13

An Extension of GDMO for Formalizing Managed Objects Behaviour

Joaquín KELLER
France Télécom
Centre National d'Etudes des Télécommunications/LAA/EIA
Technopole Anticipa - 2, av. P. Marzin
F-22307 Lannion Cedex, France
Tel : +33 96 05 21 77 Fax : +33 96 05 37 84
E-mail : joaquin.keller@lannion.cnet.fr

Abstract

In network management, the networks are modelled using managed objects, but there is a gap in GDMO (the semi-formal language for managed objects specification) : behaviours are defined using natural language. Since experiments using classical FDTs haven't been sufficiently satisfactory, this paper argues the idea of extending GDMO for formalizing managed objects behaviour and shows feasibility by defining a new template : the notification behaviour template. As a language is not well defined without formal semantics, formal semantics in Z of this template is also presented in this paper.

Keywords

Network Management, Managed Objects, Language Semantics, Z notation

1 INTRODUCTION

At network management interfaces, information is modelled using managed objects. GDMO, the semi-formal language for managed objects specification, offers only natural language for managed objects behaviour description, so, recently, standardization organizations have looked into the problem of the incompleteness of GDMO [ISO94], and several research teams have made some solutions emerge.

The aim of this paper is to demonstrate the feasibility of extending GDMO and to show the suitability, for formalizing managed object behaviour, of such extension regarding other methods.

Section 2 of this paper explains what a managed object is and what its use is in network management. This section discusses how managed objects behaviour is described currently, and the problems related to these descriptions, which have result in the search-

G. V. Bochmann et al. (eds.), *Formal Description Techniques VIII*

ing of solutions for specifying managed object behaviour.
Section 3 compares different solutions for formalizing managed object behaviour and argues the solution of extending GDMO. Also, this section explains how this extension should be, and presents a first draft of it.
Section 4 is the definition of one of the templates of the proposed extension of GDMO: the notification behaviour template. Section 5 presents the formal semantics of this new template. This section is written for an audience with a basic knowledge of Z [Spi92].
Section 6 is an illustrative example of the use of the notification behaviour template. And section 7 offers some conclusions and indicates what is the work that remains to achieve the construction of suitable extension of GDMO.

2 MANAGED OBJECTS

2.1 Managed Objects and their use

As telecommunication and computing networks have grown in size and complexity in recent years, so has the scale of the problems faced by network managers. One of the major technical challenges has been the number of different interfaces which must be dealt with when designing a system to manage a typical network. Recent work in standards address the problem of managing large, diverse networks in both computing and telecommunication environments. Managed objects have been introduced by the international organizations ITU-T* and ISO† to describe, in an object oriented style, the components of telecommunication and computing networks that are to be managed.

A *managed object* is an abstraction, or model, of a physical or logical resource that exists within a network. It describes a resource in terms of relevant data and operations that are used to manipulate that data. Managed objects are associated to an interface and indicate what is manageable in the network at this interface.

Managed objects which possess similar data and operations are grouped into classes. A managed object is described by its *managed object class* definition. This class definition specifies the data and operations that are contained in all objects of the class. A managed object is an instance of a managed object class. Managed object class allows to model managed objects associated to similar resources together.

Managed object classes differ at least on two points from classes of the classical Object Oriented paradigm : these points are *packages* and *notifications*.
In a managed object class attributes and operations are packaged in different packages, some of them are conditional and the other ones are mandatory. At the time of instantiation of a managed object class in a managed object, some conditional packages may be rejected. So two managed objects of the same class may have great differences depending on which conditional packages they implement.
Also managed objects have the ability to spontaneously emit notifications to report internal events.

New managed object classes can be created from existing classes by using strict incre-

*International Telecommunications Union - Telecommunication Standardization Sector (CCITT).
†International Standards Organization.

mental inheritance, i.e. adding attributes, operations and/or notifications to an existing class. Operations and notifications may also be refined to take in account the new attributes or to define a more specific behaviour.

2.2 Managed Object specification

The international standard "*Guidelines for the Definition of Managed Objects*" [GDMO] describes a set of templates to specify managed object classes. In short, the most important templates are :

- The `MANAGED OBJECT CLASS` template, that is used for defining a new class by inheritance and by grouping packages.
- The `PACKAGE` template, that allows the packaging of attributes, operations and notifications.
- The `ATTRIBUTE`, `ACTION` and `NOTIFICATION` templates that are used to specify respectively attributes, operations and notifications. Data types for the arguments of `ACTION`s and `NOTIFICATION`s, and for the values of `ATTRIBUTE`s are defined using the ASN.1 notation [ASN1].
- The `BEHAVIOUR` template, that is used for providing details of all operations that instances of the class may perform, and more generally, to specify every feature that is not possible to specify with the other templates. 'Behaviours' are associated to packages, attributes, actions and notifications. The only field in the `BEHAVIOUR` template consists of a free-string.

A major shortcoming of the GDMO language is that the only notation provided for defining the behaviour of a managed object is natural language. The use of natural language for describing behaviour can produce incomplete, inaccurate and ambiguous specifications, and don't allow any kind of automatic processing of the specification. Hence, it is no longer regarded appropriate for describing managed object behaviour. It is for this reason that the ISO started to search for formal description techniques for specifying managed object behaviour [ISO94].

2.3 Expected benefits from formalizing Managed Object Behaviour

While GDMO currently prescribes the use of natural language for managed object behaviour descriptions, the ISO has now acknowledged that formal description technique would be more appropriate. The additional rigor associated with FDTs should result in a general improvement in the quality of managed object definitions. Formalization of managed object behaviour should be beneficial, with consequent improvements in the internal consistency of managed object definitions.

Formal specifications can also provide an excellent basis to implement managed objects, and to produce their conformance test. Productivity and quality gains can be made through the use of techniques to automatically produce implementations from formal specifications. Similarly, the use of existing techniques for generating test cases from formal specifications would improve productivity and quality of conformance testing.

In addition to these traditional benefits expected from formal methods, early simulation

of managed object specifications will be possible and that will help managed objects definition and therefore ameliorate their specification.

3 FORMALIZING MANAGED OBJECT BEHAVIOUR

3.1 Current solutions

Since the standardization organizations ITU-T and ISO haven't recommended any formal description technique for managed object behaviour formalization, managed object specifiers with behaviour formalization needs had turned to existing FDTs. Several experiments have been led using Z, LOTOS, ESTELLE, SDL or CRS (see [ISO94], [Fes94], [Dub94]). The main problem they faced is that these languages were not designed for managed object behaviour specification.

I will not detail here the merits and drawbacks of each of these techniques, but I can list some common stumbling blocks :

- FDTs can't easily deal with ASN.1 types and values. Two solutions have been found : translate ASN.1 types and values of the GDMO specification in types and values of the FDT or extend the FDT to allow ASN.1 types and values expression.
- It is almost impossible to mix an FDT specification with the GDMO specification : variables of the GDMO specification are not variables of the FDT specification. The main solution is to translate (automatically or not) the whole specification written in GDMO into the same specification but written using the FDT notation.
- Also, contrary to the semi-formal description technique GDMO, generic FDTs don't offer tool support for the needs of network management : managed object simulation, automatic implementation and automatic generation of test sets for conformance testing.
- And lastly, since using or reading these FDTs need to have a good grasp of theoretical computer science or mathematical logic, a large range of knowledge is necessary to apply them to network management.

Taking in account that a managed object specification has at least two goals : make TMN[‡] actors agree (normalization activity) and specify implementations; at the present time, the most reliable solution is to support three specifications :

- A specification in GDMO, for normalization and tools' use, with no well defined managed object behaviour.
- A specification in an exotic FDT, for fulfilling managed object behaviour formalization needs, but with poor tool support and not well accepted by the specification users. Also, since this specification duplicates descriptions of the GDMO specification, the coherence is not guaranteed and cannot be easily maintained.
- A specification of the managed object behaviour in natural language for readers with no mathematical background. This specification duplicates the precedent specification.

[‡]Telecommunication Management Network

In concrete terms, this kind of solution is heavy and only conceivable for small problems.

Starting from this analysis, it is clear that it is necessary to build an *ad hoc* formalism to fulfill the needs of managed object behaviour formalization. Since GDMO is incomplete and does not satisfy these needs, it seems logical to extend GDMO to give it the ability of supporting formal behaviour specification.

3.2 Extending GDMO

Requisites

To be useful an extension of GDMO (in the remainder of the paper this extension will be mentioned as GDMO$^+$) must satisfy some requisites:

- GDMO$^+$ needs to be upward compatible with GDMO. That means that GDMO$^+$ should be a syntactic and semantic superset of GDMO, i.e. every specification written in GDMO is a GDMO$^+$ specification with the same meaning.
- Also, the new features of GDMO$^+$ should not be redundant with GDMO features.
- To be well integrated and ensure the homogeneity of the specification the additional syntax should have, as far as possible, the same *look and feel* than GDMO and/or ASN.1 notations.
- Concepts and terminology of GDMO$^+$ should be similar to GDMO ones and as close as possible to those used by its potential users.

If these requirements are fulfilled, it is reasonable to think that GDMO$^+$ will be easily accepted by the Network Management community. Also, it will be possible to upgrade tools that process GDMO specifications to allow the processing of GDMO$^+$ specifications.

Behaviours that need to be formalized

Managed object behaviours that need to be specified are of two kinds, one kind of them depends on how managed resources behave, these cannot be formalized without formalizing the resources. The way managed objects are related with the resource they model is not specified using GDMO, it should either be specified using GDMO$^+$ without changing the purpose of the notation. In a first stage it seems ambitious enough to give GDMO$^+$ the ability of expressing the behaviours that only depends on managed objects variables. This other kind of behaviours are those GDMO$^+$ will deal with.

Mainly the behaviours to be specified are:

- Extensions or precisions on how a managed object has to react on manager requests. Requests can be : to set or get an attribute value, to create or delete an instance of a managed object class, or to perform an action.
- Static or dynamic invariants of a managed object or of a set of managed objects must maintain, i.e. reaction on internal events or conditions that have to be true anyhow.
- Conditions and modalities of notification emissions.

To specify these behaviours, several templates are being added in GDMO$^+$.

New templates

The idea is to provide several specialized templates for specifying behaviours in addition to the GDMO BEHAVIOUR template : PACKAGE BEHAVIOUR, ACTION BEHAVIOUR, ATTRIBUTE BEHAVIOUR, NOTIFICATION BEHAVIOUR, etc...

PACKAGE BEHAVIOUR template is for specifying invariants that concern a package.

ACTION BEHAVIOUR template is for specifying how the managed object must behave when an action is invoked on it.

ATTRIBUTE BEHAVIOUR template is for specifying how the managed object must behave when an operation is performed on an attribute (get or set value).

Et cetera.

These templates are to be used in the same manner as the GDMO BEHAVIOUR template, however with some trivial restrictions : for example a PACKAGE BEHAVIOUR template cannot be referenced in an ATTRIBUTE template ...

In the next section one of these templates, the NOTIFICATION BEHAVIOUR one, is described in more detail. On purpose this description imitates the style of the standard *CCITT Recommendation X.722 (1992) | ISO/IEC 10165-4 :1992 "Guidelines for the Definition of Managed Objects"* [GDMO].

4 NOTIFICATION BEHAVIOUR TEMPLATE

4.1 Overview

This template is used to define behavioural aspects of notifications. It can be only referenced by packages and notifications. The notification behaviour template is intended to allow behaviour formalization, but when it is not possible or when the expected gains are insufficient it allows semi-formal descriptions; the INFORMALLY keyword is intended for this use.

4.2 Template structure

```
<notification-behaviour-label> NOTIFICATION BEHAVIOUR
     [DESCRIPTION delimited-string ;
     ]
     [NOTIFICATIONS <notification-label> [, <notification-label>]*;
     ]
     [EMITTED [FOR event-description [, event-description]* ]
              [WITH conditional-value-assignment
                        [, conditional-value-assignment]* ]
        [ALSO [FOR event-description [, event-description]* ]
              [WITH conditional-value-assignment
                        [, conditional-value-assignment]* ]
        ]*;
     ]
     [ADDITIONAL CONSTRAINTS condition-description [, condition-description]*;
     ]
```

```
supporting productions

conditional-value-assignment   -> if-construct | value-assignment
if-construct                   -> IF condition-description
                                        THEN conditional-value-assignment
                                             [, conditional-value-assignment]*
                                        [ELSE conditional-value-assignment
                                             [, conditional-value-assignment]*]
                                   ENDIF

value-assignment          -> EVENT-INFO = value-description |
                             <field-name> = value-description
event-description         -> formal-event-description
                                        [INFORMALLY delimited-string] |
                             INFORMALLY delimited-string
value-description         -> value-reference |
                             formal-value-description
                                        [INFORMALLY delimited-string] |
                             INFORMALLY delimited-string
condition-description     -> formal-condition-description
                                        [INFORMALLY delimited-string] |
                             INFORMALLY delimited-string
```

Note. `formal-event-description`, `formal-value-description`, `formal-condition-description` syntaxes have to be defined precisely (in the future). However it is possible to say several things about these expressions :

- Expressions with syntaxes `formal-event-description` , `formal-condition-description` when they are calculable, must have a boolean value (i.e. true or false) as result.
- Expressions with syntax `formal-value-description`, when they are calculable, must have as result an ASN.1 value.
- When a notification, to which this behaviour is applied, is active (i.e. instantiated, included in a package present in a managed object), the expressions `formal-event-description` in `FOR` sub-clauses must be calculable.
- Also, when preconditions are fulfilled for the emission of the notification, the expression `formal-condition-description` present in the `ADDITIONAL CONSTRAINT` clause and all the expressions in `WITH` sub-clauses not preceded by a `FOR` sub-clause must be calculable.
- And finally, when an event described in a `FOR` sub-clause occurs, the expressions in the immediately following `WITH` sub-clause (if any) must be calculable.

4.3 Supporting definitions

`DESCRIPTION delimited-string`

This construct allows a description of the purpose of the notification behaviour definition; e.g., '*Reporting high-level rate of errors in telecommunication systems*'. No restrictions are placed on the character set used to represent the `DESCRIPTION` and no further structure within the `DESCRIPTION` is defined.

This construct shall not be used as a means for defining behavioural aspects of notifications.

`NOTIFICATIONS <notification-label> [, <notification-label>]*`

The **notification-labels** allow the notification behaviour to be associated with notifications. When the notification behaviour is included in a package definition and when the package is instantiated, the notification behaviour applies to all the instantiated notifications referenced in its **NOTIFICATIONS** clause. When the notification behaviour is included in a notification definition, the notification behaviour applies to the notification, regardless of its **NOTIFICATIONS** clause.

`EMITTED [FOR ...] [WITH ...] [ALSO [FOR ...] [WITH ...] ]*`

`FOR event-description [, event-description]*`

The **event-descriptions** describe events that activate the notifications which are associated with the notification behaviour.

`WITH conditional-value-assignment [, conditional-value-assignment]*`

The **conditional-value-assignments** must be evaluated if any of the events described in the preceding **FOR** sub-clause occurs; if there is no preceding **FOR** sub-clause, the **conditional-value-assignments** must be evaluated if any of the associated notifications is activated.
A **conditional-value-assignment** is evaluated as follows :

When it is an **if-construct**,
if the **condition-definition** of the **IF** sub-clause is evaluated to true then the **conditional-value-assignments** of the **THEN** sub-clause are evaluated, otherwise the **conditional-value-assignments** of the **ELSE** sub-clause are evaluated.
When it is a **value-assignment**,
the whole notification information (keyword **EVENT-INFO**) or the field of the notification information (identify by **field-name**) must be equal to the value issue from the evaluation of the **value-description** expression.

Note. **field-name** is used here in the same way as in the **AND ATTRIBUTES IDS** sub-clause of the notification template [GDMO].

`ADDITIONAL CONSTRAINTS condition-description [, condition-description]*`

The **condition-descriptions** describe conditions that must be true when a notification is emitted.
These conditions are for describing post-conditions related to notification, i.e. conditions on the notification information field. They should not be used for describing events that activate the notification.

5 FORMAL SEMANTICS OF NOTIFICATION BEHAVIOUR TEMPLATE

5.1 Introduction

To be well defined, a language needs a syntax but also a semantics (see [Sch88]). When the purpose of the language is formalization, formal semantics is imperative. This formal semantics will help in defining GDMO$^+$ as a non-ambiguous language. Since formal specifications are closer to programs than informal ones, it will be also helpful in implementing tools for processing GDMO$^+$.
The language chosen for formalizing is the Z notation (for more information on Z see [Hay87, Spi92]). Reasons and relevance of this choice are explained in [Kel94]. Formalization in Z of GDMO semantics can be found in [Kel95] and ASN.1 formal semantics in [Dub95]. This section shows how the semantics of GDMO$^+$ can be formalized.
The remainder of this section presents the semantics of the notification behaviour template and how it describes behaviour of managed objects. The following Z schemas intend to be examples and therefore are not complete : they focus on what is important to the understandability of the whole example. These schemas also refer to definitions that can be found in [Kel95].

5.2 Static semantics of the notification behaviour template

The following Z schema describes the semantics of the notification behaviour template. It is possible to associate, to each valid notification behaviour, an instance of the *Notification-Behaviour* schema. Two notification behaviours have the same meaning when they are associated to the same schema instance.
This schema describes what constitute a notification behaviour.

NotificationBehaviour

$$
\begin{array}{l}
label : TEMPLATE\_LABEL \\
notifications : \mathbb{F}\ TEMPLATE\_LABEL \\
emitted : \mathbb{F}\ EmissionCondition \\
constraints : \mathbb{F}\ CONDITION
\end{array}
$$

Understanding what the variables of this schema represent is simple, since they have the same names as the clauses of the template.

TEMPLATE_LABEL is the type for naming templates in GDMO, *CONDITION* describes semantics of `condition-description` syntax, and *EmissionCondition* is detailed in the following schema.

EmissionCondition

$$
\begin{array}{l}
eventCVA : EVENT \leftrightarrow ConditionalValAssign \\
condValAssign : \mathbb{F}\ ConditionalValAssign \\
\hline
condValAssign = \varnothing \Leftrightarrow eventCVA \neq \varnothing
\end{array}
$$

The expression [1] says that when one of the variables, *eventCVA* or *condValAssign*, is empty the other one is not. An *EmissionCondition* is a set of *ConditionalValAssign* driven or not by a set of *EVENT* (`FOR` clause of the syntax).

EVENT describes semantics of `event-description` syntax.

ConditionalValAssign describes how a `conditional-value-assignment` piece of syntax has to be interpreted. The `conditional-value-assignment` concrete syntax can be modeled in an abstract syntax (for considerations on abstract and concrete syntaxes see [Sch88]) as follows :

$$\begin{aligned} &CVAsyntax ::= valueAssign\langle\!\langle \mathbb{F}\ VALUE\_ASSIGNMENT \rangle\!\rangle \\ &\quad \mid ifConstruct\langle\!\langle CONDITION \times CVAsyntax \times CVAsyntax \rangle\!\rangle \end{aligned}$$

where *VALUE_ASSIGNMENT* is the semantics of `value-assignment` expressions.

CVAsyntax describes recursive structures, a *CVAsyntax* is either a set of *VALUE_ASSIGNMENT* or a triplet in which the first element is the condition of the `IF` and the two following ones are the *CVAsyntax*es of respectively the `THEN` and the `ELSE` clauses.

ConditionalValAssign schema shows how this abstract syntax is interpreted or 'precompiled' :

ConditionalValAssign

$$\begin{aligned} &syntax : CVAsyntax \\ &reducing : \mathrm{seq}_1(CVAsyntax \nrightarrow (\mathbb{F}\ CONDITION \times \mathbb{F}\ CONDITION)) \\ &nonRecusiveCVA : (\mathbb{F}\ VALUE\_ASSIGNMENT) \nrightarrow (\mathbb{F}\ CONDITION \times \mathbb{F}\ CONDITION) \\ &atomCVA : VALUE\_ASSIGNMENT \rightarrow (\mathbb{F}\ CONDITION \times \mathbb{F}\ CONDITION) \\ \hline &head\ reducing = \{(syntax \mapsto (\varnothing, \varnothing))\} \\ &\mathrm{dom}(last\ reducing) \subseteq \mathrm{ran}\ valueAssign \\ &nonRecusiveCVA = \{va : CVAsyntax;\ pos, neg : \mathbb{F}\ CONDITION \mid \\ &\qquad va \mapsto (pos, neg) \in last\ reducing \bullet (valueAssign^{-1}\ va) \mapsto (pos, neg)\} \\ &\forall n : 2\,..\,\#reducing \bullet reducing\ n = \\ &\quad \{cva : CVAsyntax;\ pos, neg : \mathbb{F}\ CONDITION \mid \\ &\qquad cva \in \mathrm{ran}\ valueAssign \land cva \mapsto (pos, neg) \in reducing\ (n-1) \bullet cva \mapsto (pos, neg)\} \cup \\ &\quad \bigcup\{cva : CVAsyntax;\ pos, neg : \mathbb{F}\ CONDITION \mid \\ &\qquad cva \in \mathrm{ran}\ ifConstruct \land cva \mapsto (pos, neg) \in reducing\ (n-1) \bullet \\ &\qquad\quad (\mu\, cond : CONDITION;\ cval, cva2 : CVAsyntax \mid \\ &\qquad\qquad (cond, cval, cva2) = ifConstruct^{-1} cva \bullet \\ &\qquad\qquad\quad \{cva2 \mapsto (pos, neg \cup \{cond\}), cval \mapsto (pos \cup \{cond\}, neg)\})\ \} \\ &atomCVA = \bigcup\{vaSet : \mathbb{F}\ VALUE\_ASSIGNMENT;\ pos, neg : \mathbb{F}\ CONDITION \mid \\ &\quad vaSet \mapsto (pos, neg) \in nonRecusiveCVA \bullet \{va : vaSet \bullet va \mapsto (pos, neg)\}\} \end{aligned}$$

[1] *syntax* is the *CVAsyntax* to be interpreted.

[2] *reducing* is a sequence that starts with *syntax* and ends with a non recursive structure. In the doublet of sets of conditions, the first one is a set of positive conditions, i.e. conditions that must true for *CVAsyntax* expression evaluation, the second one a set of negative conditions.

[3] *nonRecursiveCVA* is the result of *reducing*.

[4] *atomCVA* is a rewriting of *nonRecursiveCVA* in which *VALUE_ASSIGNMENT*s are associated one to one to condition set doublets.

Expression [5] initiates recursion and [6] is the ending condition.

nonRecursiveCVA is 'calculated' in expression [7] using the last term of *reducing*.

Predicate [8] expresses *reducing n* regarding *reducing* $(n-1)$.

And in [9] *atomCVA* is expressed regarding *nonRecursiveCVA*.

The *NotificationBehaviour* expresses what a notification behaviour template means, but since several notification behaviour templates may be associated to a particular instance of a managed object notification, it is necessary to describe how 'notification behaviours' combines to build up the 'behaviour' of a notification instance.
This is done in the next schema :

NotificationInstanceBehaviour

$$
\begin{array}{l}
behaviours : \mathbb{F}\ NotificationBehaviour \\
eventCVA : EVENT \leftrightarrow ConditionalValAssign \\
condValAssign : \mathbb{F}\ ConditionalValAssign \\
constraints : \mathbb{F}\ CONDITION \\
atomCVA : VALUE\_ASSIGNMENT \leftrightarrow (\mathbb{F}\ CONDITION \times \mathbb{F}\ CONDITION) \\
\hline
eventCVA = \bigcup\{beh : behaviours \bullet \bigcup\{ec : beh.emitted \bullet ec.eventCVA\}\} \\
condValAssign = \bigcup\{beh : behaviours \bullet \bigcup\{ec : beh.emitted \bullet ec.condValAssign\}\} \\
constraints = \bigcup\{beh : behaviours \bullet beh.constraints\} \\
atomCVA = \bigcup\{cva : condValAssign \bullet cva.atomCVA\}
\end{array}
$$

Since there is no notion of order in notification behaviour templates, they combine very well : Merging notification behaviour templates is merely doing a set union on their 'properties'. Taking into account that, in GDMO, managed object classes are build up by combining templates, this is an important result to ensure continuity between GDMO$^+$ and GDMO.

5.3 Managed Object static semantics

Notifications and therefore notification behaviours are part of the definition of managed objects, and cannot operate alone. A managed object does not implement all the features of its managed object class. The implemented features are grouped here in a so called real class described by the following schema :

RealClass

$$
\begin{array}{l}
class : ManagedObjectClass \\
packages : \mathbb{F}\ Package \\
notifications : \mathbb{F}\ Notification \\
notifBeh : Notification \twoheadrightarrow NotificationInstanceBehaviour \\
\hline
packages \subseteq class.packages \\
class.mandatoryPackages \subseteq packages \\
notifications = \bigcup\{pkg : packages \bullet \mathrm{dom}\ pkg.notifications\} \\
notifBeh = \{notif : notifications;\ nib : NotificationInstanceBehaviour \mid \\
\qquad nib.behaviours = notif.behaviour.notification \cup \\
\qquad \bigcup\{pkg : packages \bullet \{nb : pkg.behaviour.notification \mid notif.label \in nb.notifications\}\}\}
\end{array}
$$

[1] *class* is the managed object class the managed object is an instance of.
[2] *packages* are the packages of *class* that are actually instantiated.
[3] *notifications* are the notifications referenced in the *packages*.
[4] *notifBeh* associates to each notification of *notifications* a *NotificationInstanceBehaviour*.
notifBeh and *notifications* depend on which packages are actually present in *RealClass*.

The *ObjectInstance* schema models what a managed object is :

ObjectInstance

$$
\begin{array}{l}
RealClass \\
relativeDistinguishedName : AttributeValueAssertion \\
value : Attribute \nrightarrow VALUE \\
\hline
\mathrm{dom}\ value \subseteq \mathrm{dom}\ attributes \\
\forall\ att : Attribute \mid att \in \mathrm{dom}\ value \bullet value(att)\ \underline{ofType}\ att.syntax \\
relativeDistinguishedName \in value
\end{array}
$$

An *ObjectInstance* of a *RealClass* is distinguished among from other *ObjectInstances* of the same *RealClass* by its name (*relativeDistinguishedName*) and the value of its attributes.

5.4 Semantics of the Event-Report CMIS operation

The semantics of the notification behaviour template will be achieved saying the way it influence the behaviour of the management system. The notification behaviour template allows to specify in which conditions and with which value a CMIS Event-Report is emitted, so semantics of the Event-Report CMIS operation and its relations with notification behaviour definitions have to be specified, this is the aim of this subsection.

The data carried by the protocol in an Event-Report is defined in [CMIP] using the ASN.1 notation, in Z this definition can be :

EventReportArgument

$$
\begin{array}{l}
class : ObjectIdentifierValue \\
objectName : DistinguishedName \\
eventType : ObjectIdentifierValue \\
eventInfo : VALUE
\end{array}
$$

The definitions of the Z types used in this schema are not detailed here and can be found in [Kel95].

The behaviour of the CMIS operation is modelled by the *EventReport* Z schema which is a compound Z schema defined as follows :

$$EventReport \mathrel{\widehat=} GenericEventReport \lor EventReportWithDefinedBehaviour$$

An Event-Report can be either a *GenericEventReport*, if no specific behaviour is specified using notification behaviour templates, or an *EventReportWithDefinedBehaviour*, if a specification of behaviour exists.

The *GenericEventReport* schema models the Event-Report CMIS operation, as it is defined in [CMIS], it specifies which are the variables necessary to perform the operation and the conditions that qualify the operation.

GenericEventReport

ΞMIB
argument! : *EventReportArgument*
object : *ObjectInstance*
notif : *Notification*

---

argument!.class = *object.class.registeredAs*
argument!.objectName = *distinguishedName object*
argument!.eventType = *notif.registeredAs*
argument!.eventInfo ofType notif.withInformationSyntax
notif ∈ *object.notifications*

where ΞMIB indicates that all the variables of *MIB* are relevant for operation performing, and that these variables should be constant during the operation performing.
Note. The MIB (Management Information Base, defined in [MIM]) is the 'set' of all managed objects present at a given management interface.
And where *argument!* is the information carried by the event report; *argument!* can be viewed as the 'result' of the operation.

The *EventReportWithDefinedBehaviour* schema includes evidently the *GenericEventReport*, the only additional variable is the 'behaviour' deduced from the GDMO$^+$ definitions :

EventReportWithDefinedBehaviour

GenericEventReport
nib : *NotificationInstanceBehaviour*

---

notif ∈ dom *object.notifBeh*
nib = *object.notifBeh notif*
∀ *cva* : *nib.atomCVA* • *CheckCVA*(*argument!.eventInfo*, *cva*) = *yes*
∀ *ev* : dom *nib.eventCVA* | *EvalEvent ev* = *yes* •
  ∀ *cva* : ⋃{*im* : *nib.eventCVA*(| {*ev*} |) • *im.atomCVA*} •
    CheckCVA(*argument!.eventInfo*, *cva*) = *yes*
∀ *cond* : *nib.constraints* • *EvalCond cond* = *yes*

[1] expresses the fact that the operation *EventReportWithDefinedBehaviour* is performed if a behaviour is defined for the notification within the object.
[2] gives a value to *nib*.
[3] verifies that the information carried by the notification is accorded with the `conditional-value-assignments` that applies to all events.
[4] verifies that if event occurs, the `conditional-value-assignments` that applies to this event are correctly evaluated.
And [5] checks that the additional constraints are fulfilled.

The *CheckCVA* function verifies the accordance of a *VALUE* with a *VALUE_ASSIGNMENT* regarding the positive and negative conditions associated with the *VALUE_ASSIGNMENT*. *CheckCVA* is defined as follows :

$$CheckCVA : VALUE \times (VALUE\_ASSIGNMENT \times (\mathbb{F}\ CONDITION \times \mathbb{F}\ CONDITION)) \pfun BOOLEAN$$

$$\forall val : VALUE;\ va : VALUE\_ASSIGNMENT;\ condSet : (\mathbb{F}\ CONDITION \times \mathbb{F}\ CONDITION) \bullet CheckCVA(val, (va, condSet)) = yes \Leftrightarrow (\forall cond : first\ condSet \bullet EvalCond\ cond = yes) \land (\forall cond : second\ condSet \bullet EvalCond\ cond = no) \Rightarrow EvalVA(val, va) = yes$$

Since *VALUE_ASSIGNMENT*, *CONDITION* and *EVENT* Z types are not defined yet, it is not possible to say how to evaluate them, therefore *EvalVA*, *EvalCond* and *EvalEvent* are not defined and only their types can be furnished :

$$EvalVA : VALUE \times VALUE\_ASSIGNMENT \pfun BOOLEAN$$
$$EvalCond : CONDITION \pfun BOOLEAN$$
$$EvalEvent : EVENT \pfun BOOLEAN$$

5.5 Tools that can be developed

The formalization of GDMO$^+$ and CMIS semantics will be helpful in implementing automated tools for generating code canvas for managed objects implementation.
Also, as using GDMO$^+$ will make managed objects behaviour formalizable, it is possible to envisage managed objects simulators and formal semantics of GDMO$^+$ will be useful for the design and implementation of these simulators. Simulators will feed GDMO$^+$ specifications and help validating managed object definitions.

6 EXAMPLE OF USE OF THE NOTIFICATION BEHAVIOUR TEMPLATE

This example intends to illustrate the use of the notification behaviour template. For reasons of conciseness the example is not based on a real case. It partially specifies for a particular application the behaviour of a generic notification defined in [DMI]. The `qualityOfServiceAlarm` notification is defined as follows :

```
qualityOfServiceAlarm NOTIFICATION
     BEHAVIOUR qualityOfAlarmBehaviour;
     WITH INFORMATION SYNTAX Notification-ASN1Module.AlarmInfo
          AND ATTRIBUTE IDS
               probableCause              probableCause,
               specificProblems           specificProblems,
               perceivedSeverity          perceivedSeverity,
               backedUpStatus             backedUpStatus,
               backUpObject               backUpObject,
               trendIndication            trendIndication,
               thresholdInfo              thresholdInfo,
               notificationIdentifier     notificationIdentifier,
               correlatedNotifications    correlatedNotifications,
               stateChangeDefinition      stateChangeDefinition,
               monitoredAttributes        monitoredAttributes,
               proposedRepairActions      proposedRepairActions,
               additionalText             additionalText,
```

```
            additionalInformation       additionalInformation;
REGISTERED AS smi2Notification 11;
```

The data type of the notification information is the ASN.1 type `AlarmInfo` defined in `Notification-ASN1Module` of DMI as follows :

```
AlarmInfo ::= SEQUENCE {
     probableCause                  ProbableCause,
     specificProblems          [1]  SpecificProblems OPTIONAL,
     perceivedSeverity              PerceivedSeverity,
     backedUpStatus                 BackedUpStatus OPTIONAL,
     backUpObject              [2]  ObjectInstance OPTIONAL,
     trendIndication           [3]  TrendIndication OPTIONAL,
     thresholdInfo             [4]  ThresholdInfo OPTIONAL,
     notificationIdentifier    [5]  NotificationIdentifier OPTIONAL,
     correlatedNotifications   [6]  CorrelatedNotifications OPTIONAL,
     stateChangeDefinition     [7]  AttributeValueChangeDefinition OPTIONAL,
     monitoredAttributes       [8]  MonitoredAttributes OPTIONAL,
     proposedRepairActions     [9]  ProposedRepairActions OPTIONAL,
     additionalText                 AdditionalText OPTIONAL,
     additionalInformation     [10] AdditionalInformation OPTIONAL }
```

To better understand the example it is necessary to detail the `AlarmInfo` type by giving the definition of the `MonitoredAttributes` and `PerceivedSeverity` types :

```
MonitoredAttributes ::= SET OF Attribute

PerceivedSeverity ::= ENUMERATED {
     indeterminate (0), -- used when it is not possible to assign the following values
     critical (1), major (2), minor (3), warning (4), cleared (5) }
```

The behaviour template associated to the notification specifies almost nothing, it can be regarded as a comment :

```
qualityOfServiceAlarmBeahaviour BEHAVIOUR
   DEFINED AS
      "This notification type is used to report a failure in the quality of service of the managed object.";
```

The following template details the behaviour of a managed object offering `qualityOf-ServiceAlarm` notification service. It must be included in a package of the managed object to be taken into account. It specifies conditions that cause the emission of the notification and the values the notification must carry.

```
myQualityOfServiceAlarmBehaviour NOTIFICATION BEHAVIOUR
     DESCRIPTION "This is an example of how to use NOTIFICATION BEHAVIOUR template";
     NOTIFICATIONS
          "CCITT Rec. X.721 (1992) | ISO/IEC 10165-2 : 1992":qualityOfServiceAlarm;
     EMITTED
          FOR INFORMALLY "errorRate attribute > maximum allowed (= 10) "
               WITH monitoredAttributes =
```

```
                INFORMALLY "singleton containing errorRate attribute OID",
          IF INFORMALLY "10 < errorRate =< 20" THEN
               perceivedSeverity = warning
                    -- Remark: formal syntax is not defined yet but this is an
                    -- example of how it could look
          ELSE IF INFORMALLY "20 < errorRate =< 30 " THEN
               perceivedSeverity = minor
          ELSE IF INFORMALLY "30 < errorRate =< 40 " THEN
               perceivedSeverity = major
          ELSE IF INFORMALLY "40 < errorRate " THEN
               perceivedSeverity = critical
          ELSE perceivedSeverity = indeterminate
          ENDIF ENDIF ENDIF ENDIF;
ADDITIONAL CONSTRAINTS
     INFORMALLY "field notificationIdentifier must be always present",
     INFORMALLY "field perceivedSeverity must be different than cleared (not used)";;
```

7 CONCLUSIONS

This paper has shown the problems found while using existing FDTs for formalizing managed object behaviour and has made emerge the need of building an adapted *ad hoc* formalism. As it is clear now that it is possible to extend GDMO, this seems to be the right solution since it make possible the reuse of current managed object specifications and also the upgrade of existing tools that process GDMO descriptions.
Future work will have to consider the definition of all the needed templates and the construction of an extension of ASN.1 with operators on ASN.1 values. Also formalization of the language semantics will help in maintaining coherence and in implementing tools as specification checkers and simulators, and compilers for generating code canvases and conformance test sets.

Acknowledgments
I would like to thank my colleagues Olivier Dubuisson and Michel Chabaud for their valuable suggestions for the work reported in this paper.

REFERENCES

[Kel94] J. Keller, O. Dubuisson (1994) *Formal Description of OSI Management Information Structure as a Prerequisite for Formal Specifications of TMN interfaces* Proceedings of IS&N'94. Lecture Notes in Computer Science 851. Springer-Verlag.

[Kel95] J. Keller, O. Dubuisson, R. Danion (Jan. 1995) *Description Formelle et Validation de Documents GDMO* France Telecom CNET Internal Report, NT/LAA/EIA/40.

[Dub94] O. Dubuisson (Jan. 1994) *Techniques de Description Formelle et Comportements en GDMO: Etude de l'Existant* France Telecom CNET Internal Report, NT/LAA/EIA/8.

[Dub95] O. Dubuisson (Feb. 1995) *Description Formelle en Z de la Sémantique Statique de la Notation ASN.1* France Telecom CNET internal document.

[Fes94] O. Festor (Dec. 1994) *Formalisation du comportement des objets gérés dans le cadre du modèle OSI* PhD Thesis, Université de Nancy I.

[ISO94] ISO/IEC JTC 1/SC 21/WG 4 N8808 (July 1994) *Working Draft on the Use of FDTs for the Specification of the Behaviour of Managed Objects* .

[MIM] CCITT Recommendation X.720 (1992) | ISO/IEC 10165-1 :1992 *Structure of management information : Management information model.*

[GDMO] CCITT Recommendation X.722 (1992) | ISO/IEC 10165-4 :1992 *Structure of management information : Guidelines for the definition of managed objects.*

[DMI] CCITT Recommendation X.721 (1992) | ISO/IEC 10165-4 :1992 *Structure of management information : Definition of management information.*

[CMIS] CCITT Recommendation X.710 (1991) | ISO/IEC 9595 :1990 *Common management information service definition.*

[CMIP] CCITT Recommendation X.711 (1991) | ISO/IEC 9596-1 :1991 *Common management information protocol - Part 1 : Specification.*

[ASN1] CCITT Recommendation X.208 (1988) | ISO/IEC 8824 :1988 *Information processing systems - Open System Interconnection - Specification of Abstract Syntax Notation One (ASN.1).*

[Spi92] Spivey, J. M. (1992) *The Z Notation : A Reference Manual* . Prentice Hall.

[Hay87] I. Hayes (1987) *Specification Cases Studies.* Series in Computer Science. Prentice-Hall International.

[Sch88] D. A. Schmidt (1988) *Denotational Semantics : A Methodology for Language Development.* Wm. C. Brown Publishers.

BIOGRAPHY

Joaquín Keller is currently researcher in the Artificial Intelligence Applications Department, Centre National d'Etudes des Télécommunications, France Télécom. He received the M.Sc. degree (French DEA) in mathematical logic from Université Paris VII Denis Diderot, France, in 1990. Mr. Keller participate in AFNOR (Association Fran caise de Normalisation) and ISO normalization works.