Applying UML to Design an Inter-Domain Service Management Application

Mohamed Mancona Kandé[1], Shahrzade Mazaher[2], Ognjen Prnjat[3], Lionel Sacks[3], and Marcus Wittig[1]

[1]GMD-Fokus, Kaiserin-Augusta-Allee 31,
D-10589 Berlin, Germany
{kande, Wittig}@fokus.gmd.de
[2]Norwegian Computing Center, P.O.Box 114 Blindern,
0314 Oslo, Norway
shahrzade.mazaher@nr.no
[3]University College of London, Dept. of Electrical Engineering, Gower Str.,
London WC1E6BT, UK
{oprnjat, lsacks}@eleceng.ucl.ac.uk

Abstract. We present a component-oriented approach to demonstrate the use of the Unified Modeling Language (UML) and Open Distributed Processing (ODP) concepts to design service components of a telecommunications management system. This paper is based on the work done in the design phase of the ACTS project "TRUMPET" (Inter-Domain Management with Integrity). This project undertook to produce a service management architecture suitable for emerging liberalised telecommunications markets. The criteria where that the system should be highly distributed both technologically and administratively -- i.e., across many kinds of organisations. TRUMPET project presents a good model environment to develop not only the service architecture itself but also the methodologies for producing such designs. In our approach, we use the conceptual framework of ODP and discuss some methodological issues related to ODP-viewpoint modelling of distributed systems using UML notations. We conclude with recognising the power of combining UML and ODP so as to manage the complexity of the problem.

1 Introduction

Designing and implementing complicated distributed control systems in large international groups or consortia is non-trivial. Techniques are required to ensure that all the participants understand where their work applies, the work of their collaborators, and the relationship between these. Further, when building systems which are expected to have some longevity it is important that people looking at the documentation can understand what is involved. These considerations motivate the use of architectural design methodologies and semi-formal methods in system modelling. Another important consideration is that the methodologies used should facilitate close analysis of the design to ensure, at early stages, that the system is complete in meeting its requirements and consistent in its operation. This should yield robust,

well-engineered products. Finally, when presenting the product to users, its functionality and roles within their business practices must be clearly documented and defined (following the motivations behind the Network Management Forum OMNIPoint Ensembles [9]).

This paper illustrates how two popular techniques can be combined to achieve these goals, and how this was done in the context of the TRUMPET service management architecture [3]. The TRUMPET project undertook to produce a service management architecture suitable for emerging liberalised telecommunications markets. The criteria were that the system should be highly distributed, both technologically and administratively -- i.e., across many kinds of organisations. Specifically, these organisations include: *Public Network Operators* (PNOs), *Customer Premises Network* centres (CPNs) and third party *Value Added Service Providers* (VASPs) or bandwidth retailers. Moreover, the TRUMPET consortium consisted of a number of engineers, from several companies across Europe who had to collaborate to design, build and run the system. Thus the TRUMPET project presented a good model environment to develop not only the service architecture itself, but also the approach for producing such an architecture.

The two techniques used were the ITU-T Open Distributed Processing (ODP) [10] architecture and the Universal Modeling Language (UML) [11]. ODP is a framework that supports the building of distributed systems. At the top level, ODP requires that the system at hand be modelled from a number of different *viewpoints* defined within the ODP Reference Model (RM-ODP)[10]. UML is a semi-formal modelling language which provides a diagrammatic notation and its semantics using mature object-oriented concepts. By using the appropriate UML diagrams for each of the ODP viewpoints, it is possible to design a system with clear traceability between the different viewpoints. UML and ODP, both being well documented and widely accepted within industry, can be understood by engineers from many countries and companies. Using the two together therefore makes it possible to present the designs in an -- almost -- universal language both to collaborating designers and to customers.

The rest of this paper is arranged as follows. In Section 2, a brief overview of the TRUMPET scenario is given. Section 3 describes the ODP viewpoints and discusses the rationale behind using UML for describing ODP viewpoints. In Section 4, a Virtual Private Network service is specified in each of the ODP viewpoints by using the appropriate UML diagram types. Section 5 concludes the paper by assessing the used approach.

2 The TRUMPET Scenario

The ACTS project TRUMPET (Inter-Domain Management with Integrity) focuses on the secure operation of inter-domain management systems within the Open Network Provisioning framework. The TRUMPET scenario in Fig. 1 involves the following players: two (or more) Public Network Operators (PNOs), a Value Added Service Provider (VASP), and a number of customers at various sites -- Customer Premises Networks (CPNs) [2], [15]. The customers see an end-to-end connection and are not necessarily aware of which PNOs are contributing to establish the connection. The VASP sees the connection as a set of *segments*, each supported by a different PNO, but does not know how each segment has been set up within the corresponding PNO (i.e., what Asynchronous Transfer Mode-based switches are used).

The management systems of the players mentioned above form a service provisioning system for management and provision of broadband (Asynchronous Transfer Mode-based) network connections between two customers/end users. CPN is a dedicated service in the customer organisation, which already has a contract with the VASP. The VASP management system provides network connectivity to customers by utilising the resources of one or more Public Network Operators. The VASP allows customers to create, modify and delete connections, thus effectively providing the Virtual Private Network (VPN) service to the customers. PNOs provide the physical infrastructure, i.e., the network, and the adequate management interface to interact with it.

Fig. 1. TRUMPET Scenario

The TRUMPET management system thus represents a typical component-based application for distributed service management. Entering the design phase, a need for suitable methodology and supporting notation for modelling such a distributed system emerged. Having critically reflected on existing methodologies and notation schemes, the consortium decided to adapt the Unified Modeling Language concepts to the ODP framework and design the system in such a fashion.

3 Design Approach: UML and ODP

This section is concerned with the methodology and the notation chosen for the analysis and design of the TRUMPET management system. It presents the methodology and discusses the rationale for the choice of the notation to be used.

A Methodology represents a set of rules for structuring the task of developing a system, by grouping the analysis and design information and defining steps to be followed during system refinement. Notation refers to the way, either textual or diagrammatic, of representing the information, and of describing the structure and functionality of the system and its components.

The methodology adopted in TRUMPET is that of the Open Distributed Processing Reference Model (RM-ODP) [10]. ODP provides a general architectural framework to which distributed systems, operating in an open, heterogeneous environment, must conform. The basis of this architectural framework is the development methodology encompassing five distinct viewpoints. These viewpoints allow different participants in system development to observe the system from different perspectives and levels of abstraction:

- The Enterprise viewpoint is directed at the needs of the users of a system. The system is modelled in terms of the required functionality, the domains involved, the actors and their roles.
- The Information viewpoint describes a consistent, common view of information resources that support the information requirements of the Enterprise viewpoint. It also defines relationships between information elements and the information processing.
- The Computational viewpoint is concerned with the functional decomposition of a system, i.e., the computational objects, the interfaces they offer (the functionality they support), and the interactions between them.
- The Engineering viewpoint describes the infrastructure needed to support distribution, i.e., the provisioning of transparencies. The Engineering viewpoint defines the communication needs and the deployment of functionality.
- The Technology viewpoint is concerned with the details of the components and platforms from which the system is constructed.

The viewpoints are partial views of the complete system specification, and the description of the same component can exist in different viewpoints. This gives rise to a viewpoint consistency issue, i.e., the consistency of specifications across different viewpoints. For each viewpoint, ODP has defined a language, meaning a set of concepts and rules to be used when specifying the system in the corresponding viewpoint. However, it has left open the issue of which notation to use for these viewpoint languages.

There are some major concerns when choosing notations for the ODP viewpoints. First, since ODP is based on the object paradigm, the notation for each viewpoint should support that paradigm as well. Second, the notation should be able to express the concepts defined in that viewpoint. Third, it should be possible to check the consistency of the different viewpoint specifications of a system. It would be an added advantage if one could trace an entity that occurs in more than one viewpoint. Another consideration is that the notation should be intuitive and easy to use and understand. In other words, the notation should ease the communication among development team members and others outside the team (such as customers). This latter consideration becomes even more important in a distributed, international environment such as that of TRUMPET.

Different semi-formal and formal languages are used for specifying different ODP viewpoints. Formal descriptions are employed in the ODP framework to enable a precise and unambiguous definition and interpretation of ODP standards. But usually, different languages are used for different viewpoints, making it difficult to check consistency between different viewpoint specifications and to trace system components across viewpoints. Applying a single language/notation to all of the ODP viewpoints will solve these problems, but it requires that the language/notation have a rich set of core concepts to cover all the viewpoints.

There are several (semi-) formal languages/notations that can be considered for the purpose of specifying ODP viewpoints. Among those, the notation that best fulfils the requirements

above is the Unified Modeling language (UML). Among the other languages, the Specification and Description Language (SDL) [12] is also object-oriented and has a graphical notation, but it lacks the richness of concepts available in UML. Moreover, SDL does not have any extension mechanism, such as UML's *stereotype*, to compensate for the missing concepts. The lack of extension mechanism also applies to the other formal languages considered. The formal languages based on mathematical notations, e.g., LOTOS [8] and Z [16], are not only difficult to understand and communicate, but have a limited set of basic concepts.

The approach proposed in [7] for mapping the different ODP viewpoints to UML diagram types has been developed and adopted in the TRUMPET project. This approach is illustrated in Fig. 2. The major benefit of such a mapping consists in supporting both an object-oriented modelling and design process, and the design of reusable components and distributed services [6] in the sense of distributed systems as described in the *Reference Model for Open Distributed Processing* (RM-ODP) [10]. A combination of ODP and UML helps also to map and to implement some ODP functions in different technologies like *Common Object Request Broker Architecture* (CORBA) and *Telecommunications Information Networking Architecture -- Distributed Processing Environment* [13].

Fig. 2 depicts the links between ODP viewpoints [1], and shows the relationships between these viewpoints and the UML diagrams, as well as between UML diagrams themselves (which are mapped to the same viewpoint specifying different aspects of same objects). The design of a distributed management application starts with capturing requirements of the system in terms of *Use Case Diagrams*. The results obtained from a use case model may be used to present high level *Static Structure Diagrams/Class Diagrams* as indicated in the *Enterprise viewpoint*, and these diagrams can be specified in more detail in the *Information viewpoint*. The step from *Class Diagrams* to *Statechart Diagrams* allows the specification of the dynamic behaviour of significant information objects.

Fig. 2. Relationships between UML Diagrams and ODP Viewpoints

The *Computational viewpoint* maps to the *UML Collaboration*, *Sequence*, *Activity* and *Component Diagram* types. While the UML *Collaboration Diagram* shows the interactions among instances and their links to each other, the *Sequence Diagram* describes object interactions arranged in a time sequence. The *Activity Diagram* allows the specification of the order in which activities (such as operations provided by a computational object interface) have to be executed. The *Component Diagram* shows the organisations and dependencies among components.

In addition to the *Component Diagram*, the *Deployment Diagram* is also mapped to the *Engineering viewpoint*. This latter diagram type shows how components and objects are distributed and moved around the system [4]. There is no mapping between UML and the *Technology viewpoint* offered by this approach.

4 Case Study

This section describes how, within the ODP framework, different UML concepts and notations were used to design the TRUMPET management system. A separate section is dedicated to each of the five ODP viewpoints by projecting the Virtual Private Network service in the corresponding viewpoint using the appropriate UML notation schemes.

Each viewpoint first gives an overall view of the TRUMPET management system, and then focuses on a more detailed description of the VPN service within the Value Added Service Provider domain.

4.1 Enterprise Viewpoint

The ODP Enterprise viewpoint represents an overview of the system's aims, constraints and functionality as seen by the enterprise. This viewpoint models the basic system decomposition into components, identifies actors, policies and domains, and describes the general scenarios of the system's use.

The TRUMPET system incorporates three domains (or enterprise objects in ODP terminology) namely the Customer Premises Network, CPN, the Value Added Service Provider, VASP, and the Public Network Operator, PNO. The PNO domain is further subdivided into PNO Service Layer and PNO Network Layer. These four entities were modelled as UML Packages with interdependencies, using UML Class Diagram notation as shown on Fig. 3.

Fig. 3. The TRUMPET Class Diagram Enterprise Package

Next, the desired functionality of the system is described. In the ODP context, this is done by specifying the scenarios, or Use Cases, that describe how actors/entities interact in the context of using the system.

Fig. 4. The Use-Case Diagram

The TRUMPET scenarios were specified using the UML Use Case diagram, depicting the actors, sets of Use Cases (ellipses) within a system, and associations between actors and Use Cases -- illustrated in Fig. 4. Note that the UML stereotype *<<community>>* has been used to classify the high-level enterprise object *"TrumpetManagementSystem"* as community in the sense of ODP.

Fig. 4 depicts the functionality (or Management Functions) identified in the Value Added Service Provider management service as Use Cases, and the interaction of the different users with the Use Case package. As shown, there are six Use Cases. Customers/end users are capable of reserving end-to-end connections (of a given duration and desired Quality of Service), modifying them (changing duration, Quality of Service, or both), and releasing, i.e., deleting these connections. Public Network Operators are capable of notifying the users via the Value Added Service Provider of connection activation, or notifying the users of the connection release due to a segment/link failure.

4.2 Information Viewpoint

In the ODP Information viewpoint, the information objects of the system are identified and their structures and relationships described. UML Class Diagrams were used to describe the static structure of the information objects.

Fig. 5. The Class Diagram of the VPN Service

Fig. 5 illustrates, by means of a UML Class Diagram, the overall structure of the Virtual Private Network service. That is, what entities are involved, what their relationships are, and how they interact.

In the following, the emphasis is put on the Value Added Service Provider (VASP) entity of the above diagram, and a more detailed description of this entity is given. In the Class Diagram of Fig. 6, the VASP is decomposed into its three main components: the *CustomerServer* which handles the communication with the customer domain (CPN), the *ControlServer* which, after negotiations with the involved Public Network Operators, establishes, modifies, or releases the Virtual Private Network connections, and the *MIB* (Management Information Base) which contains all the Managed Objects. These objects contain information about the different entities that the VASP either interacts with or manages, e.g., objects containing information about the VASP customers or objects representing the connections that the VASP manages. Both the *CustomerServer* and the *ControlServer* have access to the MIB for either retrieving information from it or updating it.

Fig. 6. The Overall Class Diagram of the VASP

Among the different components of the VASP, the Management Information Base represents the pure informational objects. The other two entities can be regarded as information processing units (manipulating the MIB) although each of them contains information about the current state of the VASP.

The customer MIB contains information pertaining to the VASP customers, their respective service profiles, and the terms of their subscriptions. A corresponding MIB exists for the Public Network Operators whom the VASP is dealing with. These MIBs are rather static, in the sense that the information they contain is seldom updated. The Connection MIB contains information about all the connections that the VASP is currently supporting. Furthermore, its structure reflects the view that the VASP has of a connection, i.e., a connection consisting of segments individually supported by a Public Network Operator. This MIB is being constantly updated (therefore dynamic) as requests for new Virtual Private Connections and change/release of the existing ones are received from the customers.

4.3 Computational Viewpoint

The Computational viewpoint describes how the management functions, identified via Enterprise Use Cases, are performed by the management system. Each management function is described in terms of computational objects and computational activities, the latter representing sequences of operations invoked on computational objects.

As a starting point for the computational design, the components identified in the Enterprise viewpoint can be mapped to computational objects which provide an abstract, course grain computational view of the management system. Each component can then be broken further down into a set of computational objects representing the detailed computational object model. At this level, the UML Class Diagrams have been used to describe the structure of computational objects, their interfaces, and the relationships between them. At the abstract level, UML Component Diagrams have been used to describe the system components and their external interfaces.

Fig. 7 depicts the design of the *VaspVpnManager* component (referred to in the Information Viewpoint as the *Control Server).* We distinguish the *VaspVpnManagerFacade* package containing the external structure (client view of the component) from the *VaspVpnManagerImpl* package that contains the implementation details about the internal class structure.

Fig. 7. Internal Structure of the *VASP-VPN-Manager* Computational Object Type

The *VaspVpnManagerImpl* package realises the interfaces contained in the "façade" package. It does so by implementing objects that directly support the management functions offered by the interfaces, and objects that support the former objects in carrying out their task.

Fig. 8. The *VASP-VPN-Manager* Computational Object Type Diagram with Interfaces

Note that the interface type diagrams of Fig. 8 and Fig. 9 provide the information necessary to easily produce an OMG-IDL (interface definition language) file. This would be the first step in mapping the computational design to a concrete CORBA-compliant implementation platform. The UML diagrams thus support directly and ease the implementation task for those platforms.

Fig. 9. Computational Object Interface Type for *VASP-VPN-Manager*

Next, the computational activities are described. Computational activities are the interactions between the computational objects in order to perform the management functions defined through Use Cases in the Enterprise viewpoint. Interaction between computational

objects is described in terms of an operation invocation initiated by a client object requesting an operation to be performed by a server object. Precedence rules are used to define the sequence of operations performed when an interaction takes place. To describe the computational objects interactions UML Collaboration diagrams and Sequence diagrams have been used. These two diagram types convey mostly the same information. Depending on how important the time dimension or the lifetime of computational objects are, one could use one or both of the diagram types.

Fig. 10. The *Reserve Connection* Sequence Diagram

The UML Sequence and Collaboration Diagrams as depicted in FIg. 10 and Fig. 11 describe the *Reserve Connection* Use Case defined in the Enterprise viewpoint. These diagrams illustrate the interactions, sequences of messages, and relationships among computational components (such as PnoConnectionManager, VaspVpnManager) as well as programming level objects (e.g., instances of UML objects within the *VaspVpnMamager* package) [7].

Fig. 11. The *Reserve Connection* Collaboration Diagram

4.4 Engineering Viewpoint

This viewpoint focuses on the actual realisation of interactions between distributed objects and on the resources needed to accomplish this interaction. It comprises concepts, rules and structures for the specification of the system viewed from the engineering perspective.

This viewpoint introduces three main concepts, namely, *nodes*, *clusters* and *capsules*. ODP nodes match quite closely UML nodes, therefore, as depicted in Fig. 12, the concept of *UML-nodes* was applied to design *PNO-Host*, *VASP-Host*, and *CPN-Host*.

ODP capsules are a grouping of engineering objects forming a single unit for the purpose of encapsulation, processing and storage (e.g., VpnManagerServer and PNOConnMngrServer in Fig. 12). They can be thought of as runtime modules. The UML Component Diagram is therefore the appropriate candidate to specify the set of runtime modules and their interactions. The overall view of the system, i.e., how the capsules are distributed across the nodes, is then most naturally conveyed by the UML Deployment Diagram.

An ODP cluster refers to a group of objects that are always together and can migrate only as a whole from one capsule to another (whether on the same node or not). The tight coupling that exists among the cluster objects can be best conveyed by the UML Composition concept of Class Diagrams. In Fig. 12, the Capsule Customer-Server contains a cluster, a composite object tagged with the stereotype <<*cluster*>>, that can migrate from the VASP-Host to the CPN-Host as indicated by the <<*becomes*>> stereotype.

Fig. 12. Deployment Diagram

4.5 Technology Viewpoint

The ODP Technology viewpoint describes the choice of implementation technologies used to bring the design accomplished through the four previous viewpoints to life. This viewpoint describes the configuration of the hardware and software on which the distributed system relies. Although there are no dedicated UML diagrams to describe this viewpoint, the object-oriented concepts of encapsulation and abstraction provided by UML in previous design steps allow the system to be implemented in an heterogeneous environment in terms of computer architectures, programming languages and operating systems. This is one of the main advantages of the object-oriented approach adopted by UML. The Technology viewpoint was therefore described using plain English.

The Customer Premises Network is realised as a group of Java objects providing an interface to the Value Added Service Provider's Virtual Private Network functionality. A HTML based user interface is also provided as the end-user interface to CPN Java objects. The CPN-VASP communications are implemented in Voyager, a Java-based communications mechanism. The VASP is fully implemented in Java, apart from the Management Information Base which is based on the Lightweight Directory Access Protocol (LDAP) which effectively supports functionality required by the Telecommunications Management Network [5] for (TMN)-like Management Information Base. Public Network Operator's management system is a TMN-OSI platform, the HP-OpenView, which communicates via Common Management Information Protocol (CMIP). Thus, VASP implementation requires a JAVA-to-CMIP gateway, which is realised as a platform-independent CORBA gateway. These technologies were chosen so as to fulfil the trial requirements and project aims.

5 Conclusion: Experiences and Lessons Learned

This paper discussed the rationale behind the choice of UML as a notation scheme for describing ODP viewpoints and its application in the context of the ODP framework for the design of elecommunications management services. A case study describing the development of the TRUMPET inter-domain service management system illustrated the approach. This approach proved to have some advantages that validated the choice of UML as well as some drawbacks.

There were two main advantages in using the ODP-UML combination. First, there was no need to shift paradigm when trying to represent ODP core concepts in UML, since both are base on object-oriented principles. This eased the task of mapping by allowing to focus solely on finding the most suitable UML representation for the entities to be specified -- different paradigms would entail mapping (the building blocks of) the two paradigms to each other first.

Second, both UML and ODP look at a system from different perspectives. UML's versatile diagram types made it possible to use only one notation to specify all of ODP's five viewpoints, illustrated by the example used throughout the paper (Use Cases were used in the Enterprise, Class Dagrams in the Information, Collaboration Diagrams and interface types in the Computational, and Component Diagrams and Deployment Diagrams in the Engineering viewpoints).

Furthermore, the use of a single notation for all ODP viewpoints resulted in a shorter start-up phase for the project. It also had the benefit that consistency checks between the different viewpoint specifications could be done more easily, as needed, by the project members. Conversely, the ODP framework proved to be an efficient way of structuring different UML notations and thus managing the potential high complexity of large models.

Although the UML notation has many attractive aspects for use in the design of a distributed system, it also has some drawbacks. Many of the ODP core concepts are not directly supported by UML. For such reasons, UML introduces the concept of stereotypes to provide for extensibility. In the example of the previous sections, stereotypes have been extensively used to map ODP concepts that did not have a direct counterpart in the UML notation, such as enterprise objects, communities, clusters, etc. Moreover, although the concept of an interface is part of UML, its description uses the same notation as for a class. Again a stereotype, <<*interface*>>, has been used to differentiate between the class and the interface descriptions. The existence of an extension mechanism compensates for the lack of suitable basic concepts, but the extensive use of the same notation to express quite different concepts becomes at best confusing. One of the main benefits of a pictorial notation is to be able to get a general understanding of a given diagram without having to rely on the annotation text. This benefit is lost when the same representation is used for many different core entities.

UML did not prove to have enough power to fully describe the ODP concept of the computational object. During the design, only the external interfaces provided by a component were specified, and concepts like binding rules and lifetime aspects were not included.

In conclusion, the use of the ODP-UML methodology in TRUMPET proved efficient in supporting collaborative work in a large, geographically distributed consortium. After the initial methodology was decided upon, the work assignment was agreed on and understood within an afternoon of discussions. After the labour division was made, the consortium undertook to design the system according to the approach defined.

UML being quite widespread in both industry and academia, most people involved in the project had had some kind of experience with it before. Those less fluent in UML picked it up rather easily partly because of its graphical notation and intuitiveness and partly because of their knowledge of other similar notations. The introduction of UML in the project was therefore quite smooth and established a common base of communication among the members of the development team.

The design was developed within the contractual deadline (three months). There were 15 individuals involved in producing the design document. A high quality design was achieved despite the size of the TRUMPET system and the limited development resources available. The documents produced in this phase, were also extensively used by the trials team, which helped run the system in the operational environment.

The shortcomings of the approach were mainly due to the lack of direct support in UML of ODP's core concepts. That would be greatly alleviated if UML would offer a graphical extension mechanism.

Acknowledgements

This paper is based on the original work developed by the ACTS project TRUMPET. The authors wish to thank all the partners of the TRUMPET consortium who contributed to this work. Ognjen Prnjat wishes to acknowledge the financial support provided by the British Council Overseas Research Scholarship. More information on the TRUMPET project can be found at http://ascom.eurecom.fr/ASRL/TRUMPET/Trumpet_public/.

References

1. Berquist, K., Berquist. A. (Eds.): Managing Information Highways. The PRISM book: Principles, Methods and Case Studies for Designing Telecommunications Management Systems. Lecture Notes in Computer Science, Vol. 1164, Springer –Verlag, Berlin Heidelberg New York (1996).
2. ACTS Project AC112 TRUMPET: *NIL-Security Prototype Report*, Deliverable 6 (1997).
3. ACTS Project AC112 TRUMPET: *Detailed Component and Scenario Designs*, Deliverable 8 (1997).
4. Fowler, M., Scott, K.: *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley (1995).
5. Hall, J. (Ed.): Management of Telecommunication Systems and Services: Modeling and Implementing TMN-based Multi-domain Management. Springer-Verlag, Berlin New York Tokyo (1996).
6. Jacobson, I., Griss, M., Jonsson, P.: Software Reuse: Architecture, Process and Organization for Business Success. Addison-Wesley (1997).
7. Kandé, M. M., Tai, S., Wittig, M.: On the Use of UML for ODP-Viewpoint modeling. In OOPSLA 97 Workshop on Object Oriented Technology for Service, System and Network Management. Atlanta, Georgia (1997).
8. ISO 8807: LOTOS: A formal description technique based on the temporal ordering of the observational behaviour. (1989).
9. The Network Management Forum: The OMNIPoint Strategic Framework. A Service-Based Approach to the Management of Network and systems. NJ (1993).
10. ISO/IEC 10746-1/2/3: Reference Model for Open Distributed Processing -Part1:Overview/Part2: Foundations/Part3: Architecture. (1995).
11. Rational Software, Microsoft, Hewlett-Packard, Oracle, Texas Instruments, MCI Systemhouse, Unisys, ICON Computing, IntelliCorp: The Unified Modeling Language, Joint Submission, OMG TC doc ad/97-01-01 - ad/97-01-14 .
12. CCITT: Recommendation Z100 Specification and Description Language (SDL). (1992).

13. Graubmann, P., Mercouroff, N.: Engineering Modeling Concepts (DPE Architecture). In TINA Baseline document TB_NS0005_2.0_0.94 (1994).
14. Sacks, L., et. al.: TRUMPET Service Management Architecture. In Proceedings of the 2nd International Enterprise Distributed Object Computing Workshop (1998).
15. ISO JTC1/SC22/WG19: Z Notation (draft version 1.4). (1998).