52

MODE: a Development Environment for Managed Objects based on Formal Methods

Olivier Festor[1]
Centre de Recherche en Informatique de Nancy
Bâtiment LORIA, BP-239, 54506 Vandoeuvre-lès-Nancy, FRANCE, Tel (+33).83.59.20.48, Fax (+33), E-mail: festor@loria.fr

Abstract

The need for mechanisms and techniques to describe formally managed objects behaviour in the Open Systems Interconnection (OSI) Management Framework has been recognized in various places. Building a formal specification of managed objects forces the designer to be more rigourous and allows a better understanding of what has been done. But the development of such specifications is a difficult and time consuming task which must be supported by a powerful set of tools. Moreover the effort invested in the development of the formal specification should pay off in some way during the Management Information Base development process.

In this paper, we present a development environment based on the formal mechanisms we include with the Guidelines for the Definition of Managed Objects (GDMO) notation to allow Managed Object behaviour to be formally described. This environment is intended to improve the process of building a formal description of OSI based Management Information Bases and provide several tools to exploit this formal description during the whole development process.

Keywords:

Behaviour, Development Environment, Formal Description Techniques, GDMO, Network Management.

[1]The authors work is also supported by IBM European Networking Center, Heidelberg, Germany

A. S. Sethi et al. (eds.), *Integrated Network Management IV*

1 INTRODUCTION

One of the most important and complex task of OSI based management application builders is the design and modelling of the network components they want to manage. To facilitate the specification of such network components, the GDMO notation has been standardised within ISO (ISO-10165.4 1992) and is today widely accepted and used as the description technique for Managed Object (MO) design and specification.

The need to formally describe MO behaviour and provide guidelines for using the various specification templates of GDMO in a more systematic fashion, leading to clearly structured, coherent models has been expressed in (Kilov 1992). A first attempt for a design method based on a detailed study of behaviour classification has been proposed in (Clemm & Festor 1993).

The effort invested in formalizing Managed Object behaviour can be used to derive a better product and to automate certain steps of the development process. Accordingly, this formalization effort can only be accepted and used if it is part of a well defined development process and supported by an integrated development environment which provides tools for exploiting these new functionalities in the development process.

In this paper we present the MODE (Managed Object Development Environment) development environment. This set of tools is based on the development process used in Formal Description Techniques (FDT) based approaches and supports both standard GDMO and the formal extensions we proposed to the behaviour part of the notation.

The remainder of this paper is organised as follows: the next section describes the purpose and main goals of the development environment. Section 3 provides some features of the formal mechanisms we have adopted to extend the GDMO notation. Section 4 contains the description of the Management Information Base (MIB) design tool. Section 5 presents the MIB design application. Section 6 is concerned with a validation tool which allow formally described MOs to be interactively simulated. Section 6 provides information on the status of the environment and some future directions are discussed. Finally, a summary of the presented work is given.

2 PURPOSE OF THE ENVIRONMENT

In the last years, several tools and software environments based on the standardised GDMO notation have been proposed ((Dossogne & Dupont 1993, Wittig & Pfeiler 1993)) and today several products are available on the market. Most of them have nice features, several advantages and probably some limitations. However, none supports formally described behaviour for MOs and MIBs and thus tool support for this aspect of the development process is missing.

When the decision was taken to start the development of the MODE integrated tool-set, our goal was not to produce "yet another development environment" but to implement tools in order to validate our concepts on how behaviour should be formally described and how this formal part could improve the whole development process of MIBs. We concentrated our work on trying to provide both validation and test generation tools

in an early stage of the development process which have not been considered, due to the lack of formalism in the standard, in most other toolkits. Thus we can say that these tools can be considered as extensions to other development environments rather than concurrent ones.

3 THE FORMAL BACKGROUND

The MODE development environment currently supports the extensions we have proposed to GDMO in a language called LOBSTERS (Festor 1994). Most concepts we developed for the integration of LOBSTERS into GDMO can be easily applied to other formalisms which are or about to be standardised in the OSI framework. After a summary of the LOBSTERS concepts, the link to other FDTs is discussed. Then a short overview of the selected development process is presented. The definition of this process is done to identify which support tools are expected in our environment.

3.1 LOBSTERS

LOBSTERS is the acronym of "Language for Object Behaviour Specification based on Templates and Extended Rule Systems".The notation is a compatible extension of the standard GDMO notation. In LOBSTERS, the static parts of objects (attributes, operation and action signatures, packages, ...) are exactly the same templates are those defined in GDMO. The formal behaviour part in LOBSTERS is based on an extended version of the Communicating Rule Systems FDT (Mackert & Neumeier-Mackert 1987). The notation based on a set of rules for describing behaviour has been extended with object-oriented features such as inheritance (Festor & Zoerntlein 1993). As the CRS Formal Description Technique supports the standardised ASN.1 notation and provides several operators to access and manipulate ASN.1 typed variables (Schneider 1992), the link with the static part of GDMO was trivial. A first approach to the integration of the rule mechanisms into the behaviour templates of the GDMO notation has been proposed and the result of this integration is the LOBSTERS FDT. It is fully compatible with the standard GDMO notation, i.e. it can be parsed with standard parser as well as extended ones, and provides facilities to specify formally the MO behaviour.

One of the main problems encountered during the integration was how behaviour specifications should be distributed over the various templates of a Managed Object, i.e. packages, conditional packages, attributes, actions, other MOs,inheritance, etc... This problem was resolved by defining a methodology for the development of behaviour specification and more generally by defining a rigourous approach to the specification of Managed Objects. This approach is based on specialization concepts and scope limitation in each kind of behaviour template present in a MO definition (Clemm & Festor 1993). Based on this approach, an algorithm for collecting different behaviour parts within a Managed Object definition was designed. This algorithm takes all distributed behaviour parts and builds one rule set by connecting the different rules through basic predicate logic operators such as AND/OR. Through the use of this algorithm it is possible to

determine the behaviour specification for any given MO and thus open some issues to validate, test and verify extended GDMO specifications which is not possible with the standard notation alone.

In addition to formal behaviour specification, LOBSTERS also provides a simple mechanism to specify formally the presence requirements of conditional packages. Based on basic first order predicates these formalized conditions are very helpful in increasing the automation in the development tools. Especially all generation tools can through these expressions detect automatically which conditions are to be met to generate the code associated with a given package and generate, for each MO, the validity mechanism which will check on a create-request if the given package-requirements conform to the standard specification of the MO.

3.2 Dealing with other formalisms

As mentioned above, the behaviour in LOBSTERS is described using rules where the condition and effects parts are specified with first order logic predicates. This mechanism is similar to the pre/post conditions approach of VDM (Jones 1990) or Z (Spivey 1989). Thus, most concepts developed for the integration of the rule-based approach into the GDMO notation can be also mapped onto the integration of another formalism based on predicates. Especially the distribution method and its associated collection algorithm can be applied to these Formal Methods as well.

The impact of a new formalism on the environment concerns at this time only small parts of the code. The development tools could easily integrate a formal behaviour description based on another FDT if the mechanisms are integrated into GDMO in a way that is compatible with the notation. This is important to consider at a time where new FDTs are in the standardization phase within ISO.

3.3 The formal development process

When an environment for the development of a particular type of system is designed it is always based on a well defined development process. In our approach we have based the MODE environment on the process depicted in Figure 1. This process is the one used in most Formal Description Techniques based approaches.

The first stage of the MIB design consists of the creation of a high level specification of the desired MIB (*formalization* stage). To achieve this task, the MIB designer uses both the requirements defined for its task and a set of existing MO definitions in order to reuse some of them in his model. Thus, the first tools that are required to help the MIB designer in the process are tools which facilitate the selection of MO classes from a set which can be very large and specification support tools which allow new specification to be built and integrated into the available set. These tools are mainly parsers and graphical user interfaces to access in a user friendly way all information necessary to the performance of this first task resulting in a first model of the MIB.

When a first formal model of the desired MIB is designed, the work of the MIB designer consists in a systematic refinement of this specification (*specialization*). This

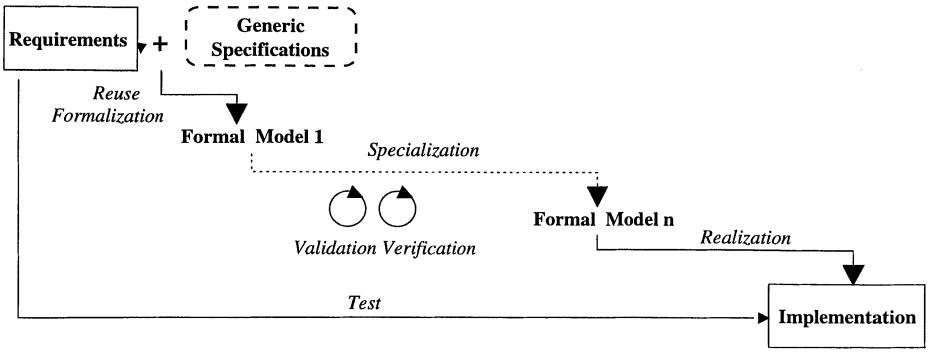

Figure 1: The FDT-based development process.

refinement can be iterated several times until the specification is precise enough to be implemented. Tool support for these stages concerns mainly validation and verification tools which guarantee that all constraints are met. These tools can be either provers or specification simulators.

When the specification is precise enough to be implemented, so-called *realization* tools are used. These tools are in most cases, code generators and compilers. Finally, when the test of the implementation towards the requirements has to be performed (*test* phase), both test execution tools and, in previous stages of the design, test generation methods and tools are required.

All these tools facilitate the work of the MIB designer, ease the whole development process and thus, justify the use of formal methods for the MIB specification.

4 The MODE environment

The MODE environment consists of two main tool-sets. The first one, called the front-end part, allows MO designers to create, parse and load managed object specifications. As the behavioural extensions proposed in LOBSTERS are fully compatible with the standard notation, the specifications which have to be parsed by the front-end can be either standard or contain LOBSTERS conditional predicates and formal behaviour parts. These tools are used intensively in the first phases of the development and are also helpful in all refinement steps where syntax parsing is necessary. Based on this front-end part, several other applications can be built and integrated in the environment. These are either validation or code generation tools.

4.1 Architecture and front-end

As depicted in figure 2, the front-end part of the environment contains, three basic parsing tools which are a GDMO parser, a second pass behaviour parser which extracts formal

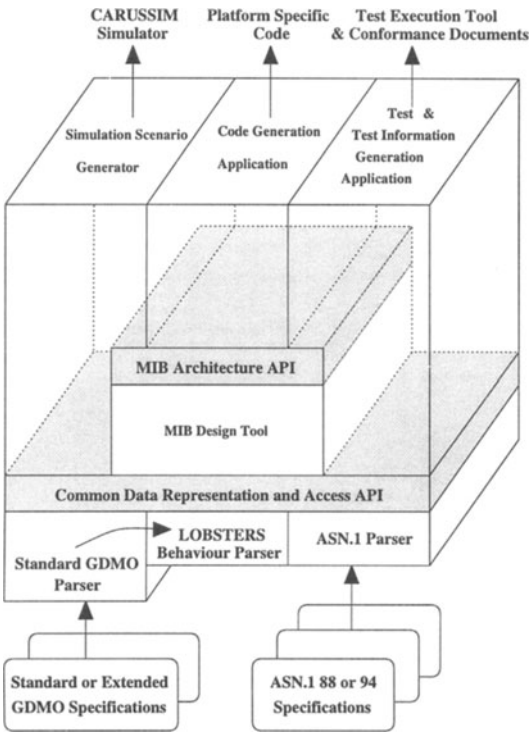

Figure 2: Architecture of the MODE environment.

expressions from the LOBSTERS specifications, and an ASN.1 parsing tool. The ASN.1 compiler is an extension of the SNACC compiler developed at the University of British Columbia (Sample 1993). It was extended with a back-end coding ASN.1 specifications in the common intermediate representation allowing these specifications to be exploited by the simulator. Some work has also been done in supporting new features in the notation according to the new ASN.1 draft. The integration of the behaviour parser was facilitated by the encapsulation of formal specifications into the basic behaviour description template of the standardized GDMO notation. This could be resolved in a more elegant way by adding specific formal behaviour templates into the GDMO notation.

These tools are used to parse and load specifications. All the information extracted from these parsing steps are stored in an internal C++ representation and accessible by tools through a well defined Application Programming Interface (API).

Based on this API several tools have been designed and implemented. These applications help the MIB designer in specifying, editing and validating his model. These tools are a MIB design application which is presented in the next section, a simulation scenario generator and a stepwise simulation tool. Several other applications can be built

over the MIB design application, e.g. code generators or test generation tools. As we have focussed our attention on the early stages of the development, the code generation was not considered yet. However some work is going on in our group on this area.

4.2 The MIB design tool

The MIB design tool is the first application built over the Common representation API. This tool provides several facilities to MIB designers for template edition, modification, syntax checking through the previously mentioned parsers, and interactive MIB building. This application provides facilities to support both reuse and formalization steps. This application serves as a front-end for all other application tools which are the scenario generator, code generators and a test design tool.

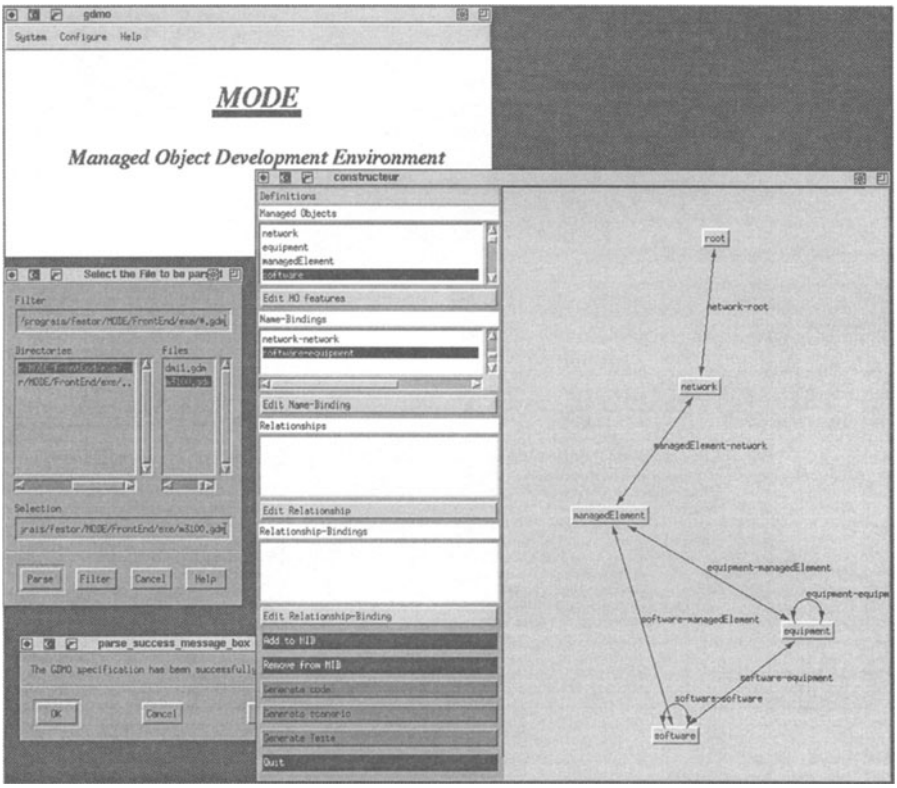

Figure 3: Interface of the MIB design tool

Figure 3 contains a screen shot of the main window from the MIB design tool. Following features are supported by the application:

- edit: several definitions can be edited in a user friendly way. These definitions can be MO classes, name-binding definitions, relationships or relationship bindings. Internal features such as attributes, actions, can also be edited but not at this level.

- add a managed object class to the MIB: if there is only one possible name-binding and if the container object is still in the expected MIB schema, the object is added to the MIB (e.g. the `managedElement` MO can only be inserted into the MIB if the `network` MO is present). If several Name-Bindings are candidate the user selects which ones are supported and all selected ones are added to the MIB. Note that the static semantics check of the definitions is performed at this level whereas the syntax check is performed at the parser level. Thus, an MO can only be added if all definitions (packages, attributes,..) are fully defined. This allows the working MIB to be always consistent.

- add an additional name-binding: if both container object classes and contained ones still exist within the MIB additional name-binding can be added to the MIB (e.g. the `equipmentequipment` name-binding can be added after the `equipment` MO was inserted).

- remove objects and/or name-bindings from the MIB: several MOs or name binding can be removed from the MIB architecture. When a MO which contains several other ones is removed, then all MO which are associated through a Name-binding to the one removed are also removed from the MIB as wee as the concerned Name-Bindings. This is done for MIB consistency requirements. For example, if the `managedElement` is removed from the MIB depicted in figure 3, then both `software` and `equipment` MOs as well as the related name-bindings (`software-managedElement`, `equipment -managedElement`, `software-software` and `equipment-equipment`) are removed too.

The MIB design application also provides facilities to define in an interactive way all parts of new MOs as well as their formal behaviour.

Concerning the relationships between MOs, the tool supports the relationship model defined in (Clemm 1993). Here also relationships can be added/removed from the MIB. However in the area of relationships, the integration of additional support such as code generation and test generation are not yet supported.

At each step during the design process, the containment tree of the current MIB architecture is visible in a user friendly way and all objects, name-bindings and relationships can be accessed.

The application has been implemented in C++. The graphical user interface was developed using OSF/MOTIF. The current MIB architecture is accessible through an API. This API is used by the application tools to collect the information they need to perform their task. We will now present one of the application tools which exploits both the presented MIB architecture and the formalized behaviour part from LOBSTERS. This application is the simulation environment.

5 THE SIMULATION TOOL

The first application we realized was a tool to simulate a MIB based on its behaviour specification. This application can be used for validation and verification of the specification of designed MIBs, to test whether they really exhibit the desired behaviour. This application can be used in several steps of the development process and is directly based on the formal description of the behaviour.

5.1 The purpose of simulation

Simulation is a way for MIB designers to analyse a model of their system in an early stage of the development. It allows the focus on different abstraction levels from basic interactions with the environment to a detailed study of internal states and interaction parameter values.

Applied to MIB scenarios, this can be very helpful to validate single Managed Objects instances (testing the interfaces and states of an object isolated from its environment), validate MIB hierarchies scenarios (testing a whole or a part of a MIB in order to check consistency of the model) and validate the management interface (adequacy of the access service to the MIB model). These validations steps have been defined and the simulator has been used for this purpose in the design of MIBs for Virtual Private Networks (see (Preuss 1993, Schneider, Preuss & Nielsen 1993)).

5.2 The CARUSSIM tool

The CARUSSIM (Communicating Automated Rule Systems SIMulator) was originally developed to be used in the area of protocol validation (Eschebach 1991). In order to adapt the simulator to the new features necessary for MOs, several enhancements have been made to the kernel (Frot, Lecorguillé, Lefranc & Orain 1993) and to the data (ASN.1) representation and manipulation part (Orain 1993). Especially full support of ASN.1 constructs such as sets, choices, sub-typing, object identifiers and value notations frequently used in MO specifications have been implemented in the tool.

In Figure 4, the graphical user interface of the simulator is depicted. It shows a simulation of a specialized Managed Element derived from the M.3100 TMN managed Object catalog. All attributes of the MO called ComputerMO are visible (operationalState, administrativeState, ...). The MO is accessible through three interfaces which are the management interface on which operations can be invoked on the MO, the notification interface through which the MO issues notifications, and an internal interface through which the MO is influenced from the real resource it models (here modelled as an environment rule system).

Through the interactive simulation, all operations such as activation, locking, ... can be performed on the MO and the MIB designer can check if the observed behaviour is the one expected. Naturally the tool can be used to simulate a more complex MIB which contains several MOs built around a containment tree, elaborated with the MIB design tool.

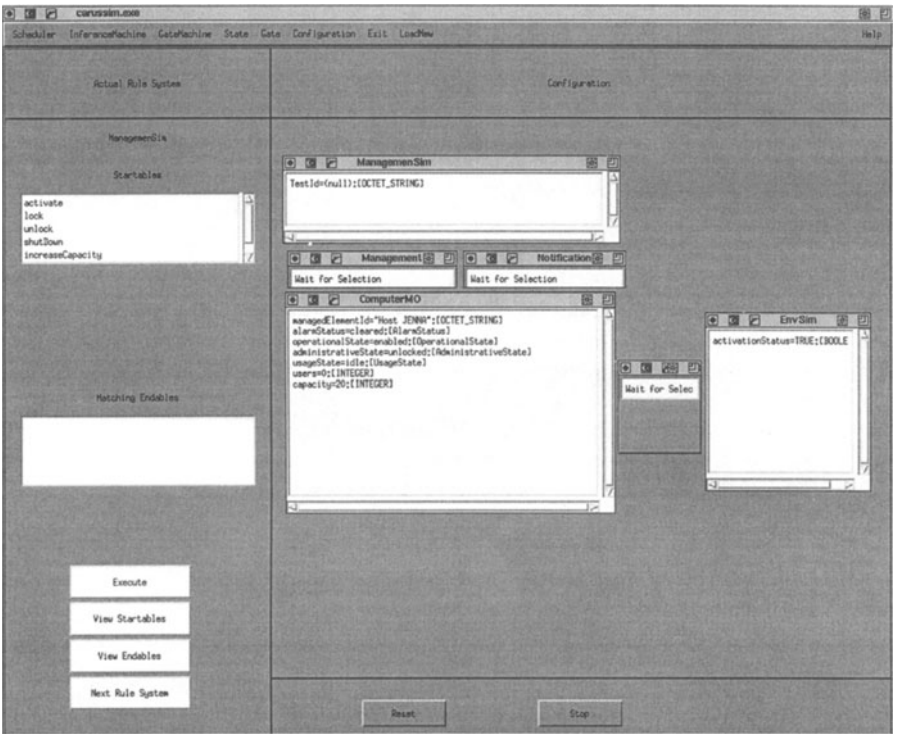

Figure 4: The simulator graphical user interface.

6 Status and future work

The main features of the MODE environment have by now been implemented. The available features are the GDMO parser, the LOBSTERS behaviour parser, the MIB design tool, the simulation scenario generator, and the scenario interactive simulator. Provision of the test generation and code generation tools is planned for mid 95. Some work has also started on a more extensive support for relationships and their influence on code and test generation.

As LOBSTERS is not a standardised notation for describing formally behaviour, we are now starting on applying the methods used for the integration of the rule based approach into GDMO, on other formal methods. In this area we are planning to integrate some features of either VDM or Z into GDMO and add these new methods to the environment. It seems that the more powerful tools for these formal methods are the ones developped around VDM. In order to provide a complete development environment based on formal methods like we started with MODE, some investigations are going on in the area of mapping the LOBSTERS concepts onto the use of VDM-SL in GDMO.

7 CONCLUSION

In this paper, we have presented a development environment for Managed Objects which is based on GDMO and the extensions we proposed in LOBSTERS for the formal specification of the behaviour part.

We have shown that the formal behaviour description with LOBSTERS can be exploited during various stages in the development process: for Management Information Base design, validation, code generation not only for static but also behaviour parts. As a result the development environment is to that respect more powefull than approaches that ignore the very important aspect of behaviour.

The use of formal methods in the development of OSI-based MIBs is heavily dependent on the availability of development tools which provide additional facilities to support and exploit the formal development process. The MODE environment is a first step toward this goal. However a lot of additional work has still to be done to apply the concepts of LOBSTERS and MODE to formal methods that are currently subject to standardization. This task is yet going on in our group.

8 ACKNOWLEDGMENTS

The author wishes to thank Alexander Clemm for his careful reading of this paper; Wilko Eschebach who developed the CARUSSIM simulator for his precious help during the extension; David Orain who spend many months on improving the whole environment and especially the ASN.1 part of the tool; Dr. Juergen Schneider and Thomas Preuss for having encouraged my work and for their feedback with respect to their application of the formalism and for testing the tools in their project; Dr. Georg Zoerntlein for its advices during the design of LOBSTERS.

9 REFERENCES

Clemm, A. (1993), "Incorporating Relationships into OSI Management Information". 2nd IEEE Network Management and Control Workshop, Tarrytown, NY, 21-23 September 1993.

Clemm, A. & Festor, O. (1993), "Behavior, Documentation and Konwledge: An Approach for the Treatment of OSI-Behavior", *in* 'Fourth International Workshop on Distributed Systems: Operations and Management'. October 5-6, 1993, Long Branch, New-Jersey, USA.

Dossogne, F. & Dupont, M. (1993), "A software architecture for Management Information Model definition, implementation and validation", *in* H. Hegering & Y. Yemini, eds, 'Integrated Network Management, III (C-12)', Elsiever Science Publishers B.V. (North-Holland), pp. 593–604. Proc. of the IFIP TC6/WG6.6 3rd. Int. Symp. on Integrated Network Management, San francisco, Ca., 18-23 April, 1993.

Eschebach, W. (1991), "Interpretative Ausfuehrung kommunizierender Regelsysteme". Master Thesis, University of Kaiserslautern, Germany, 1991.

Festor, O. (1994), "OSI Managed Objects Development with LOBSTERS". Fifth International Workshop on Distributed Systems: Operations and Management, 12-16 Septembre 1994, Toulouse, France.

Festor, O. & Zoerntlein, G. (1993), "Formal Description of Managed Object Behavior - A Rule Based Approach", *in* H. HEGERING & Y. YEMINI, eds, 'Integrated Network Management, III (C-12)', Elsiever Science Publishers B.V. (North-Holland), pp. 45–58. Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management, San francisco, California, USA, 18-23 April, 1993.

Frot, J., Lecorguillé, H., Lefranc, J. & Orain, D. (1993), "CRUSADE: un environnement de développement de protocoles". Industrial Project Report, Ecole Supérieure d'Informatique et Applications de Lorraine, 1993.

ISO-10165.4 (1992), '"Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects"'.

Jones, C. (1990), *"Systematic Software Development Using VDM (second edition)"*, Prentice Hall.

Kilov, H. (1992), "Understand→Specify→Reuse: Precise Specification of Behaviour and Relationships.". IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, Munich, October 1992.

Mackert, L. & Neumeier-Mackert, I. (1987), "Communicating Rule Systems", pp. 77–88. Proc. 7th. Int. Symp. on Protocol Specification, Testing and Verification, H. Rudin, C.H. West (editors), North-Holland 1987.

Orain, D. (1993), "A New ASN.1 Compiler for the CRUSADE Environment". Master Thesis, Ecole Supérieure d'Informatique et Applications de Lorraine, 1993.

Preuss, T. (1993), "Management von virtuellen privaten Netzen". Master Thesis, University of Magdeburg,1993.

Sample, M. (1993), *"SNACC 1.1: A High Performance ASN.1 to C/C++ Compiler"*. Master Thesis, University of British Columbia.

Schneider, J. (1992), *"Protocol Engineering: A Rule-based Approach"*, Vieweg.

Schneider, J., Preuss, T. & Nielsen, P. (1993), "Management of Virtual Private Networks for Integrated Broadband Communication". Proc. ACM SIGCOMM '93, San Francisco, CA, september 1993.

Spivey, J. (1989), *"The Z Notation"*, Prentice Hall.

Wittig, M. & Pfeiler, M. (1993), "A Tool Supporting the Management Information Modeling Process", *in* H. Hegering & Y. Yemini, eds, 'Integrated Network Management, III (C-12)', Elsiever Science Publishers B.V. (North-Holland), pp. 739–750. Proc. IFIP TC6/WG6.6 3rd. Int. Symp. on Integrated Network Management, San Francisco, Ca., 18-23 April, 1993.

10 Biography

Olivier Festor received the Master Thesis degree in Computer Science from the University of Nancy I, Nancy, France, in 1990 and the Ph.D. degree in Computer Science from the University of Nancy I, Nancy, France in 1994.

During his Ph.D., he spent three years at the IBM European Networking Center in Heidelberg, Germany, researching application of formal methods in the area of OSI-based Network Management. He is now working as a researcher at the Centre de Recherche en Informatique de Nancy, Nancy France. His current interests are in the fields of Network Management, Formal Description Techniques, MIB specification notations and development environments.