalElment::EnabledStatus`). Furthermore, time related attributes, that would probably qualify their carriers as occurrents, are introduced in many classes that are placed under different conceptual roots. In conclusion, *dependency on time* appears to be not an analytical criterion for `CIM conceptualization`, while it is a crucial conceptual driver for modeling system dynamics.

Another important ontological principle that CIM seems to be missing is that of *rigidity* [Guarino et al, 1994]. Informally, rigid classes (often called *types*) provide their instances with a permanent *identity criterion*, while non-rigid classes (*roles*) do not. Types are well suited to represent natural kinds and artifacts (e.g. *human being, hammer*) whereas Roles are mostly intended to model behaviors and functions (e.g. *student, tool*). Although, in principle, nothing prevents type hierarchies to allow multiple inheritance, it is a good common practice to adopt the *Identity Disjointness Constraint* [Guarino and Welty, 2001] to model them as a tree of disjoint classes. Roles, on the contrary, can be freely arranged into multiple hierarchies. Event though types and roles can be mixed in the same hierarchy (provided that Roles do not subsume Types), and they usually come together indeed, it would be safe keeping the two hierarchies separated, for the reasons discussed in [Steimann 2000]. Now, having a look to the `ManagedSystemElement` hierarchy and related associations, which play a central role in CIM, there is evidence that Type concepts (e.g. artifacts like `FileSystem`) and Role concepts (e.g. functionalities like `LogicalDevice`) are both included, but no explicit distinctions are made. To some extent,

accounting for Role concepts is assigned to the `LogicalIdentity` association, which allows a sort of multiple inheritance, but this modeling style appears somehow problematic.

To give a preview of how the SMO ontology could look like in the future, we briefly describe the top-level we are working on Figure 4, which is inspired to DOLCE [Gangemi et al, 2002]. In this model, continuants and occurrents are represented by the concepts *Entity* and *Event*, respectively. Moreover, we have assumed that *Object* is a type, being strictly intended to represent concrete artifacts. Also, we have introduced the concept *Role* to keep together all the different roles that *Objects* can play, while the association named *embodiment* allows specifying, for each *Role*, its underlying *Object* types (e.g. *student* → *human being*). We also adopted the notion of *homeomericity* for distinguishing *Matter* from *Object* and *State* from *Process*. Intuitively, parts of homeomeric concepts can be all described by the same predicates (e.g. *a piece of clay*, *a run*), while this is not true for non-homeomeric concepts (e.g. *a cup*, *a fall*). Finally, conceptual roots for Locations in Time and Space have been provided, with binding associations with *Entities* and *Events*, respectively.

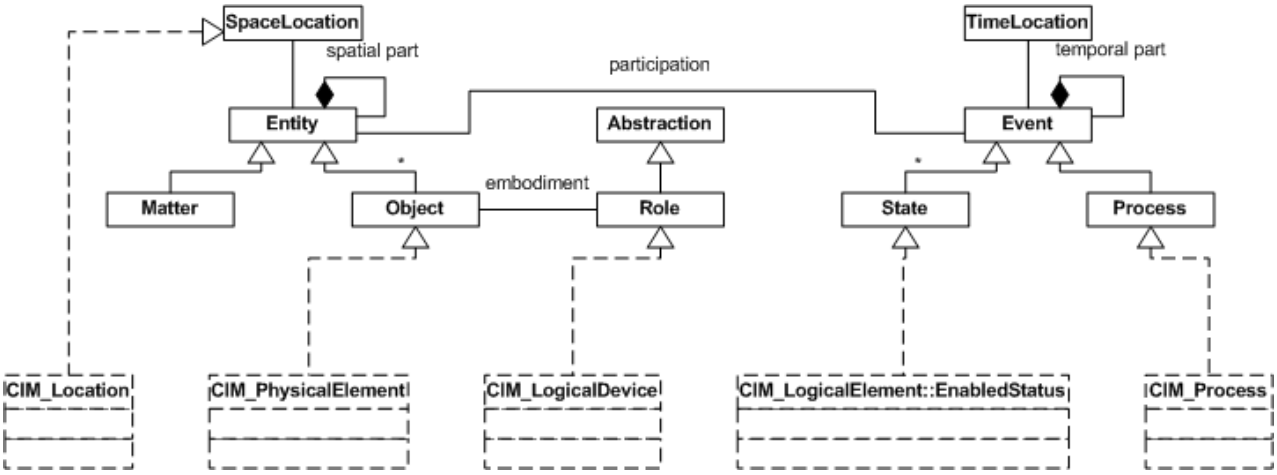

Figure 4

5. Conclusions

In this paper we have presented the approach taken by the SMO project to deal with the challenges posed by Systems Management in an Autonomic environment.

The main distinguishing feature of our work is to base the description of the system on precise logical and ontological notions, which allows for automated reasoning to detect relevant properties of the system. Such an approach is not limited to modeling only the structural/static aspects of a system defined by CIM class diagrams, as done in this paper, but can be extended in several directions. Along with the adoption of a suitable top level ontology, as discussed in the previous paragraph, one could introduce a temporal dimension to model dynamic aspects of a system. Several temporal and programming logics have been proposed for this task [Allen, 1996, Kozen and Tiuryn, 1990]. There has been some work on the interaction of such aspects with the structural ones captured by DLs, however more work has to be done to understand the impact on decidability and complexity of reasoning [Artale and Franconi, 2000].

The framework we are developing is designed to be integrated in the available infrastructures, and to leverage the existing standards and knowledge resources. The SMO development environment will allow the production of conceptual models of system behaviors, states, and dynamic that the next generation of system management application might exploit. A definite proof of the effectiveness of our approach, however, will come from experimenting the highly formalized methodology we developed in the real life of IT management.

6. Bibliography

[Allen, 1996] E. Allen Emerson. Automated temporal reasoning about reactive systems. In Logics for Concurrency: Structure versus Automata, F. Moller and G. Birtwistle editors. LNCS 1043, Springer, pages 41-101, 1996.

[Artale and Franconi, 2000] A. Artale and E. Franconi. A survey of temporal extensions of description logics. Annals of Mathematics and Computer Science, volume 1-4, pages 171-210, 2000.

[Baader et al, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003.

[Berardi et al, 2001] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams using description logic based systems. In Proc. of the KI'2001 CEUR 2001 Workshop on Applications of Description Logics. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-44/, 2001.

[Cali et al, 2001] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning on UML class diagrams in description logics. In Proc. ofthe IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD 2001), 2001.

[Calvanese et al, 1998-1] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98), pages 149-158, 1998.

[Calvanese et al, 1998-2] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98), pages 2-13, 1998.

[Calvanese et al, 1999] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. J. of Artificial Intelligence Research, 11:199-240, 1999.

[Calvanese et al, 2001] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification constraints and functional dependencies in description logics. In Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), pages 155-160, 2001.

[Fensel et al, 2001] D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. IEEE Intelligent Systems, 16(2):38-45, 2001.

[Gangemi et al, 2002] A.Gangemi, N.Guarino, C.Masolo, A.Oltramari, L. Schneider, Sweetening ontologies with DOLCE. In Proc. of the Int. Conf. on Knowledge Acquisition, Modeling and Management (EKAW) 2002.

[Guarino 1994] N. Guarino, The ontology level, in R. Casati, B. Smith and G. White, editors, Philosophy and the Cognitive Sciences, Vienna: Hölder-Pichler-Tempsky, 1994.

[Guarino el al, 1994] N. Guarino, M. Carrara,and P.Giarretta, An ontology of meta-level categories, in Proc. of the 4th Int. Conf. on Principles of Knowledge Representation and Reasoning, (KR'94), 1994.

[Guarino and Welty, 2001] N.Guarino and C.Welty, Identity and Subsumption, LADSEB-CNR Internal Report, 2001.

[Haarslev and Möller, 2001] V. Haarslev and R. Möller. High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study. In Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), 2001.

[Horrocks, 1998] I. Horrocks. The FaCT system. In Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98), volume 1397 of Lecture Notes in Artificial Intelligence, pages 307-312. Springer, 1998.

[IEEE SUO] IEEE Standard Upper Ontology, working group P1600.1, http://suo.ieee.org/.

[Lutz, 1999] C. Lutz. Reasoning with Concrete Domains. In Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI`99), pages 90-95, 1999.

[Kozen and Tiuryn, 1990] D. Kozen and J. Tiuryn. Logics of Programs. In Handbook of Theoretical Computer Science – Formal Models and Semantics, J. van Leeuwen editor. Elsevier Science Publishers, pages 789-840, 1990.

[Rumbaugh et al, 1998] J. Rumbaugh, I. Jacobson, and G. Booch. The Unified Modeling Language Reference Manual. Addison Wesley Publ. Co., Reading, Massachussetts, 1998.

[Simons, 1987] P. Simons. Parts: a Study in Ontology. Oxford: Clarendon Press. 1987.

[Steimann 2000] F. Steimann. On the representation of roles in object-oriented and conceptual modeling. Data & Knowledge Engineering 35, Elsevier Science, 2000.