Managed Object Behavior Inheritance and Scattering Support in TIMS[1]

Olivier Festor, **Dominique Sidou**, **Sandro Mazziotta**
Institut Eurécom, 2229 route des crêtes, B.P. 193,
06904 SOPHIA ANTIPOLIS CEDEX, France.
email: {festor | sidou | mazziott}@eurecom.fr

abstract

TIMS is an environment dedicated to the simulation of TMN systems. For this purpose, the environment uses both standardized definitions of Managed Objects and clear defined behaviors associated to them, in terms of syntax and semantics. To provide a good expressive power to the underlying behavior language, mechanisms such as scattering and inheritance, which are inherent to the standards, must be integrated into the environment. The way in which this integration can be achieved in TIMS is the goal of this paper. To this end, the TIMS approach to OSI-based Management Information Model simulation is presented and the problem of formally described behavior scattering and inheritance in the standardized GDMO notation is outlined. This outline then serves as a basis for presenting the different ways in which this problem can be solved in the TIMS tool-set.

Keywords : GDMO Managed Object Behavior Formalization and Simulation, Rapid Prototyping, Support of Behavior Distribution (Scattering and Inheritance).

1 Introduction

The TIMS project is concerned with the rapid prototyping of TMN systems. In the context of OSI Network Management such systems are viewed as a collection of Managed Objects (MOs) organized in a tree (the so called Management Information Base (MIB)). Each MO, results from the instantiation of a Managed Object Class (MOC) described in GDMO [Gdmo]. The overall MIB can then be manipulated though a set of well defined services [Cmis].

Learning about the information model itself, its nature and structure is an immediate benefit. Then, a more interesting benefit is to be able to test the information model according to basic scenariis combined inside complete test suites. Thus detection of misunderstandings about the information model, and erroneous or inconsistent configurations could be handled properly. Another interesting task that could be achieved with such a tool is the set up of reference configurations of TMN systems. This would enable one (e.g. an operator) to test the robustness of management applications, provided by a vendor, according to not only normal processing conditions but also unusual ones and even erroneous ones (introduction of corrupted / superfluous / loss of information. . .).

The standardized GDMO notation provides a good basis for the specification of management information bases. It provides a set of nine templates which can be used to specify all parts of Managed Objects (attributes, actions notifications, . . .) and their organization within a Management Information Base. But the notation does not provide any mechanism to describe formally the behavior associated with one or more managed objects. Moreover, behavior description in GDMO can be scattered across several templates and the behavior of a MO results from the composition of all these scattered behaviors along with those inherited from its super-class(es). From now on, distributed behaviors are defined as scattered behaviors along with inherited behaviors. To facilitate the development of clear and unambiguous Management Information Bases and to enable their simulation, a notation with well defined semantics for behavior specification must be provided and

[1]TIMS stands for TMN-based Information Model Simulator. This project is a collaboration between Eurécom Institute and Swiss Telecom PTT. It is supported by Swiss Telecom PTT, project F&E-288.

the way in which composition among templates as well as behavior inheritance must be clarified. Thus, the application of behavior distribution in the TIMS framework is discussed in this paper.

Section 2 defines the behavior paradigm employed in TIMS and identifies the need for scattering and inheritance support towards behavior. Section 3 gives a detailed description of the problems related to the distribution and lists possible treatments. The inheritance mechanism chosen within the OSI approach is also outlined in this section. Section 4 shows how these mechanisms can be supported in the current behavior implementation language of TIMS. Finally a conclusion of the presented work is given and future work is identified.

2 The behavior paradigm in TIMS

TIMS provides a behavioral simulation platform for validating OSI Network Management Information Models. The basic principle is to emulate the way in which MIB managed objects react after being solicited either by an invocation of some CMIS service primitive coming from a management party or by the occurence of a real resource change inside a network element. The TIMS platform is concerned with executing such emulated reactions, that are previouly formally defined as behaviors attached to the MIB description (GDMO specification) extended with relationship facility. A straightforward approach is to consider that behaviors manifest themselves as the propagation of effects among managed objects. Such behavior effects can be realized as (i) a MIT (Management Information Tree) alterations (a modification of an attribute value or a MO creation or deletion), (ii) an emission of a notification, and (iii) an action being executed. Obviously, in its turn each individual effect is another MO solicitation, with a possible associated behavior, and in fact, this results as a behavior propagation chain. The process completes when no more propagation is firable, or a saturation state is reached.

In this section we will present how the behavior of MOs is described in the TIMS approach and why the support of scattering and inheritance is required.

2.1 MO Behavior Formalization

In order to build any behavior simulation, MO behaviors have to be formalized in some way. Unfortunately, behaviors present in GDMO are of no use here because they are only pure prose where any other aspect involved in the management of a resource can be placed, i.e. anything ISO / ITU felt could not otherwise be specified properly.

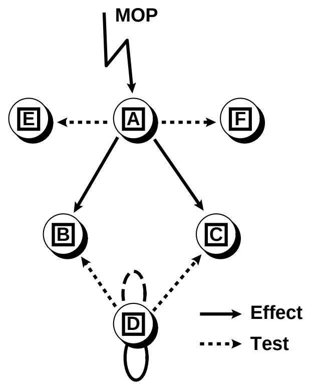
As illustrated in figure 1, two paradigms for behavior formalization are of interest in TIMS : triggered and untriggered behaviors. Triggered behaviors enables one to specify, in an imperative fashion, how a MO should react when solicited on any of its interfaces, namely the management interface or the one to the real resource. An untriggered behavior, on the other hand, is purely a declarative specification of a dependency of a MO with respect to other MOs.

In [CF93], a valuable contribution towards a systematic approach to the specification of behaviors, in terms of guidelines and methodologies is given. In particular, the scattering of behaviors among all the parts of the specification a MO consists of (i.e. packages, attributes, actions, notifications, name-bindings, parameters) and the inheritance of behaviors are discussed. The technical treatment of the gathering of distributed behaviors, in TIMS, at the MOC level is the purpose of this paper.

2.2 General Form of Behaviors

The general form of a simulated behavior is composed of :

1. **scoping context** : the relationship whose instances are concerned by the behavior.
2. **triggering context** : present only for triggered behaviors, specifies precisely the operation that triggers the execution of the behavior (attribute change, object creation, object deletion, action). Since the scoping context is a relationship, it is also necessary to specify which participant role this operation is applied.



1. Triggered behavior : a management operation is invoked on object **A** whose behavior exercise itself as side effects onto objects **B** and **C**, if additional testing conditions related to object **E** and **F** are fulfilled.
2. Untriggered behavior : object **D** depends on objects **B** and **C** : local side effects are propagated on **D**, if additional testing conditions on **B** and **C** are satisfied.

Figure 1: Triggered and Untriggered Behaviors.

3. **pre-condition** : testing condition upon the MIB state, to be satisfied in order to launch the behavior body.
4. **body** : the body of a behavior is an imperative / procedural piece of code. There is no a priori structure imposed on it. Tests and effects are intended to be used in this body, reflecting the dependencies between MOs. For instance an object may be updated, created or deleted according to the presence, absence or the state of another object. Since usual programming features (i.e. control flow structures, variable notation. . .) are required at this level, the use of an existing and well-known programming language is reasonable choice. This is also the approach taken in the DOMAINS project [FHHS93], which uses Eiffel construct to formalize behavior bodies. Though not mandatory, chosing the programming language of the targetted runtime environment may reveal very interesting in order to facilitate the implementation of the integration of behaviors.
5. **post-condition** : testing condition upon the MIB state, to be satisfied after completion of the execution of the behavior body.

One of the main constraints placed on the TIMS approach to behavior simulation is the support of the standardized GDMO notation and support of its features such as behavior distribution. This allows a behavior specifier to stay close to its GDMO specification when describing its corresponding behaviors.

2.3 Need for Relationship Facility

There are several relevant reasons to introduce a general relationship facility :

1. To formalize properly the simulated behaviors, a clean method to access other managed object is needed. The two kind of access to other MOs from a simulated behavior are **tests** and **effects**. Thus, behavior propagations, resulting as the chaining of tests and effects, are cleanly delimited by the use of well identified and defined relations. This mechanism has already been well identified by Rumbaugh [Rumbaugh88, RBP$^+$91]. Moreover, from the implementation point of view, this should enable to devise a straightforward behavior propagation engine.
2. Being able to model inter-MOs dependencies is very useful since hidden MO dependencies makes models hard to understand and behavior hard, if not impossible, to determine. In [CF93] separate modeling of relationships through dedicated MO classes is advocated, since inter-MO dependencies can then be specified as relationship behaviors.

In spite of the existing relationship mechanisms already present in GDMO (namely name-bindings and attribute pointers), a more general and non-restrictive relationship facility is required. This enables one to give a valuable representation to any relevant and possibly hidden inter-MO dependency present in its information model, thus potentially needed to formalize some given simulated behavior. To this end, the choice of the GRM [Grm94] seems the more appropriate for GDMO-based information models.

2.4 Behavior Integration Architecture

Figure 2, depicts the integration architecture : it is based on an agent toolkit used in a dummy fashion along with an eXternal Behavior Integration module in Scheme [Ce91].

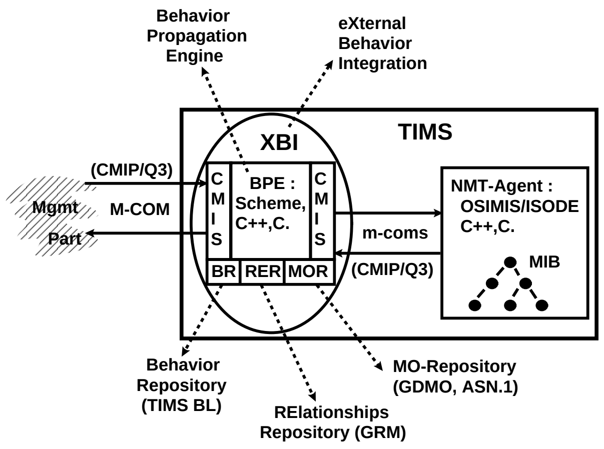

Behavior Propagation Engine
eXternal Behavior Integration
XBI
TIMS
(CMIP/Q3)
Mgmt
M-COM
Part
CMIS
BPE : Scheme, C++,C.
CMIS
BR
RER
MOR
m-coms
(CMIP/Q3)
NMT-Agent : OSIMIS/ISODE C++,C.
MIB
Behavior Repository (TIMS BL)
RElationships Repository (GRM)
MO-Repository (GDMO, ASN.1)

Figure 2 : Behavior Integration Architecture.

Dummy Agent Toolkit The runtime environment includes an agent toolkit used in a dummy fashion (no behaviors there). That is, it is used only as a MIB data store. This enables to keep things not too far from the real world and not having to re-implement all the generic management functionalities already developed here, such as CMIS processing (scoping, filtering) and Systems Management Functions (Alarm, Event, Log. . .).

eXternal Behavior Integration (XBI) Besides the agent toolkit, the XBI module is based on a Scheme [Ce91] interpreter enriched with a Behavior Propagation Engine (BPE), a CMIS-API, operators for an ASN.1 value notation, and obviously with the user's information model and associated behaviors.

Behavior Propagation Engine (BPE) As already claimed before simulated behaviors originates either in Management Operations (MOPs) either in Real Resource Changes (RRCs). Then, the behavior processing itself, in its turn, may incur other MOPs or RRCs to be exercised on other MOs and as a consequence this results as a complete behavior propagation chain. This process terminates at saturation, that is when no more behavior is firable. This component is very important in the TIMS box because it defines completely the operational semantics of the proposed Behavior Language, which is not an obvious task, since hybrid specifications combining both imperative and declarative behavior formalizations are allowed and even considered as key features. To summarize, and since more details about the BPE are out of the scope of the present paper, the BPE can be viewed as a usual forward chaining inference engine of behaviors acting upon an agent MIB.

Open Architecture The resulting architecture is open in several ways :

- Integrating behaviors externally enables one to switch to any other agent toolkit much more easily, than an internal behavior integration inside an agent toolkit and based on its proprietary API.
- Switching or adding support for another information model is possible with less effort. A new communication access module (e.g. an ORB. . .) has to replace the CMIS-API, and the associated operators for the value notation replacing those working on the ASN.1 value notation.
- Note that the resulting implementation is light-weighted. In its current stage, the previously mentioned APIs and the BPE are implemented with less than 1000 lines of code in the current scheme-based implementation.

2.5 Synthesis of TIMS Concepts

Finally, TIMS provides a full fledged object oriented modeling framework based on GDMO and extended with a general relationship facility based on the GRM. Onto this powerful representations, simulated behaviors can be conveniently plugged in a both imperative declarative fashion, and then, easily integrated and exercised in a quite realistic runtime environment. Moreover, the extensibility of the architecture (e.g. to other agent toolkits and even to other information models) suggests that the TIMS tool-set can also be viewed as a general purpose behavior simulation environment.

3 Behavior composition and inheritance in GDMO

When trying to compose the full behavior associated to a MOC, two different parts have to be considered. The first one concerns the scattering of behavior among several templates within the MOC. The second one concerns the composition of behavior through inheritance. These two parts of the composition are addressed below.

3.1 Behavior composition within one MOC

The GDMO notation allows the specification of behavior parts within several parts of a MOC description. For each of these parts (templates), which part of the behavior they support is detailed :

- attribute: behavior associated with an attribute describes how comparison operations (matching rules) have to be applied to the attribute. It also specifies its generic behavior which holds in any cases this attribute is present.
- action: in its associated behavior template, one must define the conditions under which the action can be invoked and the effects of its execution as well as the content of its response or error parameters.
- notification: the behavior template associated to a notification describes the conditions under which the notification is issued and the information carried within its parameters, if necessary.
- parameter: the behavior which is associated to a parameter aims at providing the semantics of this parameter. Parameters are typically used to convey additional information about specific processing errors.
- package: this behavior describes the dependencies between the components of the package, i.e. attributes, actions, notifications. This includes:
  - value dependencies between attributes
  - package invariants
  - attribute value constraints as part of pre- and post-conditions on actions
  - attribute value evolutions that trigger a notification.
- name-binding: the behavior associated to a name-binding describes behavioral consequences of the creation / deletion of a MOC instance using this name-binding. This behavior is not dependent of other behaviors within the MOC and thus, not subject to composition.

The standard does not define any specific semantics to each of these behavior templates neither it provides any formal notation to specify properly their contents. As these statements are generic they still allow any kind of behavior to be placed anywhere in the templates. To avoid this situation a clear statement was made on which behavior can be placed where in a GDMO template. These statements based, mainly on a self-containment principle provide some guidelines in order to restrict the behavior specification according to its location in a MOC specification.

The restrictions placed on this scattering allows to define a collect algorithm which is able to build one "behavior box" (set of behaviors associated to one MOC) that integrates all distributed behaviors and considers the conditional support of part of them for each MOC. This algorithm was developed for a rule-based approach to behavior specification (pre-/post-condition approach) in [Festor94]. It is based on assumptions build upon the restrictions mentioned previously. They describe behavior composition as follow :

- the behavior associated to a package results from the composition of its own defined behavior and the ones coming from its attributes, actions and notifications. This means that the behavior associated to an action which is refined within the package becomes the conjunction of these two behaviors.
- the behavior associated to a MOC results from the composition of the behaviors of all of its packages. This enables to build the full behavior of a MOC from the ones of its packages.

Note that, behavior associated with a conditional package is itself dependent on the presence or absence of the package within one instance. As we will see in the next section, the composition algorithm differs from the one used for inheritance and thus defines a semantics for packages which is not the same as the inheritance one. The above concept of behavior composition will serve as the basic building block for collecting behaviors in TIMS. However it must be extended with the definition of how we compose behavior bodies instead of pre-/post-conditions. This issue is discussed in more details in section 4.

3.2 Behavior composition through inheritance

Within the GDMO framework, one MOC inherits the features of its super-class(es). These inherited features are the packages, attributes, attribute groups, actions, notifications, parameters and in fact the behavior.

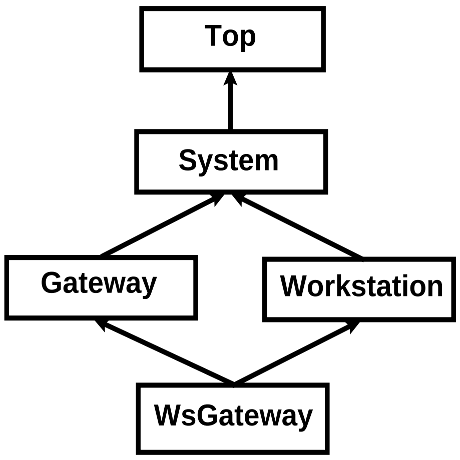


Figure 3 illustrates the multiple inheritance capacity provided by the OSI management information model through the definition of both a workstation and a gateway which inherit their properties from a system managed object. Then a subclass called WsGateway which models a workstation configured as a gateway, and thus, inherits from both Workstation and Gateway super-classes is defined.

Figure 3: A sample inheritance tree.

The inheritance mechanism defined in this approach supports multiple inheritance as illustrated above. The concept of strict inheritance as it was chosen within the OSI approach is essential towards the behavior composition since it restricts the ways in which behavior may be composed by inheritance.

If one construct is inherited from multiple super-classes, it is only present once in the super-class. Strict inheritance allows to define new features such as attributes, actions or notifications in a sub-class but restricts the treatment of behavior in the way that all features that hold in the super-class in a given situation must also hold in the sub-class under the same conditions. However, within the sub-class, effects of inherited operations, actions or notifications can be extended in the sub-class. These restrictions and the way they can be guaranteed in a specification can be easily expressed using the pre-/post-condition approach to describe the behavior [SM92]).

In order to combine the behavior bodies through inheritance, two mechanisms are possible. The first one consists of building the behavior body associated to a construct from all inherited parts and their specialization in the subclass. The second one consists in keeping only the deepest specified behavior for a construct (action, notification, . . .) and to execute only this part. This can be seen as behavior redefinition. How these approaches can be integrated within TIMS and combined with the pre-/post- assertion support is presented in the next section.

4 Application to TIMS

4.1 Preliminary Assumptions

According to the current available TIMS environment, and in order to avoid confusion towards the behavior distribution algorithm proposed, following assumptions are observed :

- a behavior is always considered atomic and isolated. This means that a behavior expressed through several expressions is either executed completely or not executed at all. Isolation means that intermediate states of the MIB resulting from partial execution of the behavior are not visible outside the scope of the behavior.
- the underlying model of behavior propagation and execution is considered to be synchronous, i.e. one can not have several behaviors executing concurrently. This restriction involves, no multiple / concurrent manager access and / or real resource change within the the MIB during the execution of any behavior propagation.

These two assumptions made it possible to propose a treatment of behavior distribution within TIMS. Extensions to an asynchronous model are out of the scope of the present paper, and are left for further study.

4.2 Principle

According to the two types of behaviors identified within TIMS and defined in section 2 (untriggered and triggered); how they relate to the above mentioned problems and how they can be composed taking into account the above mentioned constraints on distribution have to be defined.

Within the current implementation of the TIMS Behavior Language and its associated Behavior Propagation Engine, no difference is made between triggered and untriggered behavior specification forms. Behavior bodies in both forms are defined using scheme λ-expressions which express in an imperative way the operations to be applied to the MIB. Thus, behavior gathering / composition at the MOC level is reduced to the ordering / selection of scheme λ-expressions, which defines in the end part of an operational semantics for the formalism considered. Here, only the operational semantics defining precisely what is executed when a MOI (MO Instance) of a given MOC is solicited. A complete treatment would require to consider also the execution ordering for entire behavior propagations, i.e. not restricted to individual MOIs but rather on the overall MIB.

4.3 The scattering resolution algorithm

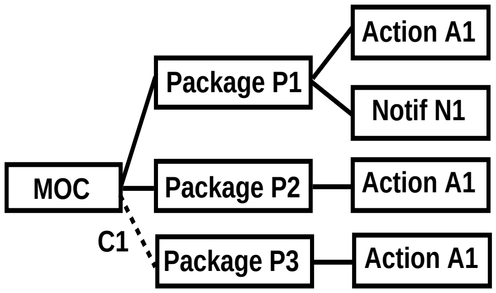
4.3.1 Composition approach

The **default** behavior associated to an action, operation or notification results from the sequenced execution of the behaviors associated to it from the most generic one to the most precise one. This means that, for an action, the piece of behavior that is executed first is the one associated to the action template, and the last one the action behavior associated to the package in which the action is referenced.

If an action is referenced in several packages then the execution order of the associated behavior in these packages is the one in which these packages are specified in the considered MOC. If a package is declared as conditional then the behavior is valid, if and only if, the package is present in the considered MOI.

Execution of the behaviors in the order they are specified may be ok in most cases. However, it may arises that in some particular contexts, this may reveal unsatisfactory, and one may need an additional degree of flexibility in order to specify its own ordering. To this end, it is allowed to define what is the exact expected behavior at the MOC level. If present this user-defined execution order would used.

Concerning the treatment of pre- and post-conditions, they would be composed as follow:



According to the composition algorithm outlined above, the default execution sequence associated to the behavior performance of action A_1 within the MOC is the following one:

$$A_1; P_1 : A_1; P_2 : A_1; if\ C_1\ then\ P_3 : A_1\ fi$$

For the notification N_1, the default behavior execution sequence will be:

$$N_1; P_1 : N_1$$

Figure 4: Behavior composition within one MOC.

- the pre-condition associated to a given behavior results from the logical "AND-ing" of all pre-conditions defined at the different levels of specification.
- the post-condition is built by "AND-ing" the different post-conditions associated to the specific behavior at the MOC, package, action or notification level.

4.3.2 Redefinition approach

The second approach that is possible for treating scattering in GDMO is the one that consists in associating a given location dependent priority to behaviors. This forces the specifier to redefine the behavior body, if necessary in a more high priority location of the MOC specification. Then, when a given behavior has to be executed, the Behavior Propagation Engine searches which expression is the most appropriate (towards priority) and executes only one λ-expression. The highest priority level is the MOC since it is the place where the specifier is supposed to have the complete overview of its object components. The MOC level is directly followed by the package one again followed by the lower priority behavior parts (action, notification, attribute and parameter). According to the example, one would require to have a behavior for A_1 at the MOC level since the action is defined several times at the same level and only the behavior body at the MOC level would be executed.

For the pre- and post-condition part of the behavior the treatment is the same as for the composition approach, i.e. pre- as well as post-conditions defined at the different levels of the MOC specification are logically "AND-ed". Thus the resulting behavior expression for the A_1 action will be:

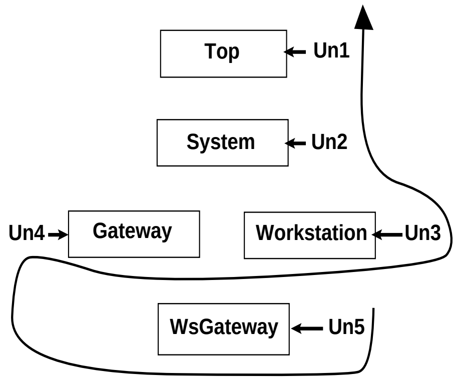
$$PRE_{MOC}(A_1)\ AND\ PRE_{P_1}(A_1)\ AND\ PRE_{P_2}(A_1)\ AND\ C_1 \Rightarrow PRE_{P_3}(A_1)\ AND\ PRE_{A_1}$$
$$\lambda_{MOC}(A1)body$$
$$POST_{MOC}(A_1)\ AND\ POST_{P_1}(A_1)\ AND\ POST_{P_2}(A_1)\ AND\ C_1 \Rightarrow POST_{P_3}(A_1)\ AND\ POST_{A_1}$$

This is the approach that is currently implemented within TIMS. It has the advantage of defining a clear behavior specification methodology. Building a new behavior from all different parts in an automated way is possible in TIMS since all expressions are available in a so-called behavior repository and the GDMO specifications are stored in the GDMO repository both accessible through a well-defined API.

4.4 The behavior inheritance algorithm

Within TIMS, the full behavior associated to a sub-class (behavior obtained after application of the scattering resolution algorithm) can be obtained through inheritance in the same way for both triggered and untriggered behaviors. To compose these behaviors, several strategies are possible.

These strategies can be classified in two categories, according to the possibility of redefinition.



Untriggered behavior Un_i associated to the sample inheritance tree will be executed by the default algorithm (bottom/up without redefinition) in the following order:

$$Un_5; Un_4; Un_3; Un_2; Un_1$$

If Un_5 would be a redefinition of the untriggered behavior associated with the class, then only Un_5 would be executed.

Figure 5: Example of untriggered behaviors and inheritance.

4.4.1 The composition approach

If no redefinition of behavior is allowed, i.e. behavior is considered to be specialized in the sub-class, all behaviors among the inheritance tree are checked for existence and executed in a given order. This order can be either bottom/up, or top/down:

1. For the bottom / up approach and in case of multiple inheritance (see figure 5), the order in which super-class behavior would be invoked could reflect the order in the `DERIVED FROM` clause of the sub-class definition.
2. For the top / down approach, one could start in the same order as proposed in the previous section within a MOC (package reference order) and then execute the behaviors from `Top`, then `Gateway` followed by `Workstation` and finally `WsGateway`.

In any other case, a default / fixed order of behavior execution has to be observed. This default sequencing of behaviors is not necessarily fixed, but the bottom / up one seems the most appropriate for both elegance and implementation simplicity.

4.4.2 The redefinition approach

If redefinition of behavior is allowed, the search of behavior associated to a given context follows a bottom-up approach, and, as soon as a behavior is found, it is executed and the search stops. In this case, only redefinition of behavior is possible. Note that redefinition does not affect the strict inheritance paradigm in the way that pre- and post-conditions are not treated like redefined behavior bodies, i.e. they are considered among the whole inheritance tree.

Pre- and post-conditions are build through inheritance in the following way:

- a pre-condition within a subclass results from the logical “OR-ing” of its sub-class specific expression and all inherited ones.
- a post-condition within a subclass results from the logical “AND-ing” of its sub-class specific expression and all inherited ones.

To summarize behavior treatment in the redefinition approach one can say that a body can be redefined in a sub-class, the pre-condition is automatically weakened and the post-condition strengthened in order to ensure strict inheritance. Thus for an action A defined in all classes of the sample inheritance tree except Top, the resulting behavior expression would be:

$$PRE_{WsGateway}(A)\ OR\ PRE_{Gateway}(A)\ OR\ PRE_{Workstation}(A)\ OR\ PRE_{System}(A)$$
$$\lambda_{WsGateway}(A) body$$
$$POST_{WsGateway}(A)\ AND\ POST_{Gateway}(A)\ AND\ POST_{Workstation}(A)\ AND\ POST_{System}(A)$$

This approach is the one which was selected within TIMS and the search order of behavior expressions through the inheritance tree is made in a bottom-up approach with the precedence level of the `DERIVED FROM` clause in case of multiple inheritance. These behavior is also built from informations gathered in both the behavior and GDMO repository.

5 Conclusion and Future Work

In this paper, some issues on the formalization of the behavior Managed Objects were presented. After a description of the TIMS projects and its goals, the principles of behavior scattering among several templates, and the way inheritance can be treated in behavior composition were outlined. According to these principles, their application within the current defined behavior language of TIMS have been presented.

The need to specify formally the behavior was clearly identified and expressed in [Kilov92b]. One of the main issue of the platform is the support of standardized Formal Description Techniques (FDTs) in order to allow MIB-specifiers to express the behaviors in an uniform way. To this end, some considerations are currently in progress towards FDT support in TIMS. The two investigated approaches are, on one hand, the Z notation [Spivey89, Rudkin91a, Kilov92a, Rudkin91b] for specifying assertions, which could be plugged around both triggered and untriggered behavior bodies; and on the other hand, the SDL'92 [IT92, BLR93, MMP93] for the imperative part used to specify behavior bodies. This work is extended with the support of the GRM for relationship specification and its impact on behavior distribution and propagation.

In the real world, one should always keep in mind that the network management is exercised in a non-cooperative environment, because of distribution, multiple management and interaction with uncontrolled real resources. For behavioral simulation purposes, this involves to be able to deal with multiple and concurrent behavior propagations. Relaxing the synchronous model assumption observed in this paper, and examining the consequences of supporting concurrent behavior propagations towards the behavior composition strategies described above seems to be a very relevant item of work, for future investigations.

References

[BLR93] Bartocci (A.), Larini (G.) et Romellini (C.). – A first attempt to combine GDMO and SDL techniques. *In : SDL'93: Using Objects*, éd. par Faergemand (O.) et Sarma (A.). pp. 333–345. – Elsevier Science Publishers B.V. (North-Holland).

[Ce91] Clinger (W.) et (editors) (J. Rees). – $Revised^4$ Report on the Algorithmic Language Scheme. *ACM Lisp Pointers*, vol. 4, n° 3, 1991. – Available through ftp on every scheme repository.

[CF93] Clemm (Alexander) et Festor (Olivier). – Behavior, Documentation, and Knowledge : An Approach of the Treatement of OSI–Behavior. *In : 4th International Workshop on Distributed System Operations and Management*.

[Cmis] Management Information Service Definition - Common Management Information Service Definition, ISO/IEC 9595, ITU X.710.

[Festor94] Festor (Olivier). – OSI Managed-Objects Development with LOBSTERS. *In : 5th International Workshop on Distributed System Operations and Management*.

[FHHS93] Fischer (Axel), Herpers (Martine), Holden (David) et Sievert (Stephan). – The DOMAINS Management Language. *In : Integrated Network Management III (C-12)*, éd. par Hegering (H.-G.) et Yemini (Y.). ©IFIP, pp. 181–192. – Elsevier Science Publishers B.V. (North-Holland).

[Gdmo] Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects, ISO/IEC 10165-4, ITU X.722.

[Grm94] ISO/IEC JTC 1/SC 21 – Information Technology – Open System Interconnection – Data Management and Open Distributed Processing – Structure of Management Information – Part 7 : General Relationship Model., mar 1994.

[IT92] ITU-T. – ITU-T Recommendation Z.100-92: Specification and Description Language, 1992.

[Kilov92a] Kilov (Haim). – Two approaches to information modeling standardization: Managed Objects for the OSI Management Information Service and Object Data Management Reference Model. *In : Computer Standards and Interfaces 14*. pp. 231–238. – Elsevier Science Publishers B.V. (North-Holland).

[Kilov92b] Kilov (Haim). – Understand, Specify, Reuse: precise specification of behavior and relationships. *In : Proceedings DSOM'92*. – Munich, Germany, 1992.

[MMP93] Mazaher (S.) et Moller-Pedersen (B.). – On the use of SDL-92 for the Specification of Behaviour in OSI Network Management Objects. *In : SDL'93: Using Objects*, éd. par Faergemand (O.) et Sarma (A.). pp. 317–331. – Elsevier Science Publishers B.V. (North-Holland).

[RBP$^+$91] Rumbaugh (James), Blaha (Michael), Premerlani (William), Eddy (Frederik) et Lorensen (William). – *Object-Oriented Modeling and Design*. – Englewood Cliffs, Prentice Hall, 1991.

[Rudkin91a] Rudkin (H.). – On the use of Z to specify OSI Managed-Objects behavior. *In : Proc. Int. Worshop on Open Distributed Processing*, éd. par Vuong (S.T.), pp. 401–420. – Berlin, Germany, 1991.

[Rudkin91b] Rudkin (H.). – On the use of Z to specify OSI Managed-Objects behaviour. *In : Proc. Int. Workshop on Open Distributed Processing*. – Berlin, Germany, 1991.

[Rumbaugh88] Rumbaugh (J.). – Controlling propagation of operations using attributes on relations. *In : Proc. ACM Conf. on Object-Oriented Programming Systems, Languages and Applications, ACM SIGPLAN Notices*, p. 285. – Published as Proc. ACM Conf. on Object-Oriented Programming Systems, Languages and Applications, ACM SIGPLAN Notices, volume 23, number 11.

[SM92] Simon (L.) et Marshall (S.). – Using VDM to Specify OSI Managed Objects. – 1992.

[Spivey89] Spivey (J.M.). – *The Z Notation*. – Prentice Hall, 1989.