Using TIMS for the Prototyping of SMFs

S. Mazziotta – J. Labetoulle
Institut Eurécom,
2229 route des crêtes, B.P. 193,
06904 Sophia Antipolis cedex, France.
{mazziott, labetoul}@eurecom.fr

R. Eberhardt
Swiss Telecom PTT,
Research & Development,
3000 Bern 29, Switzerland.
eberhard@vptt.ch

Abstract

The main objective of the TIMS (TMN-based Information Model Simulator) project is the simulation / prototyping of TMN Information Models (IM). The main contribution of the project is the definition of Managed Object Behavior Formalization paradigm based on relationships modeling : the Behavior Language (BL). Rather than describing its features one by one, TIMS BL is presented, in a tutorial approach, through the modeling of a particular System Management Function (SMF) : the Event Report Management Function (ERMF).

1 Introduction

background Current TMN-based information model (IM) standards take too long to specify, standardize, implement, test and introduce. The TMN-based Information Model Simulator – TIMS – explore the possibilities of reducing this development life-cycle. TIMS should enable :

1. the rapid prototyping of TMN agent and manager functions; and later
2. the generation of reference configurations.

TIMS approach

- **architecture of the MIB** : Past experiences in IM design indicated that one of the difficulties which plagued the TMN was the inability to completely describe the Managed Object IM and its behavior. This is not the unique problem; Managed Object behaviors are limited to object boundaries. TIMS approach relies on high–level construct modeling the relationships between objects independently of the type of implementation (pointers, attribute values, name binding, etc...).
- **formal but executable specifications** : The specifications must be separable into a "rule" part and an "algorithmic" part. The "rule" part, i.e. assertions on the static and dynamic properties of the IM, is implementation–independent and therefore normative. The "algorithmic" part describing the method of workings of an operation within or on the IM would be considered informative rather than normative.
- **keep it simple and stupid** : Last but not least, TIMS is to be a tool not only be understood by its creators but also by engineers. As a result several design limitations were introduced:
  - no engineering issues: distribution, process partitioning, etc...
  - no timing / real-time issues, no asynchronism.
  - no mathematical reasoning required.

prototyping SMF A classical TIMS case study is composed on one hand by a TMN configuration (which is technology and / or topology dependent) and on the other hand by several scenarios representing a sequence of management operations fulfilling one or several systems management functional areas (SMFA) (fault, accounting, configuration, performance and security management) requirements. These SMFAs are not standardized as such. Rather, a number of specific functions, referred to as systems management functions (SMF) have been defined to provide the basic functionality specified in the five SMFAs. Each of the SMF standards defines the functionality for the SMF and provides a mapping between the services provided by the SMF and CMISE. The support of SMFs is considered as an entire part of the TMN IM.

This explains why the need to support SMFs arises very early in TIMS. Naturally, it has been decided to simulate SMFs (like other parts of the IM) with TIMS. The prototyping of SMFs reveals as an interesting challenge since SMFs should be designed in order to be reusable.

Goal of the Paper The idea of the paper is to show, how a user can model a SMF with TIMS and thus in a tutorial approach. The Event Report Management Function [Ermf] was chosen first because it is widely used and well–known and secondly because it is enough complete to expose all the TIMS BL features.

[0] 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.

Plan of the Paper

- Section 2 gives an overview of TIMS.
- Section 3 presents the ERMF standard.
- Section 4 presents the TIMS modeling of ERMF.
- Section 5 concludes the paper.

2 TIMS

TIMS project has defined, designed and implemented mainly two part: the first one is a toolkit driving the simulation (also called TIMS box or platform) and a Behavior Language (BL) that defines Managed Object behaviors which is the basic input to the simulation. These two components are described shortly in this section.

2.1 TIMS Toolkit

This section presents shortly the technical support implemented within the TIMS platform. For more details, the reader should refer to [Mazziotta et al.96] where these aspects are described more precisely.

TIMS is a single toolkit running under UNIX. It makes uses of the OSIMIS [Pavlou et al.95] development toolkit and other public domain tools. TIMS consists of the simulator itself called the "TIMS box" and user interfaces to drive and observe the results of behavior simulations (visualization and browsing tools). The TIMS box is built as an open system with following interfaces made available : Tcl / Tk for GUI programming and visualization, a management interface consisting of upper–layer Q3 stack on top of TCP / IP (RFC 1006) and the Guile programming environment based on the Scheme programming language.

The following three components are relevant for user interaction:

- Q3 interoperable interface (left hand side of figure 1)
- TMN IM integration (bottom of figure 1)
- Test Execution Environment and GUI (top of figure 1)

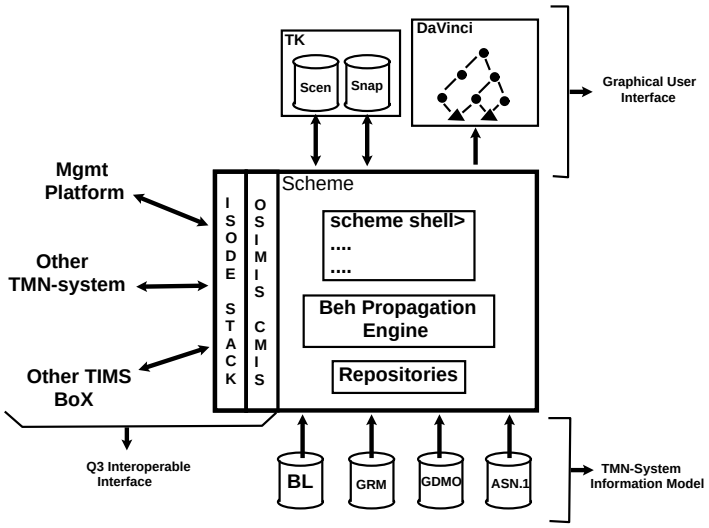


Figure 1: TIMS and its environment

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.
2. **TMN IM integration** : One of the main initial objectives of the project was to simulate / prototype standardized TMN IMs. Classically they are composed of GDMO / ASN.1 and (recently) of GRM specifications which correspond to the static part of the IM. The Dynamic part consists of the behavior specifications which are described in section 2.2.

3. **Test Execution Environment and GUI** : The GUI mainly corresponds to a command panel implemented in TK. This panel controls the Test Execution Environment and allows the Visualization of the Simulation.
   - The Test Execution Environment includes a scenario player (tool that enables to load scenarios: sequence of management commands) and a snapshot player (tool that enables to save current system state and to use it as starting point for scenario runs.
   - The Visualization of the Simulation is done with *daVinci* [davinci], a generic graph visualization tool. It presents the MIB as a graph of information objects and relationships. The user, selecting a given node of the graph, can then access the managed objects attributes and their associated values presented in the form of TK widgets. IM dynamics, i.e. the flow of interactions between objects is also visualized (Each time an object takes part in an activity, its graphical representation changes its color).

2.2 TIMS Behavior Formalization

This section presents shortly the main features of the TIMS Behavior Language (BL) followed by an introduction of a behavior. Some examples are shown during the prototyping of ERMF in section 4.

BL Features TIMS follows in essence the approach advocated by Kilov [Kilov92], where both the use of relationship-based formalization and asserted specifications are employed.

- **Relationship-based Formalization** : provides simpler, more readable and expressive behavior specification because MOs are identified through roles (participating to a given relationship) instead of raw attribute pointers or any other mechanism currently available. In the context of TMN IM, the General Relationship Model (GRM) [Grm] is a natural candidate.
- **Asserted Specifications** : Assertions define the pure specification aspect of the system. IN BL, 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 in case of "behavior interference".

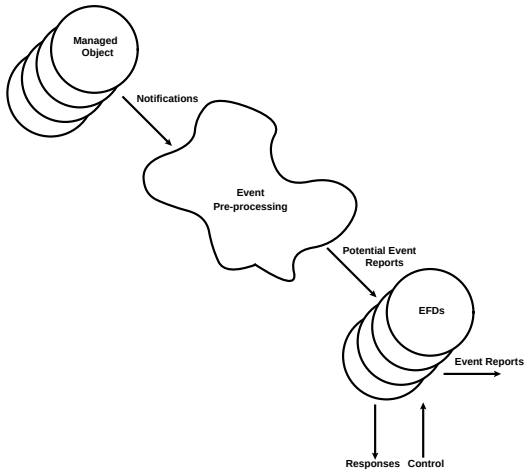
Structure of a Behavior A MO behavior corresponds to the execution of a piece of code (**body** clause) when it receives a message (**exec-trigger** clause) at one of its interfaces, if the guard (**when** clause) is evaluated to true (i.e. enables the execution). The body of a behavior is an imperative / procedural piece of *Scheme* [Clinger et al.91] 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 Event Report Management Function

This section describes the Event Report Management Function (ERMF). It is first presented the model. Then follow the description of the different functions associated to the SMF and a description of its components.

3.1 Model

The model is represented in fig 2.



The event report management model describes the conceptual components that provide for remote event reporting and local processing of PERs (Potential Event Report). The model also describes the control messages, event reporting messages and retrieval messages. The conceptual event pre-processing function receives local notifications and forms the PERs. Conceptually, these PERs are distributed to all EFDs (Event Forwarding Discriminators) that are contained within the local open system. The EFD is used to determine which event reports are to be forwarded to a particular destination during specified time periods.

Figure 2 : Event Report Management Model

3.2 Functions

Event reporting management provides the means by which discrimination and forwarding can be initiated, terminated, suspended, or resumed and through which the attributes of the EFD can be read and modified.
Event reporting management comprises the following:

- **initiation of event forwarding :** The PT-CREATE service defined in [Omf] is used to allow one open system to request that another open system create an EFD, thereby requesting that new or additional event forwarding controls be imposed.
- **termination of event forwarding :** The PT-DELETE service defined in [Omf] is used to allow one open system to request that another open system delete one or more EFDs, thereby requesting that some event forwarding controls be terminated.
- **suspension, resumption, modification of event forwarding conditions :** The PT-SET service defined in [Omf] is used to allow one open system to request that another open system change the administrative state or other settable attribute of the EFD.
- **retrieval of event forwarding conditions :**
  The PT-GET service defined in [Omf] is used for retrieving the attributes of the EFD.

3.3 The Discriminator Object

The basic superclass is the discriminator object class. The discriminator is a managed object that allows a managing system to exercise control over the management operations that may be accepted and the event reports that may be forwarded, by a managed system. Discriminators can, therefore, be created, deleted, read and modified. In addition, the activity of discriminators can be suspended and resumed by means of manipulating their administrative states.

Attributes

- Discriminator Id : This attribute is used to uniquely identify the instance of a discriminator.
- Discriminator construct : This attribute specifies the test conditions which will be used by the discriminator in testing PER.
- Administrative state : This attribute specifies the administrative state in which the discriminator is to be created. The discriminator administrative state is a subset of the administrative state defined in [Stmf]. The following administrative states are defined:
  - unlocked : processing of the information by the discriminator is permitted by a managing system;
  - locked : processing of the information by the discriminator is prohibited by a managing system.
- Operational state : This attribute specifies the operational state of the discriminator. The discriminator operational state are those defined for the operational state in [Stmf]. The following operational states are defined:
  - enabled : the discriminator is operational
  - disabled : the discriminator is inoperable

Packages To accommodate various levels of complexity in scheduling event reporting activity periods, conditional packages that are related to scheduling are defined for discriminator.
Scheduling packages provide discriminators with the ability to automatically switch between their reporting-on and reporting-off conditions. If no scheduling package is present in a discriminator, it is always in reporting-on condition.

- Availability status package : This package contains the following attribute:
  - Availability status : This attribute reflects the availability status of the managed object. When the resource has been made unavailable in accordance with a predetermined time schedule its value will be "off-duty".
- Duration package : The duration package provides the ability to automatically control the time that a managed object starts and stops functioning through the use of the start time and stop time attributes.
  - Start time : This attribute defines the date and time at which an object starts functioning.
  - Stop time : This attribute defines the date and time at which a managed object stops functioning.

Other packages are available (but are not simulated), the reader should refer to [Ermf] for more informations.

3.4 The EFD Object

The EFD allows specification of conditions to be satisfied by PERs related to managed objects before the event report is forwarded to one or several particular destination(s). The EFD is a subclass of the discriminator object class.

Attributes In addition to the attributes inherited from the discriminator, the EFD has the following attribute:

- Destination : This attribute identifies the destination(s) to which the discriminator forwards event reports. The destination may be a single AE Title (application entity title) or multiple AE Titles.

Packages

- Backup destination package : This package has two attributes which specify the backup destinations and the active destination. This package is present when it is required to provide a backup for the destination.
- Mode package : This package has one attribute and is present when a manager can specify / modify the mode for reporting events.

4 Prototyping ERMF with TIMS

This section presents the model and some of the behaviors implemented in TIMS in order to simulate the ERMF standard.

4.1 Mapping the ERMF model in TIMS

The first step of Prototyping ERMF with TIMS, is to map the standardized model presented in the previous section into a relationship-based one to be able to plug behaviors. It consists simply of identifying the relationships and their associated participants. Several MO fulfilling the same behavior must be grouped in a unique role.

Concerning the ERMF standard, it is proposed to define one relationship "EventProcess" where two roles are present:

1. the `mos` role that represents every MO susceptible to emit notification; and
2. the `efds` role that represents every EFD created within the system.

Note that an EFD is also a MO.

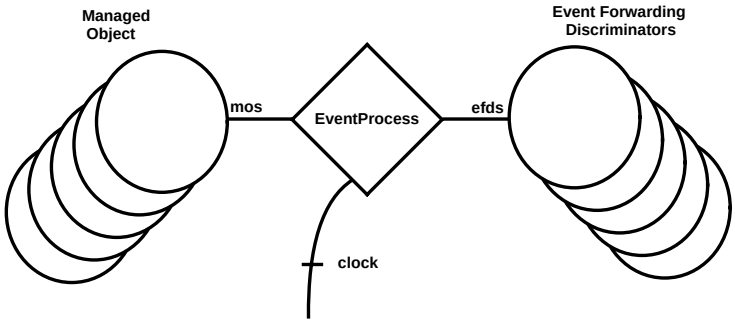


Figure 3: GRM View of Event Report Management

Following the model presented in figure 3, the GRM specification is presented below. A GRM specification are composed by relationship class specifications and by relationship mappings. In TIMS, only the relationship classes are specified; mappings are not used[1].

```
EventProcess RELATIONSHIP CLASS
   BEHAVIOUR EventProcessBehavior;
   SUPPORTS ESTABLISH,
            TERMINATE,
            QUERY get-current-time,
            USER DEFINED modify-current-time,
            USER DEFINED send-notifications,
            USER DEFINED process-per;
   QUALIFIED BY clock;
   ROLE mos
        PERMITTED-ROLE-CARDINALITY-CONSTRAINT INTEGER(1)
        REQUIRED-ROLE-CARDINALITY-CONSTRAINT INTEGER(1..MAX)
        BIND-SUPPORT
        UNBIND-SUPPORT
```

[1] The GRM productions are incomplete, especially in the `OPERATION MAPPING` template. In fact, TIMS behavior specifications can be used to specify completely precise and unambiguous mappings for abstract operations on relationships.

```
        PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT INTEGER(1)
   REGISTERED AS { Role 11 }
   ROLE efds
        COMPATIBLE-WITH eventForwardingDiscriminator
        PERMITTED-ROLE-CARDINALITY-CONSTRAINT INTEGER(1)
        REQUIRED-ROLE-CARDINALITY-CONSTRAINT INTEGER(1..MAX)
        BIND-SUPPORT
        UNBIND-SUPPORT
        PERMITTED-RELATIONSHIP-CARDINALITY-CONSTRAINT INTEGER(1)
   REGISTERED AS { Role 12 };
REGISTERED AS { RelationshipClass 1 };
```

"EventProcess" relationship is created / deleted dynamically (supports ESTABLISH,TERMINATE) and owns several internal actions (QUERY and USER DEFINED). Within the relationship an internal attribute is defined (QUALIFIED)[2].

The ROLE template allows the specifier to define the MO class associated to the role (COMPATIBLE-WITH). It also defines the properties associated to the role: cardinality and the dynamic binding / unbinding (BIND / UNBIND-SUPPORT) to the relationship.

Once the relationship and roles are defined, the dynamic part comes, i.e. the behaviors. It is first presented the behavior of the role `mos` and then it is presented the behavior associated to the role `efds` . For each behavior a non formal definition, taken directly from the standard, is presented before the BL associated code. This mimics the modeling process of behaviors, i.e. the specifier has to gather from the informal specification anything describing a behavior of interest. For the clarity of the explanation, behaviors are not completely specified. The reader is informed by "..." that a template is not expanded.

4.2 Specification of the `mos` behavior

The MO associated to the `mos` role bootstraps the system, by binding to the created MO to the `EventProcess` relationship. This allows for behaviors to be plugged in the context of this relationship. Then notifications can be send and further processed by behaviors modeling the event processor.

Binding the EventProcess Relationship

```
(define-behavior "bind-mos-ep"
   (scope (msg Create))
   (when ...)
   (exec-trigger ...)
   (pre ...)
   (body ...
         (Bind (operation-name) "EventProcess" (param moi) "mos")
         ...)
   (post ...))
```

Sending a Notification

```
(define-behavior "send-notif"
   (scope (role "EventProcess" "mos"))
   (when (and (param Set?)
              (param attribute=? "administrativeState")
              (not (asn=? (param value) (Get (param moi) "administrativeState")))))
   (exec-trigger ...)
   (pre ...)
   (body ...
         (UserDefined (operation-name "send-notifications")
                      (moi (param moi))
                      (moc (moclass (param moi)))
                      (attrlist '((type "stateChange")
                                  (attribute "administrativeState")
                                  (oldvalue (Get (param moi) "administrativeState"))
                                  (newvalue (param value)))))
         ...)
   (post ...))
```

These two behaviors show :

[2] thanks to this qualifier, for the simulation of ERMF, it has been decided to represent the clock of the system inside the relationship.

- different behaviors scopes: The first one is a scope defined in the context of the reception of a message (i.e. creation of the object) while the second one is defined in the context of the relationship `EventProcess` for the role `mos` .
- The specifier can obtain parameters of the trigger (incoming message triggering the behavior) using the *param* primitive.
- The guard (*when* clause) for the second behavior strengthens the the triggering condition of the execution of the behavior to a `Set` on the `administrativeState`. Without the restriction giben by the guard, the behavior would be executed every times the role is touched.

```
(define-behavior "from-notif-to-per"
    (scope (role "EventProcess" "mos"))
    (when (and (param UserDefined?)
               (param UserDefined=? "send-notifications")))
    (exec-trigger ...)
    (pre (not (= (card (ri) "mos") 0)))
    (body (let ((time (Query (operation-name "get-current-time") (ri))))
            (for-each
               (lambda(XXX)
                  (UserDefined (operation-name "process-per")
                               (moi XXX)
                               (attrlist '((moi (val:get '(moi) (msg)))
                                           (moc (val:get '(moc) (msg)))
                                           (evtype (val:get '(attrlist type) (msg)))
                                           (evtime (val:get '(attrlist time) (msg))))
                                  ...))))
                  (Part (ri) "efds")))
    (post ...))
```

4.3 Specification of the `efds` behavior

Initiation of Event Report Forwarding When an EFD is created, it generates an object creation notification. This notification shall be processed by the newly created discriminator. To this end, it has to bind itself in the `efds` role.

```
(define-behavior "bind-efds-ep"
   (scope (msg Create "eventForwardingDiscriminator"))
   (when ...)
   (exec-trigger ...)
   (pre ...)
   (body ...
         (Bind (operation-name) "EventProcess" (param moi) "efds")
         (Bind (operation-name) "EventProcess" (param moi) "mos")
         (UserDefined (operation-name "send-notifications")
                      (moi (param moi))
                      (moc (param moc))
                      (attrlist '((type "objectCreation")
                                  ...))
         ...)
   (post ...))
```

Forward PER If the discriminator construct evaluates to TRUE, the EFD is in the unlocked and enabled states, and the availability status, if present, is not "off-duty", then the discriminator input object passes the discriminator and will be processed further.

```
(define-behavior "forward-per"
    (scope (role "EventProcess" "efds"))
    (when (and (param UserDefined?)
               (param UserDefined=? "process-per")))
    (exec-trigger ...)
    (pre ...)
    (body ...
          (if (and (asn=? (Get moi "operationalState") 'enabled)
                   (asn=? (Get moi "administrativeState") 'unlocked)
                   (asn=? (Get moi "availabilityStatus") 'onDuty)
                   (filter (Get moi "discriminatorConstruct")
                           (val:get '(attrlist) (msg)))
```

```
            (if (asn=? (Get moi "mode") 'confirmed)
                  (EventReport ...)
                  (EventReportNC ...)
                ...)))
    (post ...))

(define-behavior "maintain-availabilityStatus"
    (scope (role "EventProcess" "efds"))
    (when ...)
    (exec-trigger ...)
    (pre ...)
    (body (let ((time (Query (operation-name "get-current-time") (ri))))
            ...
            (if (and (> time (Get moi startTime))
                     (< time (Get moi stopTime)))
                (Set moi "availabilityStatus" 'onDuty)
                (Set moi "availabilityStatus" 'offDuty))
            ...))
    (post ...))
```

5 Conclusion

Considering this experience and the relative success of prototyping ERMF, TIMS reveals as a valuable tool for the prototyping of SMFs. This may reveal very useful to evaluate SMF features, especially for SMFs that are not yet on the market or still under standardization. When prototyping SMFs, the more important feature of the tool is not the behavior formalization paradigm, i.e. the use of relationships. In fact, from our experience relationship models needed to prototype SMFs are not likely to be complex. The features of the TIMS toolkit that revealed more important are the underlying programming and algorithmic support (given by *Scheme*) and the ability to support TMN information model characteristics. In particular, being able to work with complex ASN.1 values is required in SMFs prototyping even more than with usual MO attribute values.

References

[Clinger et al.91] Clinger (W.) et Rees (J.). – *Revised*^{4} 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] The Interactive Graph Visualization System daVinci. Available at http://www.informatik.uni-bremen.de/~inform/forschung/daVinci/daVinci.html.

[Ermf] Systems Management - Part 5: Event Report Management Function, ISO/IEC 10164-5, ITU X.734.

[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.

[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.

[Mazziotta et al.96] Mazziotta (Sandro) et Sidou (Dominique). – A *Scheme*-based Toolkit for the Fast Prototyping of TMN-systems. – 1996. submitted to Seventh International Workshop on Distributed Systems : Operations & Management.

[Omf] Systems Management - Part 1: Object Management Function, ISO/IEC DP10164-1, ITU X.730.

[Pavlou et al.95] Pavlou (G.), McCarthy (K.), Bhatti (S.), Knight (G.) et Walton (Simon). – The OSIMIS Platform : Making OSI Management Simple. *In : Integrated Network Management IV*, éd. par Hall (Chapman &), pp. 480–493.

[Stmf] Systems Management - Part 2: State Management Function, ISO/IEC 10164-2, ITU X.731.