34

Design and Testing of Information Models in a Virtual Environment

R. Eberhardt[1], S. Mazziotta[2], D. Sidou[2]
Swiss Telecom PTT R&D and Institut Eurécom
[1]CH-3000 Bern 29, +41 31 338 81 45, +41 31 338 59 59
eberhardt@vptt.ch
[2]F-06904 Sophia-Antipolis Cedex, +33 4 93 00 26 43,
+33 4 93 00 26 27, {mazziott|sidou}@eurecom.fr

Abstract

The paper introduces the TMN-based Information Model Simulator (TIMS) toolkit, a rapid-prototyping environment for TMN information models and explains how it is used in a real-life example. Based on GDMO/ASN.1, GRM and a formal behavior description, the toolkit generates TMN systems and allows both the visualization of MIBs at run-time and the access to it through CMIP. Behavior of object models is described through the use of relationships which leads to considerable simplification of the behavior and is the first step for a distributed environment. Assertion mechanisms permit the designer to verify the correct state of the model at run-time. The paper publishes the results of the tools use for the specification of a management interface for the V5 access network interface.

Keywords

Information models, GDMO, Relationships, formal behavior, simulation

A. A. Lazar et al. (eds.), *Integrated Network Management V*
© Springer Science+Business Media Dordrecht 1997

1 INTRODUCTION

The goal of the TIMS-project is to provide a laboratory environment for TMN-designers enabling them to prototype solutions, build mock-ups and to test them prior to standardization, procurement or network introduction. This approach, so the intention, will speed up the standardization process and improve the quality of the specifications. A mock-up can also be used to support procurement, enhance education and improve acceptance testing of the finished products (Eberhardt et al, 1996).

We believe that the gap between formal specification and actual implementation should be as small as possible. The telecom operator profits from a tighter specification while the developer can (semi-) automatically create parts of its application. GDMO is such an example. TIMS therefore focuses on formal but executable specifications of static and dynamic schema's in the TMN, i.e. the relationships between managed (and managing) objects and their behavior description.

What is required to reach this goal? Behavior specifications must be separable into a specification part and an emulation (or algorithmic) part. The specification part defines rules on the static and dynamic properties of the information model, is implementation-independent and therefore normative. The emulation part describes the algorithm of an operation within or on the information model (e.g. Ensemble-scenarios (NMF 025, 1992)) and is non-normative. Following principles stated in (Kilov, 92), the specification paradigm is declarative and advocates the use of relationships. To keep the specifications understandable and manageable, only functional behavior properties are considered. Thus quantitative or physical aspects are abstracted away (e.g. actual distribution, timing and real-time issues). Validation is based on simulation and testing of the executable specification, therefore mathematical reasoning is not used. Finally, to ensure executability and rapid-prototyping, the proposed behavior specification language is based on an existing and interpreted language: Scheme (Clinger and al, 1991).

Plan of the Paper

Section 2 introduces the highlights of the TIMS toolkit seen from the users perspective, namely its interfaces and the method of behavior formalisation. Section 3 shows in a practical example how TMN information model design is performed. Section 4 summarizes the results of the design of the $V5.1_{AN}$ management interface which show that the approach chosen in TIMS is viable indeed. Conclusions on the work up to now are discussed in section 5.

2 TIMS TOOLKIT

Clause 2.1 introduces the technical support implemented within the TIMS platform, described in detail in Mazziotta and Sidou (1996). Clause 2.2 presents the main features of the TIMS Behavior Language (BL) and its structure.

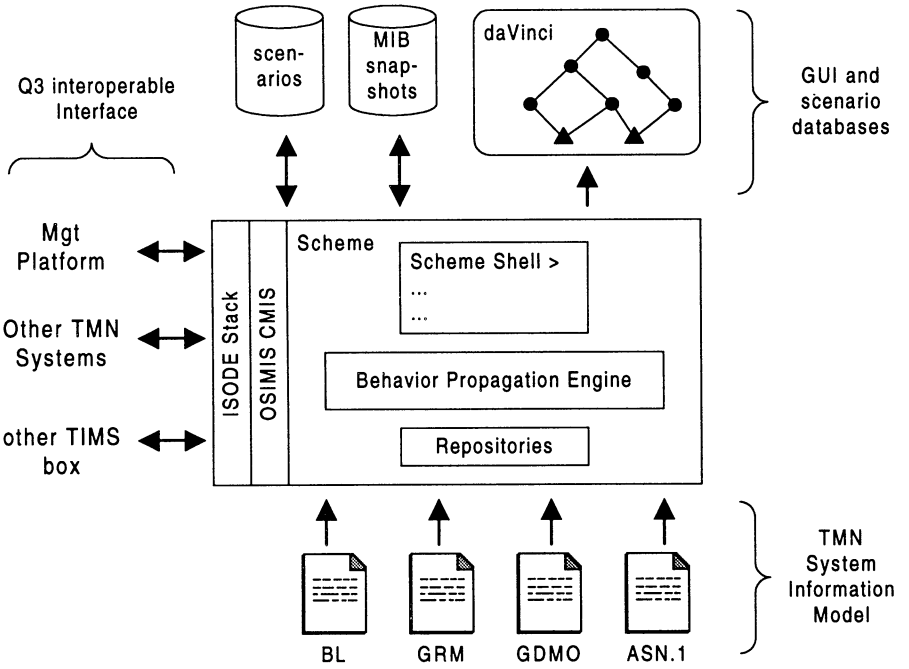

Figure 1 : TIMS and its environment

2.1 The Implementation

TIMS is a single toolkit running under UNIX. It is built around the OSIMIS development toolkit (Pavlou et al, 1995) and other public domain tools. TIMS consists of the simulator part called "TIMS box", support software such as the simulation manager and visualization and browsing tools. The TIMS box is built as an open system with following interfaces made available:

- Q3 interoperable interface (left hand side of Figure 1)

- the TMN information model integration (bottom of Figure 1)
- test execution environment and the GUI (top of Figure 1)

Q3 Interoperable Interface

TIMS boxes may be accessed via a CMIS API and an underlying OSI protocol stack. The CMIS API and the OSI-stack used are taken from the OSIMIS library and the ISODE implementation, respectively. Providing CMIS allows for (1) several TIMS boxes to interact in different roles (agent, manager, manager/agent) and (2) to integrate with real TMN applications (e.g. commercial management platforms, real network elements, network emulators, etc.). This feature is fundamental for the reuse of TIMS specifications in procurement, as reference configurations, for education and testing of real TMN systems.

Input of the TMN Information Model

TMN Information Models (IM) are composed of GDMO, ASN.1 and (recently) of GRM specifications which correspond to the static part of the specification. TIMS requires only the relationship class specification of the GRM; the relationship mapping productions are incomplete and therefore embedded in the behavior. The dynamic part consists of the behavior specification (c.f. section 2).

GDMO, ASN.1 and GRM parsers* provide suitable output that can be integrated in the Scheme environment. This makes all the required information about Managed Objects Classes, Relationship Classes available, e.g. to ensure the correctness of operations performed on Managed Objects. For ASN.1, a reasonable support of a value notation is needed since behaviors often include the creation and manipulation of complex ASN.1 values. Examples of such constructs are provided in section 3.

Graphical User Interface

The GUI mainly corresponds to the command panel which is based on the Tk toolkit. This panel controls the Test Execution Environment and allows the visualization of the simulation. The Test Execution Environment consists of a Scenario Player, and a Snapshot Player:

- The Scenario Player enables the user to run sequences of management operations or real resource changes (e.g. emulating equipment failures). Step-wise execution allows tracing all changes in the MIB.

---

* The GDMO parser is based on the OSIMIS GDMO compiler front-end (Pavlou et al, 1995), ASN.1 on the ASN Free Value Tool (ASNFVT, 1992). The GRM parser was developed in TIMS.

- The Snapshot Player enables to save the current system state and to use it as starting point for scenario runs. This feature is very important in the prototyping phase when a given state is reached only after a long simulation run.

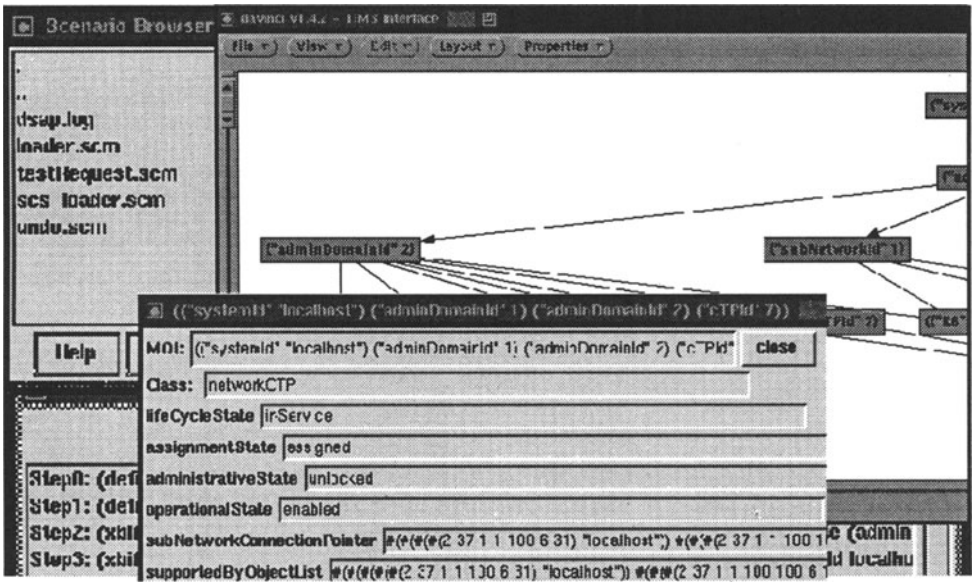

Figure 2 : Screen Shots of the TIMS Execution Environment (the graph display, the contents of a managed object and the scenario player)

TIMS presents the MIB as a graph of information objects and relationships. The visualization is done with daVinci (davinci, 1996), a generic graph visualization tool. This tool provides automatic layout generation as well as selective suppression of sub-graphs. The user, selecting a given node of the graph, can then access the managed objects' attributes and their associated values presented in form of TK widgets.

2.2 Behavior Specification

TIMS Behavior Language Features

TIMS follows in essence the approach advocated by Kilov (1992) where both the use of relationship-based formalisation and asserted specifications are employed.

Relationship-based Formalisation : BL employs the ODP notion (X.902, 1995) of "action" to describe a behavior : "An action is defined as something which happens, of interest for modeling purposes and associated with at least one object. [...]". Very often, an action can be associated to causal dependencies between objects. This leads to a behavior-model based on roles and relationships. In the context of TMN information models the General Relationship Model GRM (X.725, 1995) is a natural candidate.
Relationship-based Behavior Formalisation provides simpler, more readable and expressive behavior specifications because managed objects are identified through roles instead of raw attribute pointers or any other mechanism currently available to realize a relationship.

Use of assertions : Assertions define the pure specification aspects of the system. In TIMS, assertions are properties that are checked during the execution of the simulation. Although often considered a burden, assertions prove very valuable during incremental and component-based model development. Experience shows that the specifier can not control the whole complexity of its system and especially when there is a lot of "behavior interference". Complex behavior often emerges when relationships overlap, i.e. an object participates in several relationships in different roles. Specifying assertions is an effective method to ensure the correctness of the resulting specification at run-time (protecting the specifier from unwanted or conflicting behaviors).

Structure of a Behavior

A behavior is always defined in the context of a relationship (*scope* clause) between objects fulfilling given roles. It 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 consists of Scheme code. There is no a priori structure imposed on it. A CMIS API is provided to specify any CMIS operation within behaviors (GET, SET, ACTION, ...). In addition a GRM-like API provides access to the required relationship abstract operations, i.e. ESTABLISH, BIND, UNBIND, TERMINATE. Since usual programming features (i.e. control flow structures, variable notation...) are required, the use of an existing and simple programming language, Scheme, reveals to be a reasonable choice. The execution of the body is immediately preceded and followed by a pre-condition (*pre* clause) and a post-condition (*post* clause), respectively. Finally, each behavior is labeled by the specifier for the purpose of identification, especially during debugging.

Invariants are described by a behavior with post-condition but no body part. A relationship-invariant is a behavior with the message received being any kind of

operation executing upon objects bound to the relationship. A role-invariant restricts the message target to the objects involved in the role (cf. section 3).

3 MODELING IN TIMS

This section provides a step-wise, example-based tutorial introduction to how modeling is typically performed in TIMS. The examples derive from the management interface model for V5 (ETS 300 376-1, 1994).

Management and domain requirements, expressed in prose or in a semi-formal representation, form the basis for identifying managed and managing resources. Placing the resources in context, i.e. building the static schema, is the logical next step. Some invariants, such as the cardinality between managed objects, will have become apparent at this point. Were we writing software, we'd sit down and begin coding - in TMN partially reflected in the Ensemble-scenarios. Coding reveals whether the choice of resources (managed objects) was correct, if the static schema is useful and also gives an indication on the ease of use of the interface. This is the point where we believe a TMN-laboratory becomes useful. The following is an example on how this is achieved in TIMS:

Requirement: "A field replaceable unit (FRU) is represented as a physical resource (the equipment) and the logical resource (the functionality, here called userPort). The operational state of the userPort is dependent on the physical resources it requires (card, power supply, etc.). The userPort becomes operational automatically once all physical resources have been installed."

Give relationships a name : Given the resources and an ER-diagram, writing GRM provides its structure, the role names, their cardinality and their relationship operations.

```
Rresource RELATIONSHIP CLASS
    BEHAVIOUR RresourceBehaviour;
    SUPPORTS ESTABLISH, TERMINATE;
    ROLE physicalRole
            COMPATIBLE-WITH equipment      [...]
    ROLE logicalRole
            COMPATIBLE-WITH userPort
            PERMITTED-ROLE-CARDINALITY-CONSTRAINT INTEGER(1)
            REQUIRED-ROLF-CARDINALITY-CONSTRAINT INTEGER(1..MAX)
            [...]
```

Define constructors & destructors for the static schema : "The installation of the FRU results in the automatic generation of the logical resources (userPort) and other relationships. Trigger for this behavior is the creation of the equipment through an external message (e.g. an M-CREATE)."

```
(define-behavior "create-equipment" (scope "RmanagedElement")
    (when (and (param Create?) (param moc=? "equipment")))
    (pre ...)
    (body ...
      (set! (ttpInstance (Create userPort ...)
      (Establish (operation-name) "Rresource" (rir:gen-ri-inst "Rresource")
                 (("physical" equipmentInstance)
                  ("logical" (ttpInstance))))
      ...)
    (post ...))
```

Write scenarios (management function) : TIMS does not make a difference between code representing behavior internal to the MIB and operations between manager and agent. In the example, the operational state of userPort is set to "disabled" if it is in the "enabled" state and one of its supporting physical resources goes to "disabled".

```
(define-behavior "logical enabled->disabled "
    (scope-rel "Rresource")
    (when (and (asn=? (Get (Part (ri) "physical") "operationalState") 'disabled)
          (some (lambda (userPort) (asn=? (Get userPort "operationalState") 'enabled))
                (Part (ri) "logical"))))
    (pre ...)
    (body (for-each (lambda (userPort)
                (cond ((asn=? (Get userPort "operationalState") 'enabled)
                       (Set  userPort "operationalState" 'disabled))))
              (Part (ri) "logical")))
    (post (for-each (lambda (userPort) (asn=? (Get userPort "operationalState") 'enabled)))
```

For those not familiar with Scheme, the lambda-expression allows to define unnamed functions with temporary variables (such as a function asserting the operationalState of userPort being enabled).

Make the model water-proof : Having written some code, the designer must now review the static and dynamic schemas and check at which points the behavior could fail due to ambiguities or bugs. Write assertions or invariants (e.g. as part of a relationship) handling these cases and run the simulation. Assertion-failures will point at problems in the code. If the simulation shows unexpected behavior this indicates weak guard statements. Depending on the run-time problems, the designer may now have to revisit all previous phases up to the design level.

Finalise the model : Once the model works as expected the tough work is over. The next steps involve mapping the behavior onto a system management model (called Engineering Viewpoint, in this case OSI Systems Management). The designer needs to write the relationship mapping (pointer structures, etc.) and to implement action and notification signatures together with their parameters for behavior crossing the system management boundary. Example : "A userPort is associated to a v5Ttp using the ACTION setReciprocalPointer".

```
(define-behavior "Rv5Interface-SetReciprocalPtr"
    (scope-rel "Rv5Interface")
    (when (and (ri) (param Action?)  (param Action=? "setReciprocalPointerAction")))
    (pre)
    (body (let* ((aEndObject(car (param argument)))
                 (zEndObject (cadr (param argument)))
    [...]
-- this action results in the call of an establish & bind operation between aEndObject and
-- zEndObject.

(post ...))
```

4 A CASE STUDY: V5.1 MANAGEMENT

This section provides results and metrics acquired during the implementation of the V5.1 management model for configuration management. The case study was developed in parallel to a V5 management interface specification which is to be used for procurement purposes.

The case study focused on the overall behavior of the management architecture and less on the mapping between hardware behavior and its TMN representation (e.g. state mappings of protocol engine finite state machines). The model consists of 12 managed object classes and 18 relationship classes. It covers scenarios (management functions) for the insertion, removal and configuration of ports as well as their provisioning (cross-connection). A minimal implementation counts 46 managed object instances of which 31 represent 64 kBit/s channels and can be therefore reduced while prototyping (e.g. down to 10). The average size of a simple dynamic schema (e.g. state change between managed objects) lies at 20 lines of Scheme-code. Complex schema's, such as the constructor for a v5interface MIB structure involve around 50 lines. The code is very repetitive so that cut & paste helps reduce tedious writing.

The major effort (40 %) lay in acquiring the necessary V5 domain knowledge, sometimes down to the protocol level. Implementing the V5 static schema (resource selection, GRM) was quickly done, while embedding the fragment into an overall MIB

architecture turned out to be more difficult than expected (total 20%), mainly due to functional restrictions found in current GDMO libraries (M.3100). Implementing behavior itself resulted in a repeated review and refinement of the requirements, sometimes identifying new demands along the way. Behavior-design and debugging made up for another 40% of the effort. The modeling of invariants and assertions went hand in hand with the static schema (GRM constraints) and - when actually specified - during the development of each behavior.

4.1 Evaluation

The language Scheme proved difficult at first, as the engineers implementing the case study were more familiar with imperative languages. Once understood, however, the interpreted nature of Scheme allowed for trial-based code-development. The TMN-API's themselves were rapidly understood and easily used. The graphical interface proved useful only for educational purposes and for browsing of small MIB's. A run-time debugger for behavior traces and for the analysis of interference's between behavior executions would speed up coding.

Designing and coding the dynamic schema is very repetitive, but requires detailed knowledge of both the domain and the TMN libraries. We believe that in future it should be possible to reuse generically defined relationships and their behavior. Guards are the enabling condition for the execution of behavior. Specified incompletely, they may lead to unexpected side effects difficult to foresee by the designer. Due to GRM, many assertions and invariants came for free, saving extra coding. Not surprisingly, assertions were used only sparsely because they are often considered tedious to formulate and very large. This applies especially for post-conditions, fundamental for evaluating the consistency of the MIB.

Relationships proved to be very useful for both design and coding. Making the model accessible via CMIP requires the additional effort of implementing the relationship mapping. We believe that a relationship management service reflecting GRM-constructs would simplify management scenarios, provide for distribution transparency and aid in behavior description.

The V5 management interface specification project running in parallel benefited a lot from our design work, especially when it came to understanding the finer points of the model, its restrictions and pitfalls. The TIMS scenarios could be mapped almost 1:1 into the ensemble specification.

5 CONCLUSIONS

The paper gives an overview of the rationale guiding TIMS' development. It describes the TIMS TMN toolkit and its highlights. The relationship behavior formalisation is introduced and a guide on how to model TMN interfaces using TIMS is provided.

Although the initial learning curve is steep, it compares favorably with commercial toolkits. While Scheme has its advantages, it isn't widely known and not sufficiently readable to be used as specification; a specification-typesetter could be considered a solution.

The language features and the use of relationships provide everything required by ODP and align reasonably well with new methodologies suggested in ITU-T (G.851, 1996).

The greatest cost remains in acquiring the domain knowledge necessary to be able to develop a management interface. A laboratory environment helps understanding both the domain and its management design. The speed in which new TMN information models were implemented (smaller test cases with up to 8 managed objects were implemented in less than 2 weeks) indicates that it is possible to bring TMN into the laboratory.

5 REFERENCES

ASNFVT (1992), ASN.1 Free Value Tool, at ftp://osi.ncsl.nist.gov/pub/osikit/"

Clinger, W. and Rees, J. (1991) - Report on the Algorithmic Language, Scheme. ACM Lisp Pointers, vol. 4 (3), 1991. - Available at http://www.cs.indiana.edu/scheme-repository/doc/standards/r4rs.ps.gz.

davinci (1996) The Interactive Graph Visualization System daVinci, available at http://www.informatik.uni-bremen.de/~inform/forschung/daVinci/daVinci.html.

Eberhardt, R. and Sidou, D. et al. (1996) Executable TMN specifications with TIMS, proceedings of the NOMS'96 , IEEE. Available at http:/www.eurecom.fr/~tims

ETS 300 376-1 (1994) Q3 interface at the Access Network (AN) for configuration management of V5 interfaces and associated user ports; Part 1: Q3 interface specification, ETSI Technical Standard, December 1994

NMF 025 (1992) The Ensemble Concept and Format, Network Management Forum

G.851-01 (1996) Draft Recommendation, Managemeent of the transport network - Application of the RM-ODP framework. June 1996

X.725 (1995) 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.

X.902 (1995) Basic Reference Model of ODP - Part 2: Foundations, ISO 10746-2, ITU X.902.

Kilov, H. (1992) From OSI Systems Management to an Interoperable Object Model: Behavioural Specification of (Generic) Relationships, Proceedings 3rd Telecommunications Information Networking Architecture Workshop (TINA 92), 1992, Narita, Japan,

Mazziotta, S. and Sidou, D. (1996) - A Scheme-based Toolkit for the Fast Prototyping of TMN-systems - 1996. Seventh International Workshop on Distributed Systems : Operations & Management.

Pavlou, G. and McCarthy, K. and Bhatti, S. and Knight, G. and Walton, S. (1995) The OSIMIS Platform: Making OSI Management Simple, Integrated Network Management IV, 1995, Santa Barbara, USA

6 BIOGRAPHY

Rolf Eberhardt joined Swiss Telecom R&D in 1991 following his graduation in computer science from ETH Zürich, Switzerland. He has worked as TMN specification engineer, consultant and in standardization (ETSI TM2, ITU-T SG4, NMF), mainly on transmission networks. His interests focus on behavior formalisation, access networks and X-interfaces.

Sandro Mazziotta graduated in computer science in 1993 from Nice-Sophia Antipolis University, France. In 1994, he joined the Corporate Communications department of Institut Eurécom where he pursues a Ph.D. thesis. His research interests include the specification, validation and testing of the dynamic behavior of object-based distributed systems.

Dominique Sidou received his computer science degree: "Diplome d'Etudes Approfondies (DEA)", from University Paul Sabatier, Toulouse - France in 1989. His research interests focus on distributed systems: behavior modeling and validation, distributed object technologies (ODP, CORBA), and network and systems management (TMN, SNMP). He is research engineer at Institut Eurécom (Sophia-Antipolis, France).