Guidelines for the Specification of Managed Object Behaviors with TIMS

Sandro Mazziotta, Dominique Sidou

Institut Eurecom, 2229 rte des Cretes, BP.193, 06904 Sophia Antipolis CEDEX FRANCE.
email : {mazziott | sidou}@eurecom.fr

1 Introduction

The deficiencies of TMN Information Models in regard to behavior aspects have justified the need 1/ for better behavior specifications and 2/ for validation frameworks and tools. In effect, GDMO/ASN.1 standards used to describe Information Models are not satisfactory because :

- behaviors are given in unstructured prose.
- behaviors are limited within Managed Object Class (MOC) boundaries.

This results as unclear, often ambiguous and incomplete behavior specifications which might lead to non-interworking TMN-systems components (e.g. managers, agents). TIMS[1] approach tries to define a framework and supporting tools to achieve better behavior specifications for TMN systems. For that purpose, a Behavior Language (BL) and its operational semantic has been designed. They are both the main consideration of this paper. The initial Static Information Model (GDMO/ASN.1) extended by BL specifications are then plugged in a tool: the "TIMS toolkit" which provides a prototype of the TMN system. For more information on the toolkit, the reader should refer to [1].

The problem of ambiguous formalisation in prose is solved by providing a formal specification of Managed Objects behaviors. There are currently several formalisms proposed by the Network Management bodies. Where TIMS approach is particular, is in the way to solve the second problem (i.e. limitations within MOC boundaries).

This problem is solved through the use of a relationship model. That means that with such model there exist relationships access facilities for a given Managed Object. The specifier should not consider its TMN system (to be emulated) as a tree (MIB) but he should rather consider a graph where there exist two kinds of nodes, Managed Objects and Relationship Objects. Rather than proposing a notation for relationship modelling, it has been decided to support the General Relationship Model (GRM).

The main objective of TIMS is to provide a rapid prototype of the TMN system. The chosen approach of the project is to simulate the system thanks to executable behaviors of object reacting to pertinent test-cases.

---

1. This work was done in the context of the TIMS project. TIMS stands for TMN-based Informa tion Model Simulator. This project is a collaboration between Eur ecom Institute and Swiss Te lecom PTT. It issupported by Swiss Telecom PTT, project F E-288.

Another interesting feature of the BL is the use of assertions (pre and post-conditions). This provides a simple and efficient way to ensure the correctness of the executions. Assertions contribute to better specification. That is why they are clearly separated from the pure simulation aspects.

This paper explains precisely how to specify behaviors with BL. The envisionned result is to provide to the reader guidelines for the specification of Managed Objects behaviors with TIMS. The next section of the paper presents the TIMS behavior formalisation concepts. First, TIMS behaviors are explained. Then, combined with simple examples, the BL and its operational semantic are explained. Rather than enumerating the different features, a tutorial approach is adopted.

2 TIMS Behavior Formalisation Concepts

2.1 Specifying Simple Behaviors

Basically a behavior can be viewed as a piece of code that is executed when a given message is received by the system. A message is any operation that occurs at an interface of an object. For the following of the explanation this particular message is identified as the trigger. In the piece of code describing a behavior new messages can be sent to the system which can be in their turn triggers for other defined behaviors.

All behaviors that will be presented in this section are first presented in informal prose. Then, the second step is to translate this prose to a pseudo formal notation based on relationships where the participating roles are identified. The last step that will be presented in section is to translate behaviors from the pseudo formal notation to BL executable behaviors. A trivial example of behavior is: simple-beh is “If object A receives operation Op, then operation Op’ is executed on object B”.

In order to support behavior simple-beh, one needs to define a relationship class “Rel” with two roles “r1” and “r2” that participate in this relationship following Figure 1. For that purpose, the prose should be decomposed in two parts, the test part and the effect part. There is an implicit causality relationship between the test part and the effect part. This causality is reflected in TIMS BL by expressing the test and effect parts in two different clauses: the guard and the body. The test condition that enables the execution of the effect part is put in the guard clause while the effect part in put in the body part.

Following the previous conventions behavior can be now expressed:

<LABEL : “beh-simple”,
GUARD : “object A receives operation Op”,
BODY : “operation Op’ is executed on object B”>

If the relationship model is applied, the behavior becomes:

<LABEL : “beh-simple”,
REL : “Rel”,
GUARD : “role r1 receives operation Op”,
BODY : “operation Op’ is executed on role r2”>

An other trivial example of behavior is: other-simple-beh is “if object A reaches state S, then operation Op’ is executed on object B”. The same method is applied on the behavior other-simple-beh. The difference between these two behaviors lies only in the test condition. In the second case, only the effect on object A is used as trigger for the behavior. The original cause is out of scope. That means in other words that it is not important to know the trigger that made the object A to become in the state S.

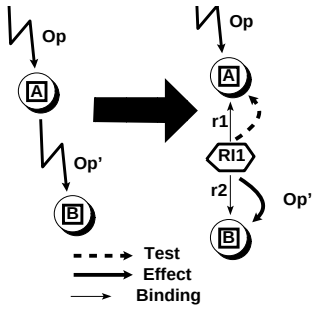

Figure 1: simple behavior

A more formal notation of behavior other-simple-beh is:

<LABEL : “other-simple-beh”,
REL : “Rel”,
GUARD : “role r1 reaches state S”,
BODY : “operation Op’ is executed on role r2”>

2.2 Specifying Behaviors with Assertions

An interesting feature of the BL is to provide a way to verify the correctness of behavior executions. This is done thanks to assertions checked before (pre) and after (post) the execution of the behavior bodies. Note that assertions are totally independent of the notion of guard since when an assertion is not met (evaluate to false), the execution is stopped and an exception is raised. When a guard is met, this enables the execution of the behavior.

A third trivial example based on exemple on the figure called simple-with-assert is: “If object A receives operation Op, verify that object A reaches state S , then operation Op’ is executed on object B, verify that now object B reaches state S’”. It is added on the pseudo-formal notation, two new clauses a pre-condition and a post-condition. A pseudo formal notation of behavior simple-with-assert is:

<LABEL : “simple-with-assert”,
REL : “Rel”,
GUARD : “role r1 receives operation Op”,
PRE : “role r1 reaches state S”,
BODY : “operation Op’ is executed on role r2”,
POST : “role r2 reaches state S’”>

The guard, the pre and post-condition are boolean expression. There is no limitation on the condition to check and the number of conditions...

2.3 Execution Model

The Behavior Propagation Engine (BPE) algorithm defines the operational sematics of BL. BPE defines the execution model which is the purpose of this section. For each message incoming to the system, behaviors are considered for evaluation. Depending on their guard, some behaviors are fetched (selected) and are executed. Because a behavior body can send in its turn other messages, new behaviors may be executed. This results as a propagation of behaviors which terminates when the process saturates, i.e. at a new steady state of the system where no more behavior can be fetched. Figure 2 gives a schema of the BPE algoritm in action.

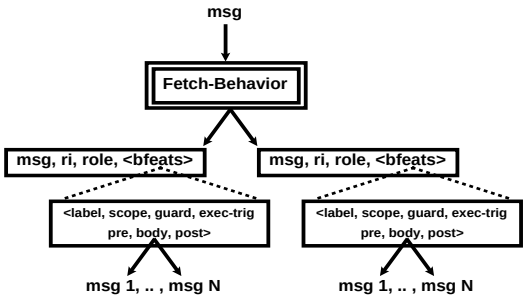

Figure 2: BPE Algorithm in action

An incoming message is send to the Fetch-Behavior algoritm. This produces two behaviors execution contexts (bec). A bec is composed of a message (trigger), a relationship instance, a role and the behavior fetched itself (the behavior contains all the info necessary to execute each behavior i.e. guard, pre, body and post).

2.4 Nondeterminism in Behaviors Specification

Nondeterminism is an inherent property of Distributed and Reactive Systems (TMN-systems are perfect examples). This reflects simply the fact that when several messages are enabled to occur they may execute concurrently (concurrency), sequentially in an arbitrary order (unordering), one may be chosen arbitrarily (choice), or through any other combination of these alternatives. Since real world distributed systems feature nondeterminism, any faithful specification should also.

- several behaviors are specified for a given object fullfilling a given role in the scope of a given relationship.
- a message is executed on an object that is participating in two (or n) differents roles in the context of different relationships.

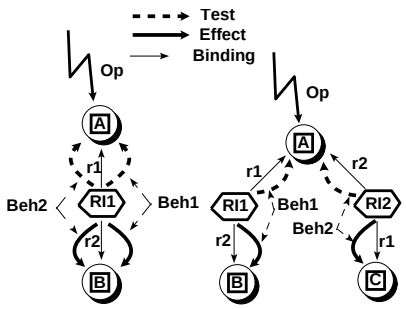

Figure 3: two cases of Nondeterminism

In TIMS BL, nondeterminism may occur when for a given message, several behaviors can be fetched and executed. As in Disco[2], there is no need for explict construct in the language (like the || of LOTOS) to specify concurency.

Summary

A MO behavior corresponds to the execution of a piece of code (body clause) when it receives a message at one of its interfaces, if the guard (when clause) is evaluated to true (i.e. enables the execution). The body of a behavior is an imperative~/ procedural piece of Scheme[3] code. There is no a priori structure imposed on it. Since usual programming features (i.e. control flow structures, variable notation, ...) are required, the use of an existing and well-known programming language, Scheme reveals to be a reasonable choice. A MO behavior can either be defined in the context of a relationship (behavior associated to a role) or it can be defined associated to a message (i.e classical CMIS operation or GRM abstract operation). This definition is done in the (scope clause). The execution of the body is immediately preceded and followed by a pre-condition (pre clause) and a post-condition (post clause), respectively.

3 Conclusion

TIMS provides an environment for the rapid-protoping of distributed and reactive systems. The approach is based on a Behavior Language doted with an operational semantics. This results as executable specifications that can be exercised in a simulation environment. The paper has presented the salient features of the Behavior Language : (i) the use of relationships according to the GRM enables to organise a behavior specification into clear, and manageable pieces. (ii) specification aspects (guard, pre, post) are clearly separated from emulation aspects (body) thanks to well identified behavior clauses.

References

1. Sandro Mazziotta and Dominique Sidou. “A Scheme-based Toolkit for the Fast Prototyping of TMN Systems.” Seventh International Workshop on Distributed Systems: Operations & Management. October 1996, L’Aquila , ITALY.
2. Hannu-Matti Jarvinen and Reino Kurki-Suonio. “DisCo Specification Language: Marriage of Action and Objects.” Proc. of 11th International Conference on Distributed Computing Systems. May 1991,Arlington, Texas, USA.
3. W. Clinger and J. Rees. “Revised^4 Report on the Algorithmic Language Scheme.”ACM Lisp Pointers, 1991.