Behaviour modelling : a contribution to CIM

M. Sibilla, A. Barros de Sales, J. Broisin, P. Vidal

Institut de Recherche en information de Toulouse (IRIT), Paul Sabatier University, Toulouse – France

{sibilla, broisin, barros, vidal}@irit.fr

F. Jocteur-Monrozier

Centre National d'Etudes Spatiales (CNES) Toulouse - FRANCE

francois.Jocteur-Monrozier@cnes.fr

Abstract

From an informational management point of view, a class schema is a static representation in so far as it doesn't model behaviour of managed system elements.

Our approach aims at selecting works achieved by the Software Engineering domain (OMG/UML) so as to reach universal understanding : concretely, we defined an UML profile for CIM class modelling and we have used the OMG/UML Statechart diagram for modelling behaviour and generate code. We also design an active CIM_Dependency pattern, which simply describes behavior between managed elements of different classes such as executing actions from event occurrence.

We give an overview of mechanisms in order to support and deal with these behaviour descriptions in a distributed management environment. We conclude on the benefits of modelling behaviour, which result from our experimentations with our CORBA-based management platform-called Cameleon.

1. Introduction

From an informational management point of view, a class schema is a static representation in so far as it doesn't model behaviour of managed system elements. In the CIM core and common models, the CIM_Action Application schema alone models an installing process which can be automated.

Moreover, in management architectures, Event modules generally have their own specific models to express resulting actions to invoke when some types of events occur. We can't but regret its being uncorrelated to the class schema.

The Management domain suffers from a lack of design tools regarding Informational point of view. So, a conformance to Software Engineering standards leads us to use a common approach, common tools and gives us a common understanding of static and dynamic management knowledge.

Our approach aims at selecting works achieved by the Software Engineering domain so as to reach universal understanding. Following on from the CIM OMG/UML class diagram standard use, we have chosen to integrate the OMG/UML Statechart diagram for modelling behaviour.

This document is organized as follows:

The first part identifies the abstract elements of the Core model which are involved in modelling behaviour. Secondly, we focus on modelling Object behaviour by integrating UML Statechart diagrams into CIM. The third part deals with the behaviour among object classes. Finally, we give an overview of how we have implemented our approach in our distributed management platform, called CAMELEON.

2. Abstract elements of Behaviour modelling

Resting on the CIM common model which describes some static semantic knowledge of the managed world, we have retrieved the basic elements so as to model some behaviour knowledge (see figure 1).

Managed Object/Class

Managed objects all derive from CIM_ManagedSystemElement class. Before the CIM model 2.7 preliminary version was released, this class had a SINGLE "state" propriety (Status), which management requires as a priority piece of information. Its updating may result from invoking a method of its class, from an action or else from an external event.

Event

An event is an instance of the CIM_Indication class. In the network management domain, the importance of events (Alarm type for example) has been underlined in the very CIM meta-model (Class derived Indication). A specific inheritance graph derived from the CIM_Indication root class is defined in the Core and Common models. Subscribing and filtering concepts are specified in [1]. Indications may concern either the schema life cycle (ClassIndication: create, delete, update), or class instances life cycles, or else alert notifications (complex events or at a higher level issuing from a set of events) coming from managed objects or from an external source (instrumentation or integration intermediary).

Action

An action corresponds to an operation to be invoked on an element of the common model. The actions we first wish to be able to specify are: Creating or deleting an instance, Updating a property, and Invoking a method on a particular instance.

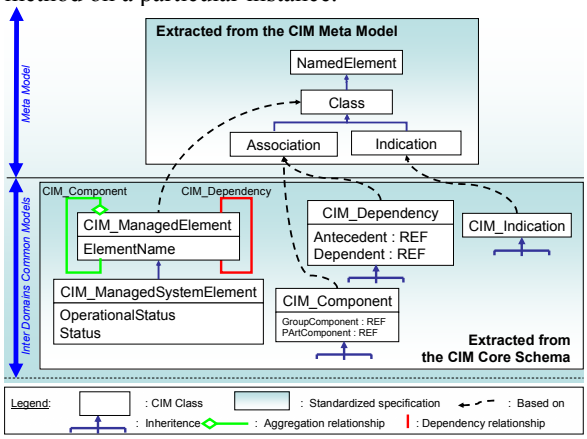

Figure 1 : Basic elements of Behaviour modelling

The behaviour knowledge we aim at formalizing so as to automate its management deals with managed objects –management abstraction of managed entities: we wish to express their change of "state" conditions as well as their behaviour during those changes and the behaviour influence among managed objects based on the Dependency relationship.

3. Objects behaviour

3.1 Existing work

As far as management is concerned, we first focus on the state of managed entities. It represents the operational condition of the managed entity at a given time. As regards the OSI management [2], the global state of an entity is the combination of three states, namely:

- the Operational state which has two possible values : enabled, disabled,
- the Usage state which has three possible values : idle, active, busy and
- the Administrative state which has three possible values : unlocked, locked, shuttingdown

Some rules have been graphically specified to informally standardize those states combinations and transitions.

The work achieved by O. Festor on objects behaviour and relationships formalization, along with EUROCOM Institute's, have highly contributed to the expression of the GDMO standardized information model using SDL'92 [3, 4, 5].

However, both the advent of the Unified Modelling Language -UML (especially here, its Statechart diagram) and the CIM common management models have led us to study a new combinatorial approach.

For the moment, in the CIM 2.7 preliminary version, some major experimental modifications are proposed. The OSI states management logic cannot be found directly and we are baffled at its management. As a matter of fact, the "CIM_ManagedSystemElement.OperationalStatus" and "CIM_Enabled LogicalElement.Enabled Status" properties are proposed (whereas the more general property "CIM_ManagedSystemElement.Status" is depredicated).

In accordance with the UML statechart diagram, our work has focused on a single state whose expression we wished to formalize so as to enrich the common abstraction model.

3.2 Integrating UML Statechart diagram into CIM

We are aiming at describing the behaviour of an object based on all its possible states. According to its state, certain methods will be available on the object or not, some code invoking will be possible during states transitions.

Our goal is to be able to specify "behaviour" through UML Statechart diagrams according to the following principles:

- by associating a Statechart diagram to a class, when possible

- by managing the states and possible transitions among those states
- by executing some code on pre-invoking and post-invoking a method or on entering and exiting a state
- by forbidding or ignoring certain methods depending on the object state

Specific grammar has been defined to have a textual representation of the UML Statechart diagrams so as to be able, then, to generate some code from those formal descriptions.

Such behaviour modelling will ensure security improvement (control) and will entail automated –and thus- more secure code generation. Besides, the graphical aspect of the diagrams cannot but make understanding easier, which will in turn improve the transition from specification to design and implementation.

The states automates descriptions for a set of classes can be regrouped into a single state file or placed into separate state files. This file abides by strict grammatical rules (referencing in Figure 2), which propose some labels for describing a states automate for the class under consideration [6].

3.3 Description of the Statechart Grammar

The "***action***" label makes the listing of all the methods available on the relative CIM class possible and must be filled in. True enough, this is redundant with the CIM class methods definition, but it has been introduced so as to ensure independent tools handling CIM models and statechart diagram descriptions.
Specific libraries can be integrated into the automate description code thanks to the "***import***" label. Each state is preceded by the "***state***" tag which defines the state description block. Within a state, some methods can be ignored by using the keyword "***ignore***". When entering and exiting a state, some specific code may be executed thanks to the labels "***on enter***" and "***on exit***". Before and after invoking a particular method, some code may be invoked by using the keywords "***on pre-invoke***" and "***on post-invoke***". This grammar also permits to define transition conditions between two states thanks to the keyword "***transition***".

Using Java code enriches our approach with an open processing, namely: method invoking on instances of the repository, Java specific code …

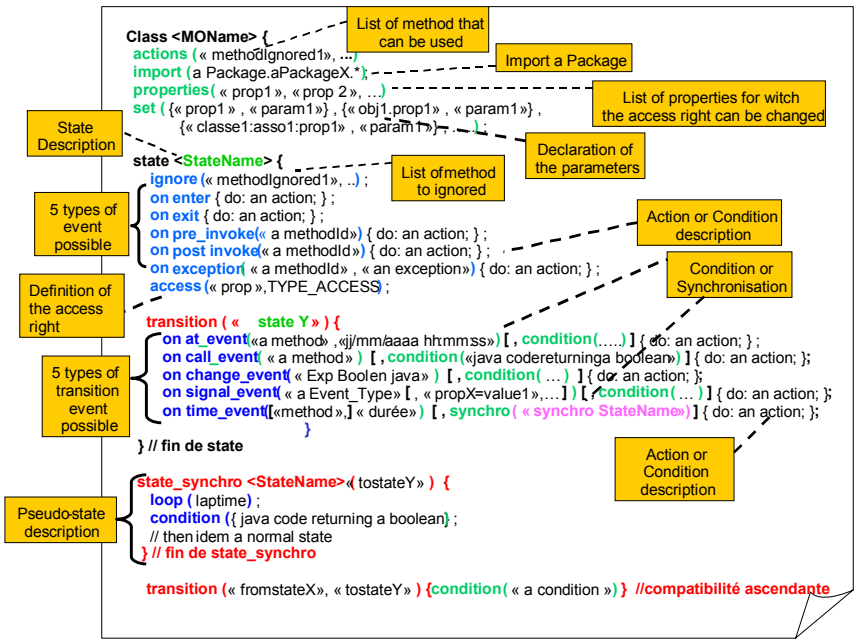

Figure 2: Principle of the description grammar of Statechart diagram

Advanced extensions

All the possibilities of the UMLv1.4 Statechart diagrams have not been taken into account in this proposal. However, we are now adding a few extensions (presenting in Figure 3) whose implementation requires the following management functions:

- Event management function (exception or CIM_Indication),
- Time management function, and
- The instance distinguished naming in a distributed context.

On entering a state, a timer can be triggered and an action can be executed after pre-defined period of time (Time Event in UML) thanks to the keyword "***after***".
Within a state, an exception occurring after some method invoking on an object (an event such as "Signal Event" in UML) may be mentioned thanks to the keyword "***on exception***" defining the expected exceptions.

Taking into account external events (such as CIM_Indication) which happened on other objects is interesting in our management context, for it meets with the concerns of behaviour macro-modelling. Capturing this type of events might aim at being able to specify some behaviour due to be found for a given state of an object on an event occurring.

Thanks to this grammar, we are able to associate a Statechart diagram to each class of the CIM model (derived from CIM_ManagedSystemElement). The presence of a state file associated to a class is signalled by a new "*state*" qualifier defined as follows:

Qualifier State: Boolean = false, Scope(class),
Flavor(DisableOverride,Restricted) ;

This grammar may be enriched so as to meet the statechart diagram designers' and users' needs. The strength of this diagram lies in its sheer formalism which makes it accessible to all.
State combination is still to be taken into account in this proposal.

4. Behaviour correlation among objects

4.1 Existing work

Initially, OSI management function "Attributes for representing relationships" defined a set of relationships specific to the OSI context. The "Generic relationship model" model has completed the first modelling without helping with any other purely functional aspects [7]. Modelling semantic links among managed objects refines management knowledge when solving problems or facing Alarm reporting. Nevertheless, if we are able to model and then to automate some part of the management expertise dealing with influence and behaviour dependency among entities, we will improve automation in controlling and correcting critical situations. In [8], specifying and classifying these relationships as well as dealing with them separately help improving the management functionality of anomalies. Nevertheless, modelling these relationships does not express any action to be executed facing some relevant triggered event.

CIM meta-model defines the Association concept to represent some semantic relationships among object classes. The stress is more particularly put on a composition relationship (CIM_Component) and on a dependency relationship (CIM_Dependency) in the Core model.

Our first investigation consisted in expressing and implementing some dynamics on dependency relationships, which behaviour influence stems from.

4.2 Active dependencies: a new pattern

A dependency relationship links together two object classes at least: one of them is identified as Antecedent and the other as Dependent. For readability's sake, we will use the English words Antecedent and Dependent throughout the paper to name the instances they refer to.

We wish to be able to express the link between an *event* occurring on an Antecedent instance and an *action* to be taken on the Dependency instance. This possibility must on no account modify the CIM_Dependency class since this class is defined in the CIM "Core" model in an abstract way. The action to be taken necessarily implies some elements (property, method,…) from the classes referred to by Antecedent and Dependent. Our contribution consists in specifying the different kinds of actions to be taken depending on the "target" element of the Antecedent instance.

Figure 4 presents an extract of the inheritance graph of the actions which can be expressed and associated to any relationship deriving from the CIM_Dependency class. The CIM_Dependency class links two instances of the CIM_ManagedSystemElement class - "Antecedent" and "Dependent" - related to each other.

The *ActionOnDependency* action is an abstracted class from which all possible actions to specify on a Dependency are derived. This generic action has three properties common to any action: its name (name), the language describing the selection query, and the selection query of the relevant event, raised by Antecedent.

The *DependencyAction* Dependency relation references an instance of the *CIM_Dependency* class, and the *ActionOnDependency* action associated with this Dependency. Several actions could be associated with an Active Dependency.

To illustrate our approach, let us take the two classes *CIM_System* and *CIM_Service*. There is a dependency relation on which we can express an action relating to a *CIM_InstIndication* event (see Figure 4).

An example of action we can express (and, implicitly, manage automatically) is given on Figure 5.

This action consists in ensuring a mapping of property values between Antecedent and Dependent. Here, the query specifies the event starting the realization of the action, namely: every update of a *CIM_System* Antecedent instance having its OperationalStatus state modified in "Aborted". This action can be associated with each *CIM_HostedService* Dependency instance for which an automation is required.

```
Instance of  PropertyValueMappingAction  {
     Name=  "AoD"
     TargetProperty = "OperationalStatus ";
     QueryLangage = "WQL" ;
     Query= "SELECT *
            FROM CIM_InstModification
            WHERE SourceInstanec  ISA CIM_System  AND
                  SourceInstance.OperationalStatus   == 15 OR …"
     SourceValue : {15, …}; // Aborted
}    TargetValue : {15, …}; // Aborted
```

Figure 5: An example of Action associated with an Active Dependency

Thus, the opening of WQL queries [9] "SELECT… FROM… WHERE…" leads us to consider general events (*CIM_InstIndication* instances) and to express consequent correlations and actions from designing phase. Based on the mechanisms defined in CIM Event, this query allows to create a filter in the manager responsible for the Antecedent object to subscribe with the reception of these events.

5. Implementation

CAMELEON [10] is a management platform for complex systems, resulting from a R&D project named SUMO - SUpervision et Maîtrise des Operations - carried out in collaboration by CNES, IRIT and ALCATEL CIT.

The components of the CAMELEON architecture are those of the WBEM architecture suggested by the DMTF [11], in which Management functions are implemented like they are defined in OSI Management [12].

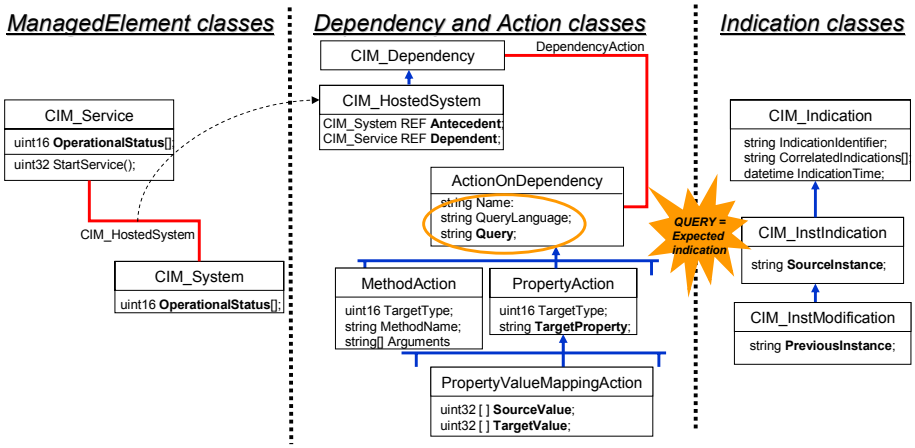

Figure 4: Active Dependency Pattern and Inheritance tree of ActionOnDependency class

Thus, there are not, within our architecture, dedicated or centralized components. It is a distributed organization of the management applications which introduces the concepts of Object Manager and Object Provider. In our architecture, any manager has a repository containing class descriptions and instances. These management information are:

1. Either instrumented by gateways towards heterogeneous management modules.
2. Or treated by management functions associated with the manager.

5.1 State Management

States description is established in a pre-defined formalism, a specific grammar which describes states diagram by means of keywords. It is held in a "state" file. A parser is then applied to the file and generates the corresponding java class which may be compiled. For example, before and after invoking a specific method, some code may be invoked by using the keywords "on pre-invoke" and "on post-invoke".

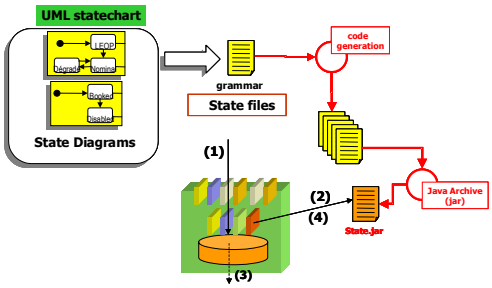

Figure 6 : Functional description of the State management fuction

(1) : method invocation to an object within the repository of an object manager
(2) : execution of the related pre-invoke
(3) : execution of the invoked method
(4) : execution of the related post-invoke

Within Object Manager, some transparent redirecting mechanisms enable us to redirect method invocation on "State" qualified objects to the state management function. Thus, by using Java typical reflection mechanisms, the state management function executes the code corresponding to the context of the concerned object (state value, invoked method, pre-invoke …)

5.2 A Management Function for Active Dependencies

Instances *of ActionOnDependency* class are treated by the Relation Management function (of Dependency, for the moment) in the following way:

- Searches for all instances of the *DependencyAction* class and subscribes to event triggering by Antecedent object, regarding the "query" element specified in each action.
- Waits for any event. When an event is received, this process searches for actions to be executed on the associated *Dependent* instance depending on the type of event (update, access …) and on the concerned *Antecedent* instance.

Active Dependencies management function is a basic function of any manager. It is therefore implicitly instanciated within any manager. Any active Dependency managed by a manager must be described in this manager repository abiding by the grammatical rules associated with Active Dependencies. It will be possible to execute an action on the referenced dependent element(s) after each selection of pertinent event.

Each Dependency instance binds two instances of classes being able to belong to various managers. Thus, for each Dependency instance, the management function will subscribe to each manager by creating a filter starting form the query expressed in the associated action. This function thus automates the initialization phase of the peer to peer "Push" model between managers responsible for the Dependencies and those responsible for the Antecedent referred in the Dependency relations.

6. Conclusion

On the basis of a meta-model and of universal models for network and services management engineering, we have contributed to the formal expression of both object behaviour knowledge (through the integration of UML Statechart diagram into CIM) and behaviour correlation among objects through modelling active dependencies.

Our contribution on the behavior modeling of CIM classes brings an answer based on a standardized approach broadly adopted by the Community of Software Genius. From experimentations, we can say that these statechart descriptions reduce code lines by 3 (classical program length has 1300 code lines rather than the corresponding state file length has 200 + 300 lines). Moreover, these descriptions are readable, extensible and "customizable". We also have possible code generation from diagrams. By using state machine diagrams, we can automate test step or make simulation.

These approach was extended by the definition of an Active Dependency pattern making it possible to specify a Management expertise between different objects easily. From its generic character (*CIM_Dependency*) and the initialization of a query language to express a selection of relevant events, we offer the elements to initialize the "Push" model between managers and to automate it.
Nevertheless, with opening as well as design and implementation flexibility in mind, both contributions complement each other.

10. References

[1] DMTF DSP0107. "CIM_Event Model White Paper". Version 2.6. March 2002

[2] IS 10164.2 ISO/IEC "OSI Management : State Management Function". Juin 1993

[3] S. Tata, L. Andrey, and O. Festor. "A practical experience on validating gdmo-based information models with sdl'88 and '92". In SDLForum'97, September 1997.

[4] E. Nataf. "Contribution à la spécification et à l'exploitation des relations entre objets de gestion de réseaux". *PhD thesis, Thèse de l'Universitè H.Poincarè*-Nancy I, France. octobre 1998.

[5] D. Sidou, S. Mazziotta, and R. Eberhardt. "TIMS : a TMN-based Information Model Simulator, Principles and Application to a Simple Case Study". *Sixth Int. Workshop on Distributed Systems : Operations and Management*, IFIP/IEEE, Ottawa - Canada. 1995

[6] M. Sibilla, D. Marquie, A. Barros de Sales. "Automatisation d'expertise de gestion: Intégration de descriptions formelles du comportement au modèle universel statique". Colloque francophone de Gestion de Réseaux Et de Services. Mars 2003.

[7] ISO/IEC IS 10165-7. "OSI Management : Generic Relationship Model". 1996

[8] Keller et al. "Dynamic Dependencies in Application Service Management". *International Conference on parallel and Distributed Processing Techniques and Applications*. Las Vegas, NV, USA, June 2000

[9]DMTF DPS 0104. "WBEM Query Language" (Draft)14 June, 2000

[10] Sibilla M., Barros De Sales A., Desprats T., Marquié D., Steff Y., Jocteur-Monrozier F., Rivière A-I.: "CAMELEON : A CIM Modelware platform for distributed integrated management". *The Annual DMTF Developers' Conference* (Academic Alliance Contest winner). San Jose, 2002.

[11] Thompson J.P.: Web-Based Enterprise Management Architecture. IEEE Communications Magazine, No. 3 (1998) 80-86

[12] ISO/IEC 10164. "Information Technology. Open System Interconnection. System Management". Parts 1 to 21.