Application of SDL-92 for the specification of OSI Management Systems

M. Rodríguez, R. Calmeau, E. Fernández
Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática
E.T.S.I. de Telecomunicación. Universidad de Valladolid
Campus Miguel Delibes S/N
47011 Valladolid
SPAIN
{manrod,calher,eusebio}@gbien.tel.uva.es

Abstract

Network resources in OSI network management are modeled using managed objects. These objects are specified by the semi-formal language GDMO that describes their behaviour (using natural language) and static structures. This drawback in the behaviour specification can be solved using formal methods. A mapping method from GDMO information models to SDL-92 specifications based on the direct translation of the GDMO template into SDL constructs is presented in this paper. This solution simplifies the automatic translation process between both languages and solves some of the problems because of the differences in the object oriented concepts involved. An example of the proposed translation method and its application to OSI management system specifications are also presented.

Keywords

OSI Management, Management Information Model, GDMO, SDL-92, ASN.1

1. Introduction

An important task in the development of OSI management applications is the specification of the management characteristics of network resources (known as management information). To structure the management information, ITU-T[1] has developed the Management Information Model (MIM), [1], which defines managed objects as entities which represent network resources from a management point of view. These objects are described by the standard notation GDMO, [2], which allows the description of the managed objects static structure and their behaviour, though behaviour is usually specified by using natural language. This fact may introduce ambiguities in the understanding of the behaviour and also limits the possibilities of managed object and agent simulation. The use of a formal description technique (FDT) to describe managed object behaviour can avoid these drawbacks. Among the different FDTs, SDL-92, [3], has been selected due to its object oriented capabilities (similar to the GDMO ones) and its powerful tool support.

A method to obtain a formal specification in SDL-92 of managed objects and their interface with the agent using their GDMO definition is presented in this paper. A model of the agent and its interfaces with a manager and managed objects is also presented. The method uses the definition of ASN.1, [4], data types in SDL specifications as defined in the Z.105 ITU-T Recommendation (SDL combined with ASN.1), [5].

This paper is organized as follows: Section 2. describes the proposed method to translate GDMO management information models into the corresponding SDL specification. The suggested SDL model for a generic managed object and agent is also presented in this section. Section 3. presents a example of the translation algorithm. Finally, section 4. offers some conclusions and indicates the open issues for future work.

2. Managed System Specification in SDL

2.1 Related Work

Several proposals have been made for the formal specification of information models by now. One of them focuses on extending GDMO with behavioural notations, obtaining a formal specification of the behaviours. This approach (GDMO+ [6], [7]) proposes several extensions to GDMO behaviour definition based on pre- and post-conditions.

The other approaches combine several FDTs with GDMO. Mazaher and Moller-Pedersen, [8], have proposed a set of translation rules from a GDMO specification to an SDL-92 formal model, including data types and behaviour definitions. This method is based on mapping: i) managed object classes into SDL-92 process types;

[1] International Telecommunications Union

ii) attributes into variables of process; and iii) operations (predefined, actions, and notifications) into remote procedures and signals. With this mapping, every operation request on a managed object is translated into a remote procedure call. As every managed object class will have different operations, there will be many possible different remote procedure calls. This fact complicates the design of agent and manager processes. A mapping based on signal exchange and local procedures will be more appropriated, as there will be only one signal for each kind of management operation (the signal parameters would select the specific operation).

Bartocci, Larini and Romellini, [9], have also proposed a mapping based on ASN.1 data types translation into SDL ADTs[2] and the definition of an SDL process type to represent the managed object behaviour. This work was continued by Bartocci and Ferrrero in [10] and [11]. In the last two papers, they propose the introduction of SDL procedures in the behaviour templates. A model of a managed object and the interface with its agent (based on signal exchange) is also presented. This method does not clearly show how to obtain, automatically, a SDL specification of a managed system from a GDMO information model. Support of multiple inheritance is not explained either.

Another work of interest is the method proposed by Samir Tata, Laurent Andrey and Olvier Festor, [12], which takes advantage of the new facilities in the Z.105 recommendation for the use of ASN.1 together with SDL abstract data types. This method is based on the previous ones, but the differences are not clearly explained.

Our translation algorithm is mainly based on the work of Bartocci and Ferrero, but introducing some changes: i) the use of initializing procedures to automate the translation algorithm and to support conditional packages; ii) the use of procedures for combining properties to support multiple inheritance of static properties; and finally, iii) some minor changes in the ASN.1 data type definitions.

The resulting algorithm is presented in the following sections.

2.2 GDMO-SDL translation

The proposed approach is based on the direct translation of elements in the GDMO templates into SDL constructs and/or ASN.1 data types definitions. For each element that can be a part of a managed object (parameter, attribute, attribute group, action and notification) an ASN.1 data type is defined. Every data type (named as *generic data type*) has the capability of storing the static properties of one of those elements. These data types will be used in the declaration of variables in the SDL model of a managed object.

[2]abstract data type

Because of the definitions of parameters, attributes, attribute groups, actions and notifications described in the GDMO templates, can be included in different managed object classes, a value returning procedure is defined for each template (except for the behaviour one). Every procedure (named as *initialization procedure*) will store the static properties defined in the template in the different fields of a variable of the proper generic data type, and will return its value to the calling entity.

Among the static properties of a managed object are the ASN.1 data type of parameters, attributes, and actions and notifications associated data. The CMIP protocol uses the ASN.1 predefined type ANY to transport these values, but the value syntax of this type is not supported by the language defined in the Z.105 ITU-T recommendation. The solution suggested in this recommendation is to substitute the ANY type with the predefined type `CHOICE`. Thus, a `CHOICE` type will be defined to transport the parameter values, another one for the attribute values, and so on. Every `CHOICE` will have several fields, one for each element of the same kind in the information model.

The present algorithm simplifies the automatic generation of managed object SDL specifications because the implementation of the static properties of a managed object only involves the call to the procedures corresponding to the mandatory GDMO packages and the conditional ones whose associated condition is fulfilled, and the call of procedures corresponding to the superclasses of the object. The procedures corresponding to packages will also call the procedures resulting from the translation of the attributes, attribute groups, actions and notifications belonging to the packages. This method also allows the reuse of GDMO definitions: a GDMO template definition is translated into a SDL procedure, which will be called as many times as references from other templates.

Dynamic properties of managed objects, expressed in the behaviour templates using natural language, should be substituted by one or more SDL procedures (*behaviour procedures*). According to MIM, GDMO, and the draft of GDMO+, we can distinguish the following behaviours types:

- *Preconditions*: conditions that must be true just before the execution of a task (for instance, the emission of a notification). A precondition can be modeled by a boolean returning value SDL procedure which returns true if the precondition is fulfilled.
- *Invariants*: conditions that must be true for some time interval. For instance, during object life-time, or before and after an action execution. In the last case, the invariant can be modeled by a boolean returning value procedure that can be invoked before and after the task execution.

- *Postconditions*: conditions that must be true just after the execution of some task. The postcondition can be modeled by a SDL procedure that executes the statements needed to force the postcondition to be true.
- *Events*: that can cause the execution of some task (if the existing preconditions and invariants are true). The events may change the managed object behaviour and can be represented by boolean value returning SDL procedures. Every one return a true value if the event has occurred. These procedures should be invoked periodically to know if a management operation depending on this event must be executed.

  If the event is due to a change in the managed resource, it cannot be full specified in GDMO, because of the fact that GDMO specifies managed objects, not real resources. The specification of this kind of behaviour in the SDL model cannot be obtained directly from GDMO, and depends on the SDL model chosen to represent the real resources and their communications with the related managed objects.
- *Action behaviour*: is the sequence of simple operations that are executed when an action is invoked. It can be represented by a SDL procedure that executes the operations.
- *Attributes relationship*: it represents some rules affecting the values that an attribute may take depending on the values of another related attributes. This behaviour can be represented by a SDL procedure that executes the tasks needed to guarantee that the rules about attributes values relationship are fulfilled. They should be invoked after modification of the value of any of the attributes involved in the relationship, or when the managed object is not processing any management operation.
- *Managed object creation and deletion rules*: they are the preconditions, invariants and postconditions related to the managed object creation and deletion, and the events that trigger these operations (in addition to the corresponding CMIP operation requests). These rules depend on the name binding used to name the object instances.
- *Attribute matching rules*: they identify how the matching rules related to the filtering task are applied to an attribute. These rules can be modeled by boolean value returning SDL procedures. The procedure parameter is the value we are interested in to compare with the attribute value. It returns true if both values match according to the rule.

To simplify the translation algorithm, the name of procedures can reflect their functionality. For instance, the names might take the form `templateType_be-`

haviourType_templateName. templateType should be replaced with the type of the template, behaviourType with the type of behaviour procedure (createPrecondition, orderingFilter, etc.), and templateName with the name of the behaviour template.

An scheme of the translating algorithm is shown in Figure 1. The solid lines represent the result of the translation, and the dashed ones the auxiliary data used in the process.

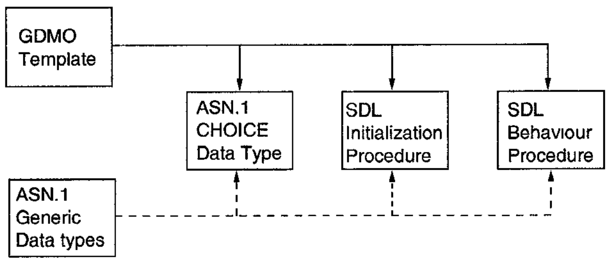

Figure 1: Translating algorithm scheme

The results of the translation of a information model (ASN.1 data types definitions, initialization and behaviour procedures) will be grouped in a SDL package which will be used in the definition of the SDL model of the managed system.

2.3 Managed object specification

With the translation algorithm presented in previous section, we obtain ASN.1 data types specifications and a set of initialization procedures that can be used to store in variables the static properties of a managed object, and a set of behaviour procedures to represent the object specific dynamic behaviour. In this section we present the SDL model of a managed object, which integrates those aspects with the behaviour common to all managed objects.

Every managed object is modeled as a SDL process type. This process inherits its generic static and dynamic properties from a generic managed object process type (genericManagedObject process type), which can receive management operations and emit notifications as defined in the MIM recommendation. It also includes all the managed object specific properties defined in the procedures obtained in the previous section. The communications between the managed object process and its agent are done through signal exchange. There will be a signal for each kind of management operation. The parameters of every signal will be a subset of those associated to the corresponding CMIP operation.

Managed object attributes, attributes groups, actions and notifications static properties are stored in array variables defined in the SDL process type. The data types of the arrays elements are the *generic data types* cited in section 2.2.

The multiple inheritance mechanism permitted in GDMO cannot be directly translated into SDL, because SDL only admits simple inheritance. This feature can be included in the SDL model by calls to the initialization procedures corresponding to the managed object super-classes. The static properties obtained from these procedures will be combined according to the rules expressed in MIM. This task will be done by the SDL procedure `addProperties`.

Figure 2 shows the definition of a SDL process type modeling a generic managed object. It shows the signals the process can receive corresponding to the different management operations requested by a manager, and sent via the local agent (signals from signal list `MOCMIPreq`). Receiving a signal determines the execution of a procedure that performs the proper tasks, including the sending of a signal to the agent with the operation response data (signal from the signal list `MOCMIPresp`). The procedures whose tasks depend on the managed object class being modeled, have been defined as **virtual**. This fact lets the procedure redefinition in the SDL process subtype modeling the specific class.

The initial transition consists in the execution of the init procedure, used to initialize the data structures of the process type. This procedure will be redefined in each specific process type to call the initializing procedure corresponding to the managed object class modeled by the process type (this procedure is obtained from the translation method explained in the previous section). After that, the process enters in the monitor state where it waits for management operation requests. These operations are performed by the shown procedures, which will execute the apropriated behaviour procedures defined in section 2.2.

2.4 Agent specification

An agent can be modeled by a SDL process which communicates with the manager processes using signal exchange. Each signal represents a kind of management operation in a similar way as the ones used in the interface between agent process and its managed objects processes. The parameters of every signal will be the ones associated to the corresponding CMIP operation.

The agent behaviour consists of a generic one, inherited from a generic agent process type (process type *genericAgent*), and a specific one that depends on the

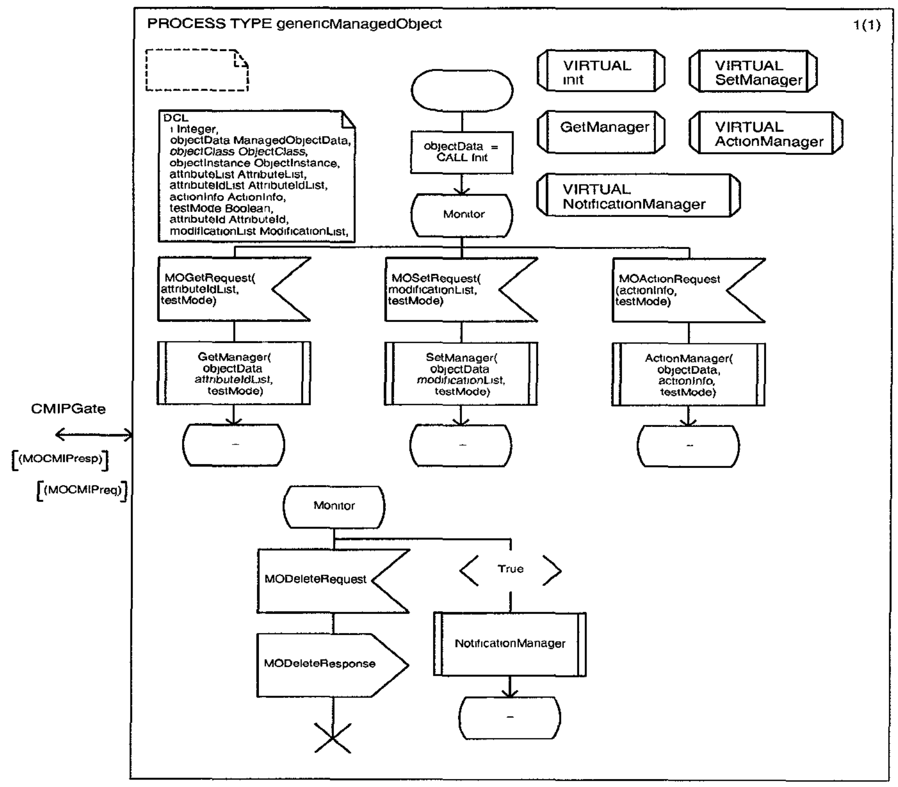
PROCESS TYPE genericManagedObject
1(1)
VIRTUAL init
VIRTUAL SetManager
GetManager
VIRTUAL ActionManager
VIRTUAL NotificationManager
DCL
i Integer,
objectData ManagedObjectData,
objectClass ObjectClass,
objectInstance ObjectInstance,
attributeList AttributeList,
attributeIdList AttributeIdList,
actionInfo ActionInfo,
testMode Boolean,
attributeId AttributeId,
modificationList ModificationList,
objectData = CALL Init
Monitor
MOGetRequest(attributeIdList, testMode)
MOSetRequest(modificationList, testMode)
MOActionRequest (actionInfo, testMode)
GetManager(objectData attributeIdList, testMode)
SetManager(objectData modificationList, testMode)
ActionManager(objectData, actionInfo, testMode)
-
-
-
CMIPGate
[(MOCMIPresp)]
[(MOCMIPreq)]
Monitor
MODeleteRequest
True
MODeleteResponse
NotificationManager
-

Figure 2: Generic managed object process type

objects properties of its Management Information Base (MIB).

The tasks that a SDL process acting as a OSI generic agent should be able to perform are the following:

- *Scoping*: if the requested operation includes a scoping field, the agent must select a set of object instances in the naming tree according to the scoping type and base object specified.
- *Filtering*: if the requested operation includes a filtering field, the agent must apply the filter requested to the objects previously selected by scoping (or to the base object if no scoping is specified). A filter is a logical union of attribute value assertions and/or tests of the presence of attributes in a managed object. To test

these assertions we can use the procedures `equality_attributeName`, `ordering_attributeName`, etc. defined in section 2.2 that model how a matching rule can be applied to a particular attribute. Only the objects whose attributes verify the filter will be selected to receive the management operation. In our model, scoping and filtering functions will be implemented by the `scopeAndFilter` procedure, which returns a list of process identifiers corresponding to the selected objects.

- *Synchronization*: a management operation affecting several objects can specify two types of synchronization: i) atomic, if the operation must be executed on all the objects or not, if some of them cannot execute it (for instance, if a precondition is not fulfilled); or ii) best-effort, if it is admissible to execute the operation only on some of the selected objects. If the operation includes the atomic synchronization, operation will be invoked on all the selected objects using the true value in the parameter `modeTest` to test if the operation is possible in all of them. If this is true, it will really invoke it on them (`modeTest = False`). Otherwise, an error to the manager requesting the operation will be sent. If the operation only needs best-effort synchronization, the agent can really invoke the operation on the objects (some of them can reply that the operation is not possible).
- *Processing of operations on several objects*: the agent should translate every operation into a set of operations applied to each of the objects selected by scoping and filtering.
- *Event discrimination*: notifications emitted by managed objects are sent to the local agent, but they do not include address information about the managers that should receive them. Thus, managers should register at the agent which handles the corresponding objects, the notifications they want to receive. With this information, the agent can forward the received notifications to the proper managers.
- *Communication with the SDL processes that model managed objects*: the agent has to translate the naming information of a management operation (object class and instance) into a SDL process identifier (PId).

There are some behavioural aspects expressed in GDMO templates that are not really dynamic properties of the managed object but agent properties. These properties, that also must be implemented by the agent process, are the following:

- *Constraints about managed object instance creation*: as the managed object is not created yet, the agent is responsible for checking if the create constraints are fulfilled. These constraints depend on the name binding used for instance naming.

As shown in section 2.2, these constraints can be expressed in the SDL procedures obtained from the translation of the behaviour templates referenced from the name binding ones (procedures named `nameBinding_CreatePreconditions_...`, `nameBinding_CreateInvariants_...`) These procedures can be used by the agent to check if the new object can be really created. After creation, agent must guarantee that object creation invariants and postconditions are fulfilled by executing the procedures `nameBinding_CreateInvariants_name` and `nameBinding_CreatePostconditions_name`. The agent also has to obtain the initial values for each attribute if the create behaviour includes an IVMO[3] as source of values, and to send them to the managed object process.

- *Delete constraints*: before deleting an object, the agent has to check if the delete constraints from the name binding are true. This can be done by the procedure `nameBinding_DeletePreconditions_name`. If preconditions are true, the object can be really deleted; otherwise, the agent sends an error to the manager requesting the delete operation.

Informally, the complete behaviour of the agent process could be as follows. The agent process can receive management operations requests (mapped to signals in our model) from several manager processes. The information about base object, scoping and filtering associated to the request is used by the agent to obtain the set of target managed object processes (identified by their PId). The agent requests the operation to each of these objects and wait for the responses. Response information received by the agent may be used to build the response information returned to the manager (if the operation is confirmed).

If the agent receives a notification issued by some of its managed objects, it will find the managers that should received it using the event discrimination function. With this information, notification will be forwarded to the proper managers.

The agent and generic managed object model are combined with the data type definitions and procedures result of the translation of the MIB to obtain the complete SDL model of the managed system.

Figure 3 shows the integration of the translation algorithm results with the generic managed object and agent models to obtain the specific managed system model. The solid line represents the result of translation of the GDMO MIB, while the dashed ones represent the data used to obtain a specific part of the system.

[3]Initial Value Managed Object.

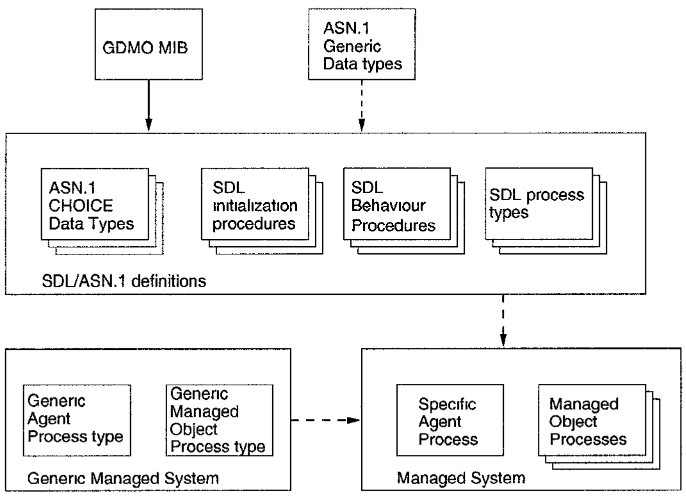

Figure 3: Overall translation procedure

Currently, we are developing a tool to automate the presented algorithm. The actual beta version of the tool is able to obtain the ASN.1 CHOICE data types, SDL initialization procedures and a skeleton of the managed object process types from a GDMO MIB description. To analyze and validate the SDL definitions obtained from the translation tool and to develop the generic managed object and agent SDL models, the SDT tool from Telelogic is been used.

3. Translating algorithm examples

An example of the translation algorithm proposed is presented in this section. The example chosen is the `vpServiceId` attribute, defined in [13].

3.1 Attribute `vpServiceId`

The definition of the `vpServiceId` attribute is as follows:

```
vpServiceId ATTRIBUTE
WITH ATTRIBUTE SYNTAX  ASN1XuserModule.VpServiceId;
MATCHES FOR EQUALITY;
BEHAVIOUR  vpServiceIdBeh;
REGISTERED AS  {xuserAttribute 29};
```

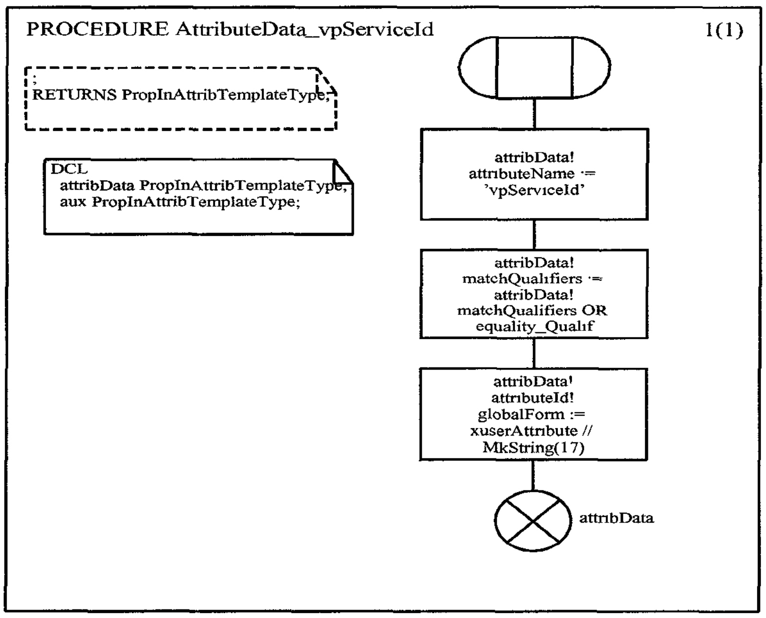

Figure 4: Attribute `vpServiceId` initialization procedure

The translation of the `vpServiceId` attribute will result in:

- The modification of the ASN.1 CHOICE type `AttributeValue` (used to transport attribute values in management operations) to reflect the presence of a new attribute in the information model. This data type definition will include a new field with the *vpServiceId* attribute syntax (extracted from the GDMO template construct `WITH ATTRIBUTE SYNTAX`):

```
AttributeValue ::= CHOICE {
-- vpServiceId attribute syntax --
attributeType_vpServiceId      Type_vpServiceId
}

Type_vpServiceId ::= ASN1XuserModule.VpServiceId
```

- The initialization procedure `AttributeData_vpServiceId`, shown in Fig. 4. This procedure will store the attribute template name, the filtering operation supported and the object identifier of the attribute in the proper fields of a variable of a generic ASN.1 type. The value of this variable will be returned as the result of the procedure. This procedure will be called from the procedures corresponding to the packages including this attribute.

- The behaviour procedure `equality_vpServiceId` that may be used by the agent process in a filtering operation involving this attribute (this information is extracted from the `MATCHES FOR` section of the GDMO template and from the attribute behaviour).

4. Conclusions

The mapping of management information models expressed in GDMO to SDL specifications has been investigated in this paper. The approach is based on the mapping of the whole set of GDMO templates to SDL procedures which describe the corresponding static and dynamic properties and ASN.1 data type definitions. These procedures will be used in the SDL process types that model the managed objects. This solution facilitates the reuse of templates definitions, and simplifies the automatic translation between both languages. It also solves some of the problems due to the differences between them (i.e. GDMO conditional packages and multiple inheritance, not directly supported in SDL).

Since SDL-92 is intended for the specification of whole systems, SDL might be used to specify the managed system (agent and managed objects) and the managing system. The presented method can also help in this task because some of the procedures and data types definitions obtained from the translation procedure could be used for the specification of the agent process and for the partial definition of the manager process (at least the interface with the agent). The formal definition of managed systems could be of interest for analysis and simulation purposes.

Future works will have to consider the problem of modeling object behaviour inherited from several superclasses and/or packages.

References

[1]. ITU-T Recommendation X.720: Management Information Model, January 1992.

[2]. ITU-T Recommendation X.722: Guidelines for the Definition of Managed Objects, 1992.

[3]. ITU-T Recommendation Z.100. SDL: Specification and Description Language, 1992.

[4]. ITU-T Recommendation X.208: Abstract Syntax Notation One (ASN.1), 1988.

[5]. ITU-T Recommendation Z.105: SDL combined with ASN.1 (SDL/ASN.1), March 1995.

[6]. J. Keller. An Extension of GDMO for Formalizing Managed Object Behaviour. In *FORTE'95*, 1995.

[7]. ITU-T Recommendation X.722, Draft Amendment 4: GDMO+ Specifying the Behaviour of Managed Objects, January 1997.

[8]. Shahrzade Mazaher and Birger Moller-Pedersen. On the use of SDL-92 for the Specification of Behaviour in OSI Network Management Objects. In O. Faergemand and A.Sarma, editors, *SDL'93: Using Objects*. SDL Forum, Elsevier Science Publishers B.V., 1993.

[9]. A. Bartocci, G. Larini, and C. Romellini. A first attempt to combine GDMO and SDL techniques. In O. Faergemand and A.Sarma, editors, *SDL'93: Using Objects*. SDL Forum, Elsevier Science Publishers B.V., 1993.

[10]. A. Bartocci and A. Ferrero. Integrated Use of SDL and GDMO. In R. Braek and A. Sarma, editors, *SDL'95 with MSC in CASE*. SDL Forum, Elsevier Science Publishers B.V., 1995.

[11]. A. Ferrero and A. Bartocci. SDL and GDMO Integration. Technical report, CSELT, September 1995.

[12]. Samir Tata, Laurent Andrey, and Olivier Festor. A practical experience on validating GDMO-based Information Models with SDL'88 and 92. In *SDL'97*. SDL Forum, September 1997.

[13]. EURESCOM. *Pan-European TMN - Experiments and Field Trial Support. Deliverable 7: Specifications of the Xuser Interface for ATM Network Management. Volume 4 of 5: Annex 3 - Xuser Interface Information Model*, November 1996.