TIMS : a TMN-based Information Model Simulator, Principles and Application to a Simple Case Study.

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

Rolf Eberhardt,
Swiss Telecom PTT – Research and Development[1].
Ostermundigenstrasse 93, 3000 Bern 29,
Switzerland. email: eberhard@vptt.ch

abstract

The main contribution of the paper is to set up an architecture for the behavior simulation and the rapid-prototyping of TMN systems. The underlying formal system is classically defined in terms of a formalism used to specify the simulated behaviors along with a simple operational semantic. A general outline of the so-called Behavior Language (BL) formalism is given, defining how the managed object behaviors are emulated when solicited from any interface. A hybrid approach, combining both imperative (triggered behaviors) and declarative (untriggered behaviors) specification in the context of relationships is used. Finally, since rapid-prototyping is the key feature of the targeted tool-set, the behavior integration and development environment is considered as an actual third dimension of the system. Thus, the different alternatives for behavior integration are evaluated, with respect to the well stated hypothesis and assumptions leading the project. This evaluation results as a non monolithic but rather integration approach : the so-called eXternal Behavior Integration (XBI). The concepts as well as the XBI approach are validated through a "not so" simple case study dealing with configuration management in the optical access network and service life-cycle provisioning.

Keywords : MO behavior formalization and simulation, rapid prototyping, TMN systems.

1 Introduction

In the context of OSI Network Management and TMN (Telecommunication Management Network), managed objects classes are specified using quite complex information models like GDMO (Guidelines for the Definition of Managed Objects) [Gdmo]. Moreover, MIBs (Management Information Bases) resulting from the instantiation of such information models and representing a particular configuration of the Telecommunication Network are then much more complex to handle and their implementation not available. Therefore, the need for tools to exercise and test such TMN systems in a simulated environment arises.

Such tools would enable one to test the validity of information models themselves, and even to set up a rapid–prototyping environment for TMN development prior to their introduction in the real TMN context. Here, the targeted tool should enable one to exercise an information model, representing a given network topology, through the execution of classical Common Management Information Service [Cmis] management operations. Another interesting purpose of such a tool would be to set up reference configurations of TMN systems, which 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 information, information loss. . .).

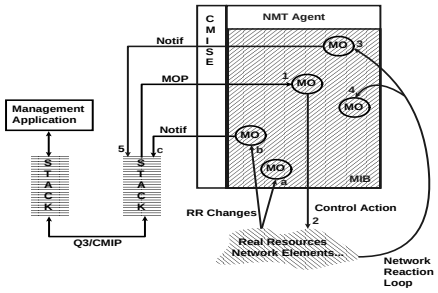
Plan of the paper : In section 1.1, the behavior simulation principle is stated, while in section 2 the hypothesis and assumptions leading the design and realization of the TIMS tool-set are given. Section 3 presents the behavior formalization approach adopted within TIMS. The two supported formalization paradigms (imperative and declarative) are presented in section 3.2, accompanied with an overview of the BL. The different alternatives for the realization of a rapid prototyping environment for TMN development is considered in section 4. The simple single order behavior propagation engine is described in section 5, which provides the operational semantic of the BL. In section 6, a case study is used to validate the current implementation of the TIMS tool-set : configuration management in the access network is exercised along with life-cycle service provisioning. Finally, section 7 concludes and gives some hints for further issues.

1.1 Behavior Simulation Principle

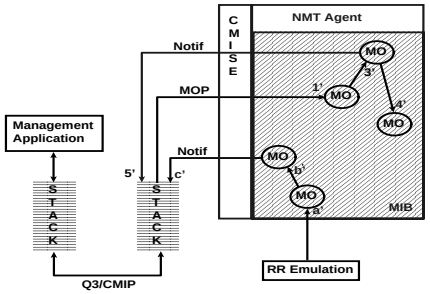
The two managed object interfaces, from which MOs are solicited : the Management OPeration (MOP) interface and the Real Resource (RR) interface are illustrated on figure 1(a) as their use can be observed in the real world.

Thus, Simulated Behaviors (SBs) originate either to management operations either, to Real Resource changes. Then, in the simulated environment, the network reaction loop as well as the real resources interactions have to be emulated in some way.

[1]This work was done in the context of the TIMS project. 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.



(a) Real World Managed Object Interfaces.



(b) Simulated Managed Object Interfaces.

Figure 1: Real World to Simulated Managed Object Interfaces.

The management control loop 1,2,3|4,5 of figure 1(a) is emulated in figure 1(b) as 1',3',4',5'. The real world network reaction has been emulated thanks to the introduction of direct effects 3' and 4'. In figure 1(a), a,b are interpreted[2] as two correlated real resources changes that are emulated on figure 1(b) thanks to a' and a subsequent effect b'.

This is the main objective of the present work. A possible transition from the real world context to a simulated one is shown on figure 1(b).

Behavior Simulation Principle Since at the network management level all the available information is encapsulated inside managed objects, 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 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. The execution of all these kind of effects may be subject to the fulfillment of some testing conditions on attribute values of some MOs.

2 Hypothesis and Assumptions

The formalization of simulated behaviors and their integration into a runtime environment define the two basic building blocks of any formal system, namely its syntax and its associated semantic. However as shown in figure 2, there is another major dimension to take into account : the development environment. In this section, the hypothesis and assumptions pertaining to all of these sub-problems, are briefly listed.

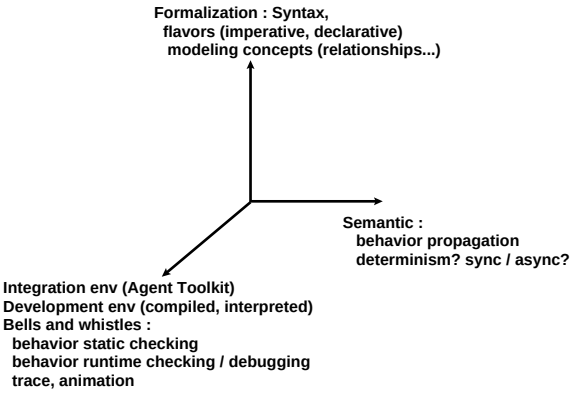


Figure 2 : Problem Dimensions.

Simplicity assumption : The formalism is assumed to be simple and cheap so that its resulting integration is expected to be easier. The first objective is to prove the underlying principles and usefulness of the behavioral simulation approach itself, in particular by providing executable specifications.

No timing constraint assumption : No duration can be associated to the execution of any effect. Only cause-effect transitions can be modeled inside behaviors.

Closeness to real world assumption suggests to use a generic agent toolkit software to support the runtime environment, since it is thanks to such software components that real world agents are usually built. Whatever the nature of a simulation is (e.g. performance or behavioral), its quality is always dependent on the modeling accuracy which is in direct relation with its closeness to the real world. In the context of TMN systems, the targeted simulated environment is present in the real world as network management agents, usually implemented thanks to generic agent toolkits softwares. Therefore a way not to go too far from the real world is to rely on a hosting agent toolkit for the behavior integration environments. Though such software components are quite complex to use (which is clearly against the simplicity assumption stated just before), they also provide a big amount of generic management facilities that should not be re-implemented in the simulated environment :

- CMIS service processing, multiple object selection, error handling,
- MO data store (e.g. the MIT), agent development API,
- Interoperability with any other management party (e.g. a management platform) is possible. This may reveal as a key point if reference configuration is targeted.
- Support of System Management Functions (SMFs) such as alarms, events, logs constitute very important mechanisms that should not be re-invented in any case for the simulated environment.

Flexible and comfortable development environment assumption : Obviously, in order to achieve rapid prototyping of any TMN information model, flexibility and comfortable development environment is advocated. However, agent toolkits onto which the TIMS platform have to rely in some way, provide tedious development environments usually intended for skilled MIB implementors. Thus, flexibility and closeness to real world assumptions are clearly contradictory, and a fine compromise has to be determined. This is discussed in section 4.

3 Behavior Formalization

This section is intended to emphasize the need for behavior formalization in general [Kilov92, Kilov et al.94], and in particular the problem of MO behavior formalization [Clemm et al.93]. The particular position of simulated behaviors is also discussed. Moreover, the need of a general relationship facility such as the one provided by the General Relationship Model is, as in [Clemm et al.93], also advocated; to model any hidden dependency between MOs, which would be required to formalize properly a given simulated behavior.

3.1 Relationship-based Behavior Formalization

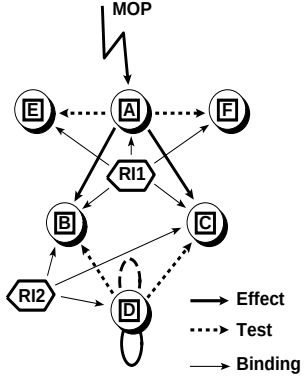
Behaviors represent general dependencies between MOs, e.g. how one MO influences other MOs, or general properties valid among well defined MO sets. It is natural to think about such dependencies in the context of relationships. Relationships provide the means to specify cleanly the different neighboring contexts in the scope of which behaviors should manifest themselves. Relationship classes give, in their turn, the means to define the different neighboring flavors of interest for each MO instance.

3.2 Behavior Flavors

Within the relationship framework, it is very straightforward to formalize how a MO change should affect its neighbors. This corresponds to triggered behaviors, where a triggering context determines in an imperative fashion when a behavior should be executed. On the other hand, being able to specify general dependencies among the MOs participating in a given relationship without any reference to any triggering context is also very interesting. The resulting so-called untriggered behaviors are thus specified in a purely declarative fashion.

As claimed in section 1.1 simulated behaviors originate either in Management Operations (MOPs) either in Real Resource Changes (RRCs). Then if defined, their simulated behavior is processed which in its turn may incur other MOPs or RRCs to be exercised on other MOs and as a consequence a complete behavior propagation chain, that terminates when no more propagation is firable. On the formalization point of view, two paradigms are of interest :

1. An **imperative** approach (see figure 3, behavior 1), enables one to specify behaviors directly associated to its triggering context, that is a given MOP or a RR change. From now on, such behaviors are referenced as **triggered behaviors**.
2. A **declarative** approach (see figure 3, behavior 2), is also useful in the case one wishes to specify some property that has to be maintained among a given managed object set, without being constrained to take into account any triggering context. Such behaviors are called **untriggered behaviors**.



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.
3. **Relationship context** : As already noticed, in any cases the behaviors are specified in the context of a relationship. That means that for the triggered behavior a relationship instance associating {**A, B, C, E, F**} has to exist. In the same way, for the untriggered behavior, a relationship instance associating {**B, C, D**} has to exist.

Figure 3: Triggered and Untriggered Behaviors, in the context of relationships.

The use of one paradigm rather than the other one is dependent, on one hand, on the **semantic** of the behavior itself, and on the other hand, on the behavior **specifier**. A behavior specifier who usually programs with a procedural language is naturally brought to an imperative formalization, even if in some cases a declarative form is more synthetic and appropriate.

3.3 General Form of Behaviors

The general form of a simulated behavior is composed of :

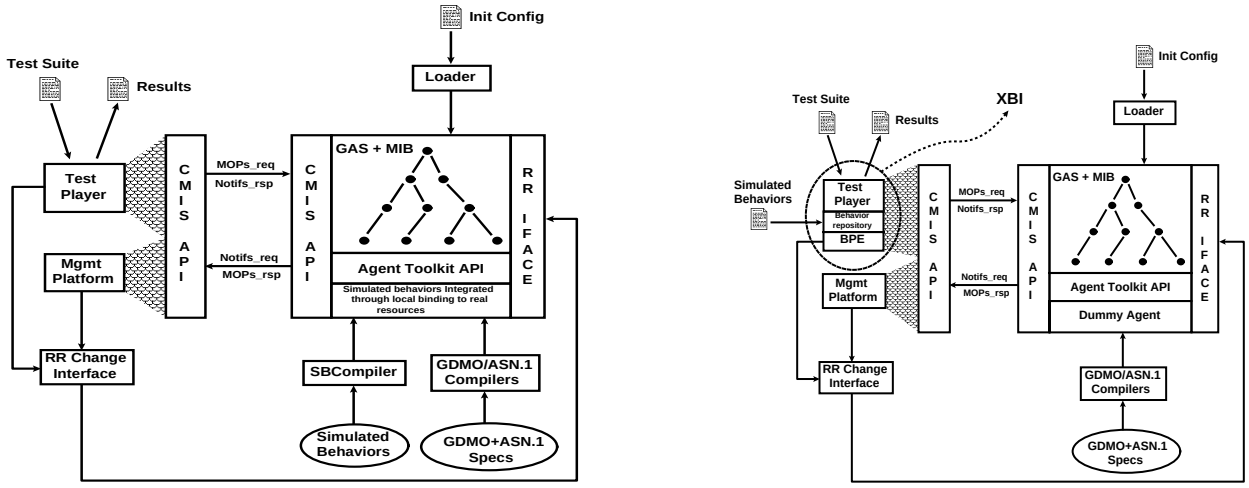
1. **scoping context** : the relationship whose instances are concerned by the behavior.
2. **triggering context** : present only for triggered behaviors, specifies precisely the event (management operation or real resource change) that triggers the execution of the behavior (attribute change, object creation, object deletion, action). Since the scoping context is a relationship, it is also necessary to specify to which participant role this event is applied.
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 *Scheme* [Clinger et al.91] code[3]. There is no a priori structure imposed on it. Tests and effects are intended to be used in this body, reflecting the dependencies between the MOs participating in the relationship instance. 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 a reasonable choice. This is also the approach taken in the DOMAINS project [Fischer et al.93], which uses Eiffel construct to formalize behavior bodies. Though not mandatory, chosing the programming language of the targeted runtime environment may reveal very interesting in order to provide executable specifications and by this means facilitating the integration of behaviors in the runtime environment.
5. **post-condition** : testing condition upon the MIB state, to be satisfied at the completion of the execution of the behavior body.

4 Behavior Integration

In this section, the different alternatives for the realization of a rapid prototyping environment for TMN development is considered. Basically, these alternatives depend on the extent the targeted tool-set is intended to rely on the generic software packages used to build network management agent, the so-called “agent toolkits”. The considered alternatives are :

1. The Internal (to agent toolkit) Behavior Integration (IBI). Here behavior are integrated in the same way a normal agent would be implemented. That is thanks to the agent toolkit API where the usual local mappings towards the real resources are realized.
2. The eXternal (to agent toolkit) Behavior Integration (XBI), uses an agent toolkit for all the nice features provided that we do not want to re-develop (e.g. CMIS processing, MO data store, SMFs...). The behaviors themselves are integrated externally, in an environment of our choice, which inter-operates with some kind of dummy agent through the standardized CMIS interface.

[3]The choice of the *Scheme* programming language is argued in section 4.



(a) Internal Behavior Integration.

(b) eXternal Behavior Integration.

Figure 4: IBI vs. XBI.

	Internal BI	eXternal BI
Distance from real world	(+) Agent with SBs is a normal agent	(-) Dummy agent
Agent toolkit dependence	(-) Total, GAS API support	(+) Only a fixed code rriface lib
Rapid prototyping	(-) Systematic recompilation, relinking	(+) Interpretation possible, modification involves a simple reload
Behavior debugging	(-) Tedious, with normal debugger	(+) Straightforward
Coupling with management	(+) Immediate	(-) Involves a coupling module, decomposition, recomposition (not a big problem)

Table 1: IBI vs. XBI, pros. and cons.

3. The Without (agent toolkit) Behavior Integration (WBI), can rely for example on a FDT and its development environment. This provides a standardized specification framework. The problems with this approach are : (i) it involves to re-implement, as soon as needed, all the nice features provided by generic agent toolkit software packages; (ii) the closeness to real world assumption is no more observed, in particular all the GDMO, ASN.1 and CMIS implementation details which often cause many problems may probably be abstracted away with such an approach. Though attractive for its independence from external software packages, the arguments above makes the WBI approach not adapted to the original objectives stated in section 2. That is the reason why, this item is a priori rejected and is not to be discussed further below.

Thus, the choice has to be done between the IBI and the XBI. In table 1, are synthesized IBI versus XBI pros. and cons.

Synthesis From this comparison study the more reasonable alternative seems to be the eXternal Behavior Integration. In effect, the only drawbacks are : (i) the distance from real world agent systems, since the agent system is still used, but only in a dummy fashion; and (ii) the fixed pieces of code that need to be provided in order to realize the behavior integration separately. This includes mainly a real resource emulation, which is implemented thanks to an action onto a `RRIface` MO and enabling to avoid GDMO defined access rules so that simulation of real resource evolution is made possible (update of a read-only attribute, creation of a non-creatable MO. . .). This makes the XBI approach clearly a good compromise taking all the required features from the agent system, without being restrained to the use of a particular one, since inter-operability is ensured thanks to the use of CMIS/P based interactions between the XBI and the dummy agent. Thus, onecan even imagine to build a tool-set

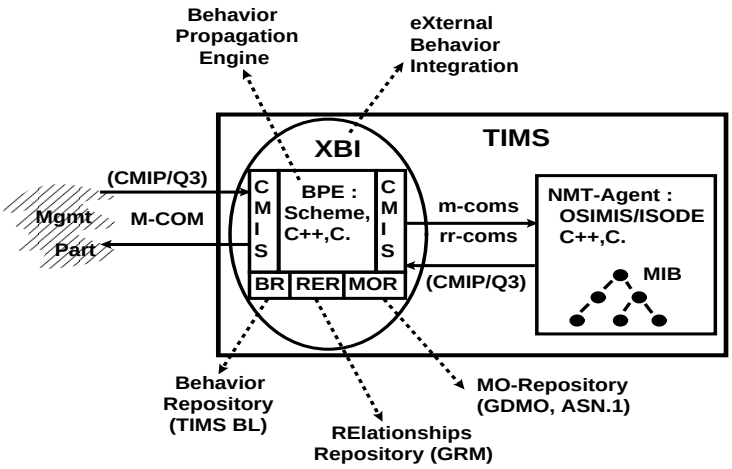


Figure 5: eXternal Behavior Integration (XBI).

running with different agent systems to be sure they behave in a proper fashion. Finally, the key point in favor of the XBI is that separate behavior integration enables to chose the environment the more adapted to the formalization and development of simulated behaviors, in an easy and quick fashion. Since the best context in order to achieve rapid-prototyping, is to use an interpreted framework, it was natural to make this choice for the TIMS tool-set. Therefore, the current implementation of the XBI is based on the interpreted language *Scheme* [Clinger et al.91]. *Scheme*-like programming frameworks provide a powerful functional programming paradigm along with a simple syntax which is extensible through high level macros. The macro facility is used to implement the general form of behaviors given in section 3.3 and illustrated in section 6. Note that, that relying on a hosting language such as *scheme* enables to reuse all the nice builtin programming constructs (and not to re-invent them) like a variable notation, basic as well as composite data structures, basic constructs for control flow. . . . The overall design of the Behavior Language is quite simple, it is defined by the unique *define-behavior* construct, into which usual *scheming* is encapsulated within behavior bodies (pre / body and post). The *scheming* environment is extended with the APIs giving access to the different entities of the currently supported information models, i.e. GDMO, ASN.1 and the GRM [Grm]. The fact that the BL is supported by a programming environment, is very important in order to ensure executability of the resulting BL specifications.

On the agent toolkit side, the Generic Management System (GMS) of the OSIMIS [Pavlou et al.] /ISODE platform is used. From this environment is also used the provided CMIS Services API (MSAP), this API is integrated into the XBI through the library extension facility furnished by the *Scheme* programming environment under use [Lord95]. Thus, the XBI is doted with a full CMIS API, and CMIP processing capability.

The behavior simulation and prototyping environment is shown on figure 5. XBI modules are the various repositories of interest with respect to the information model under use, along with the Behavior Propagation Engine (BPE) which can be seen as the heart of the XBI. Moreover, the BPE defines the operational semantic of the Behavior Language (BL), which is not an obvious task since the BL is a hybrid formalism enabling to combine both imperative and declarative constructs. The BPE basically works as a forward chaining inference engine : it is initially solicited through the XBI interface to either invoke a management operation or to signal a change related to an underlying resource; then it propagates the initial solicitation according to the behaviors in effect, until saturation, that is until the system reaches a new steady state. At any given time, a snapshot of the system is defined by the set of managed objects and corresponding attribute values (the so-called Management Information Tree). The current implementation, is limited to a single order propagation. This simple BPE, though not able to deal with all complexity introduced by the inherent non determinism of BL specifications, was powerful enough to run the “not so” simple case study presented in section 6.

5 Single Order Semantic

Algorithm 1 bpe:tbp(msg) :

```
1   ribks ← fetch-behaviors(msg)
2   bpe:execute(msg)
3   ∀ <ri, block> ∈ ribks :
4       if triggered?(beh(block))then
5           eval(body(block), ri, msg)
6           check-post(post(block), ri, msg)
7       else
8           eval(body(block), ri)
9           check-post(post(block), ri)
10      end if
```

This algorithm, (SO-BPE), defines a low level semantic for the BL, in the sense that propagation order choices made in cases of non determinism are hard-coded inside the BPE algorithm itself. The entry point to the BPE machinery is the "Triggered Behavior Propagation" (`bpe:tbp`) function. This is normal since the BPE is always originally solicited through an explicit trigger coming from either the management or the real resource interface. The message structure passed (`msg`) to this function represents any operation along with its associated parameters valid in the context of the information model utilized. Typically, operations include CMIS, real resource interface and relationship services. Parameters can be a targeted MO, an attribute, a value (ASN.1). . . . The overall structure of the BPE is quite simple, the candidate behaviors for further execution are fetched according to the message sent; then their body are actually executed and their post checked. Obviously, in any circumstance where a new message is sent to a MO during the evaluation of a behavior body (see instruction 5), the `bpe:tbp` is recursively called with the new message sent.

The `fetch-behaviors` function can be detailed as follows :

Algorithm 2 fetch-behaviors(msg) :

```
1       { <ri, block>, ri ∈ parties(moi(msg)) ∧                  ▷ party(moi) = <role, ri>
2           ∃ beh ∈ br, block ∈ blocks(beh) :                    ▷ br : behavior repository
3           rel(ri) = rel(ctx(beh)) ∧ {                          ▷ blocks(beh) : pre-body-post behavior blocks
4           { untriggered?(beh) ∧ firable?(block, msg) ∧ check-pre(pre(block), ri) } ∨
5           { triggered?(beh) ∧ check-pre(pre(block), ri, msg) }}
```

One should notice that for the invocation of triggered behavior bodies, the triggering message sent is also provided for the actual execution. This enables, for instance, to test the CMIS parameters given to the original management operation. For a `M_Set`, the new value can be tested in the pre-condition to specify behaviors associated to state transitions on an attribute.

For untriggered behaviors, since there is no explicit trigger that enables to determine when to execute them, this is supported internally by the system thanks to the function `firable?`. The principle is to see if the operation invoked on a given object in a given role requires execution of the untriggered behavior, i.e. in order to maintain things consistent. For instance, if a variable is modified, and if the untriggered behavior makes in its body a `get` on it, the behavior is considered as firable. Obviously, this is an approximation, in some case this execution may not be necessary. In some cases, the behavior specifier may have to take into account this feature. However, a mechanism has to be provided to effectively propagate untriggered behaviors in the system.

Enhanced Semantics Note well however, that if we assume that the "∀" construct of instruction 3 in algorithm 1 goes through the matched behaviors to be propagated in a random order, then by playing the behavior simulation several times one can get some valuable insights about potential non determinism in the behavior specification. A Randomized Single Order operational semantic is thus defined by the algorithm that may be called the RSO-BPE algorithm. In any case the full single order semantic is still useful because it is the one that is used in the early stages of the rapid-prototyping process, where there is not yet any need for a full fledged behavior analysis semantic, and which would even be painful at this stage of development. More enhanced semantics for the BPE are discussed in [Sidou95]. They define dynamic behavior analysis mechanisms able to emphasize the non determinism in the system. A possible application for management policy conflict detection is also described.

6 The Simple Case Study

A real-life TMN environment is considered to verify the TIMS design decisions; its application in the Optical Access Network Management (Fiber-in-the-loop concept) is currently in the standardization process in ETSI TM2 [De tm2209]. This choice proved to be a wise one; (1) the team has learn more about the TMN-specific problems of Information Modeling and (2) immediate feedback of the results into ETSI was possible.

Configuration Management in the access network (AN : equipment, network, services) and life-cycle provisioning is considered and especially the interface between Service Providers (SP), and Network Operators (NO), which permits the SP to access, provision and manage services using the AN. Note that, the transport technology is of no importance to the service provisioning process. In the AN area, several SP will be offering services, even competing ones, across the same AN provisioned by one NO. It is possible and likely that one SP will have relationships to multiple NOs. To permit provisioning, two types of

resources are necessary. Logical resources model the connectivity across the AN and resources such as bandwidth. Physical resources represent the equipment necessary for service invocation. SP and NO retain the same resources. These are :

Transmission Layer logical resources :

- SNC (SubNetworkConnection): represents connectivity and bandwidth across the AN.
- nTTP (networkTrailTerminationPoint): terminating a SNC. This MO represents the endpoint of a service.
- nCTP (networkConnectionTerminationPoint): internal structure of the nTTP showing the time-slot allocation.
- serviceResource: represents the transport service as a whole and correlates logical, physical resources to a required service.

Physical resources :

- equipment: represents the equipment housing aka a slot, a service unit such as a card . . . .
- subNetwork: represents the total bandwidth allocated to a SP across the AN.

Both logical resources can be provisioned independently of physical resources. However, it is only possible to activate a service when both logical and physical resources have been assigned.

Organizational resources :

- node represents the sum of all equipments in one location independent of ownership.
- administrativeDomain is an entity containing all resources allocated to a single SP.

Test suites are composed with more basic building blocs or scenarios. The model covers management functional areas necessary for service creation; these involve :

1. installation / removal of equipment (physical resources).
2. configuration of logical resources.
3. path setup / activation.
4. service activation (this is what is illustrated in the example below).

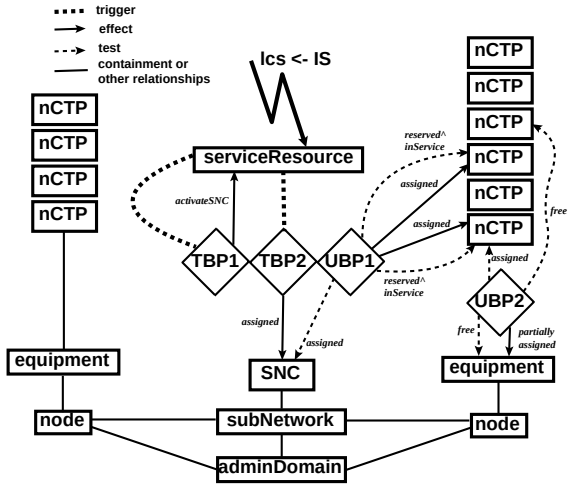


Figure 6 : Example of configuration for the simple case study.

A typical configuration example is shown in figure 6, where a SP constructs a service from following components : nCTP (representing the logical capacity of an equipment), SNC (containing the transport capacity across the network) and the service resource.
Diamonds in the figure 6 represent behaviors within the context of relationships. The key relationship used for the activation of a service resource is the "service-mgmt" relationship between the nCTPs, the service resource and the sub-network connection.
Most of the Behavior propagation presented below takes place in the context of this relationship, i.e. among its participant MOs. This propagation is initiated by the activation of a service resource. The propagation then flows thanks to untriggered propagation to the "equipment-resource" relationship between nCTPs and an equipment.

Example : This example illustrates the activation of a service ressource (see figure 6). The actual BL specifications of the concerned behaviors is provided below. The *define-behavior* construct is shown along with the primitives for the MO and relationship access APIs (`mo-send` and `ri-send` primitives). The remaining parts are pure *Scheming* (lambda, and, or, equal?, cond, if, for-each. . .) or functions calls from the *Scheme* library used (`some, every`. . .) [Eigenschink et al.94].
The behavior propagation can be decomposed in the following steps :

- TPB1: Setting attribute "LifeCycleState" of MO "serviceResource" from "planned" to "inService" results as the invocation of an action "activateSNC" on this service resource itself.
- TPB2: The body of the action "activateSNC" makes the "SNC" assigned.
- UPB1: This untriggered behavior becomes firable, because of the SNC became assigned, the effect is to assign the related NCTPs
- UPB2: Finally, because there are both assigned and free "nCTPs", the untriggered behavior defined on the "equipment-resource" relationship can be fired with the effect of setting the equipment partially assigned.

```
(define-behavior   ;;; TBP1
  (scope rel "servic-mgmt")
  (trigger (role "srv")
           (interface mgmt)
           (op update "lifeCycleState"))
  (pre ...)
  (body (mo-send action "activateSNC"
                 (interface mgmt)
                 (mode confirmed)
                 (moi (ri-send binding-value
                               (role "srv")))
                 (argument (ri-send binding-value
                                    (role "snc")))))
  (post ...))

(define-behavior    ;;; UBP1
  (scope rel "service-mgmt")
  (trigger)
  ((pre)
   (body ...
     (for-each
       (lambda (nctp-elem)
         (if (and
              (equal? "reserved"
                (mo-send get
                   (moi nctp-elem)
                   (role "nctp")
                   (attribute "assignmentState")))
              (equal? "inService"
                (mo-send get
                   (moi nctp-elem)
                   (role "nctp")
                   (attribute "lifeCycleState")))
              (equal? "assigned"
                (mo-send get
                   (moi (ri-send binding-value
                                 (role "snc")))
                   (role "snc")
                   (attribute "assignmentState"))))
             (mo-send set
                     (interface rr)
                     (mode confirmed)
                     (moi nctp-elem)
                     (attribute "assignmentState")
                     (value "assigned"))))
         (ri-send binding-values (role "nctp"))) ...)
  (post)))
```

```
(define-behavior    ;;; TBP2
  (scope rel "service-mgmt")
  (trigger (role "srv")
           (interface mgmt)
           (op action "activateSNC"))
  (pre)
  (body (mo-send set
                 (interface rr)
                 (mode confirmed)
                 (moi (param argument)))
                 (attribute "assignmentState")
                 (value "assigned"))
  (post))

(define-behavior         ;;; UBP2
  (scope rel "equipment-resource")
  (trigger)
  (pre)
  (body ...
    (if
     (and
      (equal? "free"
       (mo-send get
          (moi (ri-send binding-value
                        (role "physical")))
          (role "physical")
          (attribute "assignmentState")))
      (some (lambda (nctp)
              (equal? "free"
                (mo-send get
                   (moi nctp)
                   (role "logical")
                   (attribute "assignmentState"))))
            (ri-send binding-values (role "logical")))
      (some (lambda (nctp)
              (equal? "assigned"
                (mo-send get
                   (moi nctp)
                   (role "logical")
                   (attribute "assignmentState"))))
            (ri-send binding-values (role "logical"))))
     (mo-send set
       (interface rr)
       (mode confirmed)
       (moi (ri-send binding-value (role "physical")))
       (attribute "assignmentState")
       (value "partiallyAssigned"))) ...)
  (post)))
```

7 Conclusion and Further Issues

The simple case study has been very useful to validate the design choices made for the described behavior formalization and simulation environment. The eXternal Behavior Integration (XBI) architecture has been designed. The XBI enables, on one hand, to reuse and in this way to avoid re-inventing / implementing all the basic features of a generic agent toolkit software. On the other hand, all the remaining pure behavioral parts can be exercised in the environment the more interesting for the purpose of rapid-prototyping, i.e. an interpreted support enabling maximal flexibility in terms of dynamic evaluation.

From the formalization point of view, the design resulted as a Behavior Language (BL), combining both imperative (triggered behaviors) and declarative (untriggered behaviors). In the current version of the BL, behaviors have to be specified within the scope of a relationship. Relationships provide the means to specify cleanly the different neighboring contexts in the scope of which behaviors should manifest themselves. Within this framework, it is very straightforward to formalize how a MO change should affect its neighbors, and more general inter-MO dependencies.

The key design of the BL was to embed it in a programming environment, to ensure executability. Thus, the notation is deliberately based on the *Scheme* language, because it is small, simple (syntax), very powerful, extensible through high and low level macros and interpreted which is ideal for rapid prototyping.

Thus, the BL constructs are defined as *Scheme* syntax extension thanks to the available high level macro facility.

A simple single order operational semantic has been defined and implemented for the BL. The resulting Behavior Propagation Engine (SO-BPE) has been tested onto the “not-so” simple case study.

To caricature, TIMS can simply be viewed as the very simple concept of a *Scheme* environment just enriched on one hand with a syntactic BL sugar, and on the other hand, with the needed APIs for object and relationship accesses. The whole stuff is then animated by a “not so” complicated simulation engine (BPE). In fact, this simplicity can also be viewed as a feature rather than a bug. In effect, simplicity ensures consistency and provides a non monolithic and open environment. For instance, high level *Scheme* macro enables to concentrate on the language concepts rather than having to bother with implementation details

about lexical analysis and parsing. This facility will be used to integrate in the BL enhanced relationship support. Semantical aspects can also be improved by integrating other BPE algorithms doted with behavior analysis capabilities. This would enable to handle non determinism, so that specification ambiguities could be raised and concurrency supported to some extent. The XBI being perfectly decoupled from the agent system, one can think to it as a generic behavior simulation kernel to which could be plugged APIs for other information models (e.g. OMG / CORBA). This should be possible as soon as all remains in the scope of objects oriented relationship models.

Finally, TIMS is much more a behavior simulation laboratory extensible and customizable on both formalization paradigms, semantics and environmental issues.

References

[Clemm et al.93] 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*.

[Clinger et al.91] Clinger (W.) et (editors) (J. Rees). – $Revised^4$ Report on the Algorithmic Language Scheme. *ACM Lisp Pointers*, vol. 4 (3), 1991. – Available through ftp on every scheme repository.

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

[De tm2209] Transmission and Multiplexing : Operations and Maintenance of Optical Access Networks, ETSI TM2, DE-TM2209, STC Draft 08d, 16.9.1994.

[Eigenschink et al.94] Eigenschink (T.R.), Love (D.) et Jaffer (A.). – SLIB: The Portable Scheme Library, 1994.

[Fischer et al.93] 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.

[Grm] ISO/IEC JTC 1/SC 21, ITU X.725 – Information Technology – Open System Interconnection – Data Management and Open Distributed Processing – Structure of Management Information – Part 7 : General Relationship Model.

[Kilov et al.94] Kilov (H.) et Ross (J.). – *Information Modeling: An Object-Oriented Approach*. – Prentice Hall, 1994, *Object-Oriented Series*.

[Kilov92] Kilov (Haim). – From OSI Systems Management to an Interoperable Object Model: Behavioural Specification of (Generic) Relationships. *In : Proceedings 3d Telecommunications Information Networking Architecture Workshop (TINA 92)*. – Narita, Japan, January 21-23 1992.

[Lord95] Lord (Thomas). – The Guile Architecture for Ubiquitous Computing. – to appear in the proceedings of the Usenix Tcl / Tk Workshop'95, 1995. available at http://www.cygnus.com/library/ctr/guile.html.

[Pavlou et al.] Pavlou (G.), Knight (Graham) et Walton (Simon). – Experience of Implementing OSI Management Facilities. Department of Computer Science, University College of London, available with the OSIMIS distribution.

[Sidou95] Sidou (Dominique). – Behavior Simulation and Analysis Techniques for Management Action Policies. *In : to appear in SMDS'95 Workshop : Services for the Management of Distributed Systems*. – Karlsruhe, September 20–21 1995.