A Network Management Platform adaptable to CIM Model evolution

Nathalie Rico, Omar Cherkaoui and Elmi Hassan
Univerisité de Montréal, University of Quebec in Montreal
rico@info.uqam.ca, cherkaoui.omar@uqam.ca

Abstract

Network Management application developers are facing the challenge of the constant evolution of the information model used by the applications and the escalation of the functionalities the applications need to fulfill. The main problem is to introduce the new functionalities or new services on the platform in an efficient, cost-effective, and flexible manner. The difficulty is to minimize the impact on the already developed core software and applications. This paper proposes a framework allowing the evolution of the network management platform to adapt to the changes occuring during the Platform life cycle. The proposed platform framework is based on the CIM information model and its PCIM extension. The framework allows to extend the platform capability and enables an automatic validation while elimating the need to write adaptation code. We present the platform framework in the case of the policy manager. We illustrate how the platform can easily evolve to incorporate the new classes corresponding to the new services using the example of the Virtual Private Network (VPN) policy management.

1. Introduction

Networks are evolving at a dramatic paste, both in size and in complexity and need management solutions that can adapt to those transformations and cope with the increased complexity. One problem that the industry is facing is the constant and parallel changes occurring in the information models used to build those network management applications and store persistent data. Such changes in the information model usually mean that new services and data need to be addressed. The application that used the information model needs to be enhanced to support these new services and data. This paper proposes a solution to this platform evolution problem. We developed a Framework architecture that allows the evolution of the management applications to changes in the information model or to services. We present the platform framework in the case of the policy manager.

This work was triggered by our own experience in developing a policy management platform at UQAM. The platform was subject to multiple iterations. Changes were required to comply with the new policy information model standard versions. New applications based on the policy platform were developed: a policy application for a virtualclassroom [1] and a policy management for Optical Virtual Private Networks (VPNs) [2] [3]. For each new application introduced on the policy platform, developers had to rewrite a large portion of the platform code to adapt the platform to the new model needed by the application. The developer's difficulty in adapting the platform led to the proposed framework that recreates the platform core.

Various approaches have been proposed to solve the problem of platform evolution or new service introduction. Historically, the first approach proposed used APIs. This approach quickly led to scalability issues. The limitations of the API approach resulted in the use of middleware (such as a message bus or CORBA) [4][5]. However those middleware approaches force the applications to use adapters for interfacing with the message bus. Writing adapters can be time consuming and adapters are subjet to changes when new services are introduced. Another serious problem with this approach is that the process flow is embedded in the application or adapter logic. Other approaches are being proposed using a component-based architecture [6][7]. Those approaches allow plug-and play operation, dynamic discovery and dynamic invocation. The dynamicity is achieved by using an external process engine that defines the process flow. The process definition is exported into some standard format and loaded into the runtime environment of the process engine. This approach also requires a bus of middleware to exchange the information. The developer needs to know the bus API to add the new required services. Also, as more services are introduced, APIs need to evolve to allow information sharing.

This article presents a different approach that facilitates the model changes or the introduction of new services occurring in different time-scale of the Framework life cycle. The proposed Framework architecture allows the network management platform to expand in order to support new requirements with minimal code changes. The framework also allows to extend the platform capability while eliminating the need to write adaptation code. The

Framework data core is composed of generic classes that are adaptable to any data and are capable of linking themselves together. An XML (eXtended Markup Language) Schema hierarchy defines both the data structure and their possible associations. The information model description in the XML file can then be changed without hardly any modification to the Framework data core. The framework enables an automatic validation with a validation tool linked to the XML schemas.

The second section of this article presents the platform framework. The platform framework is illustrated for the case of the policy manager. The third section details the platform core. The policy model is based on the Policy Common Information Model (PCIM). The fourth section presents the mapping between the information model to XML and XML to the LDAP repository. We illustrate how the platform can recreate the new classes corresponding to new service addition using the example of Virtual Private Network (VPN) policy services. Finally, we draw concluding remarks and present an outlook of future work.

2. Platform framework

The platform life cycle is presented and a conceptual view of the platform is shown.

2.1. Platform life cycle

In analyzing the life cycle of a network management platform, we came to the realization that the platform undergoes various types of changes required to support new network elements, new services or to evolve the fundamental static portion of the platform.

Figure 1 illustrates the framework changes occurring in different time-scales of the platform life cycle. The first time-scale relates to the most static portion of the information model and is subject to major changes every three or four years. It implies substantial code updates and the release of a new Framework version. The second time-scale ranges from four to six months and corresponds to the introduction of new applications using the platform Framework. For example, in the case of the policy platform, it corresponds to the introduction of VPN MPLS and VLAN Services [1][2][3]. In that case, the updates to the information model do not affect directly the Framework. The last and shortest time-scale represents the release on the market of new equipment, a frequent event that can happen every month or two. Finally, configuration or topology changes can impact the platform.

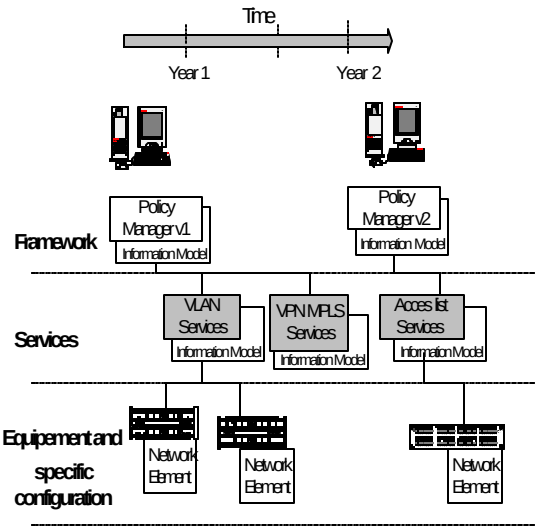

Figure 1. Platform Life Cycle

2.2 Conceptual View of the Platform

Figure 2 illustrates the conceptual view of network management platform consisting of a presentation layer, a business layer and a persistent layer.

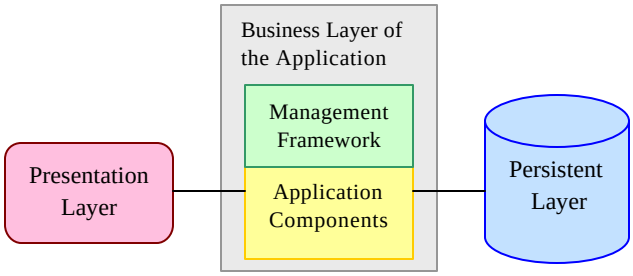

Figure 2. Conceptual View of a Network Management Platform

The Business Layer is composed of two levels, the Manager Framework and the Application Components that are "plugged" to the Framework. These Components are made of two types of classes, classes that extend some of the Framework classes and newly created ones that belong to the Application domain. The Framework is composed of core classes that can be extended by the application in order to use the management services it provides.

In the case of the policy manager, the policy manager framework is based on CIM and its policy extension PCIM.

The LDAP repository is used to store the management information.

3. Case of Policy Manager: platform framework

This section presents the case of the policy management platform and details the platform framework.

3.1 Platform Framework overview

In the case of the policy manager, the model used is the CIM and its extension PCIM.

Illustrated below in figure 3 is the proposed Framework architecture. The architecture is composed of four levels of classes in order to support the different time-scale change requirements. The first level is the Generic or meta classes (e.g. CIM_Class and CIM_Association). The second level corresponds to the Static classes modeling the Framework behavior. Generic and Framework classes model the most static portion of the information model that changes less frequently. Impacted on a more frequent basis, the Application level classes are specific to the application (e.g. Vendor Policy Condition) or model network equipments. Finally the fourth level corresponds to the Virtual classes that change on the most frequent basic.

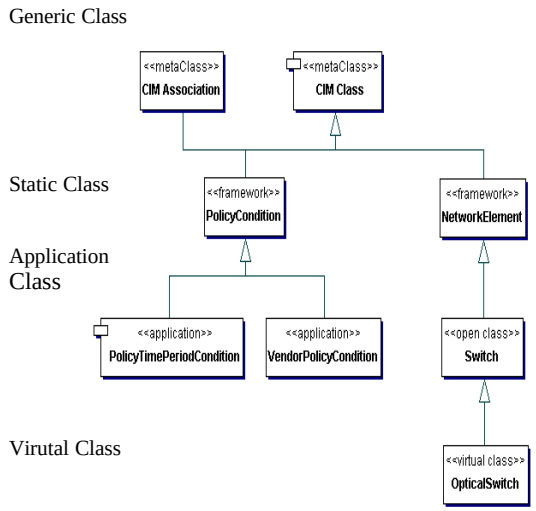

Figure 3. Overview of the Framework Architecture

3.2 Framework Classes

The Framework itself is built on two kinds of classes; Generic and Static (see Figure 4). The **Generic classes** use CIM Meta Schema elements and are implemented as defined in the CIM Specification. The Generic classes in the Policy Manager Framework are hence composed of the following four basic classes: CIM_Class, CIM_Association, CIM_Property and CIM_Reference.

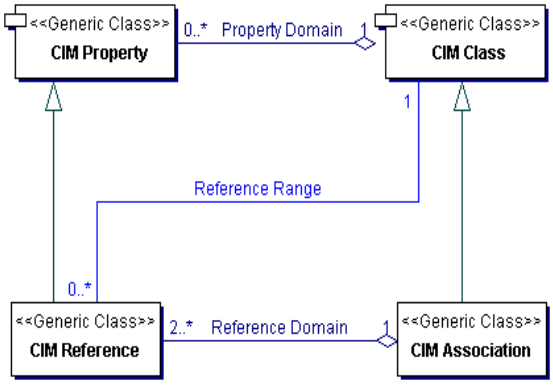

Figure 4. CIM Meta Schema Classes and their relations

The CIM and PCIM schemas define the **Static classes** in the Framework.. The **Static classes** of the Framework are specialization of the Generic classes and they incorporate the business behavior. The services offered to application developer give them the opportunity to extend CIM and to combine new services to the ones provided by the Framework. The extended classes are called **Application Classes** in the Franework. The validating rules of the Static classes are defined in the Framework and by the application developer for the new or extended classes. The validation process is described in section 4.

The last level of customization proposed enables the end-user to create new classes of object, **Virtual classes.** Virtual classes extend the classes offered by the applications. Virtual classes can be used in the case of new network element releases that need to be supported. For example, a new type of Switch can be created and stored on the LDAP server with its own set of properties on top of the already defined Switch's properties. The schema is modified and the new object type will then be recognized.

4. Communication and validation tool

The Framework uses XML to describe the information model and to validate the structure. The XML information is then mapped in the LDAP structure.

4.1 Model mapping

XML Schema can provide metadata through two documents: a schema document specifies the properties

(metadata) for a class of resources (objects). Each instance document provides specific values for the properties. DMTF has proposed a mapping from CIM to XML using the "Meta Schema Mapping" instead if the "Schema Mapping" [11]. In a meta-schema mapping, Meta Elements such as Class and Property will describe the CIM Schema instead of describing each element separately. That decision was based on the limited capacities of the DTD technology to describe the semantics of XML Documents. Since then, the W3C has approved the XML Schema Standard that allows more elaborate semantic description.

In our proposed Framework, we use XML Schemas and builds a Schema Mapping of CIM instead of a Meta Schema Mapping in order to provide a more powerful validation tool to the Framework. If the Meta Schema Mapping is used, only the form and not the content of XML Documents can be validated. In the following example showing an XML Document written following the Meta Schema Mapping, it is not possible to constrain an instance of "Class" to have an attribute name "PolicyRuleName". If the PolicyRule instance does not contain an element named PolicyRuleName, no error will be raised directly by the XML Schema since the Meta Schema Mapping can only determine that a Class instance must have a "name" attribute.

```
< Class name="PolicyRule">
  < Property
name="PolicyRuleName">Policy12</Property>
</Class>
```

Instead when the Schema Mapping is used, the XML Document is as follows:

```
< PolicyRule>
   < PolicyRuleName > Policy12 </ PolicyRuleName >
<PolicyRule>
```

In this case, the XML Schema clearly states that an element named PolicyRule must contain a child element named PolicyRuleName. If the PolicyRule does not contain a PolicyRuleName element, the XML Schema will raise an error.

The proposed policy manager framework uses the LDAP for storing the policy information [12][13]. The XML schema is mapped to the LDAP schema as per the standard proposals [14].

4.2 XML and Validation

The framework architecture is based on Generic self-described classes. The mandatory properties and the possible relationships between Classes are validated through the use of XML Schemas. Using such method of validation allows a one-step verification of instances of all elements described in the XML Schema. It also allows changing the information model and the inherent validation by updating the XML Schema.

A hierarchy of XML Schemas describes the information model used by the Framework and by the applications. These XML Schemas each represent a level of the Framework architecture. At the top of the hierarchy is the CIM XML Schema that describes the most static part of the Framework, in this case the policies. All the data, coming either from the GUI, the Core Components or the LDAP server then only needs to be transformed into an XML Document that uses the proper XML Schema in order to be validated.

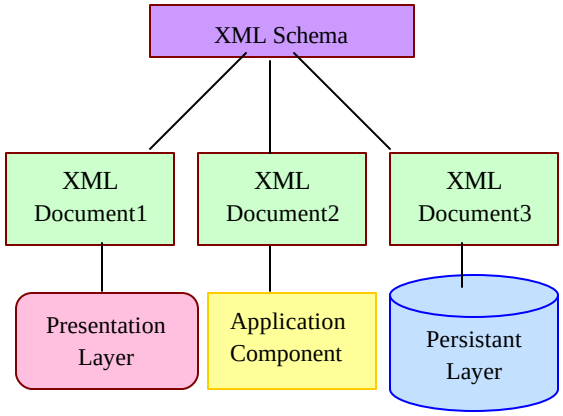

Figure 5. Validation with an XML Schema

Figure 5 represents a single level architecture using one XML Schema to validate data coming from three different layers in the application. The architecture has multiple levels: multiple classes of objects and their XML Schemas form a hierarchy as shown in figure 6.

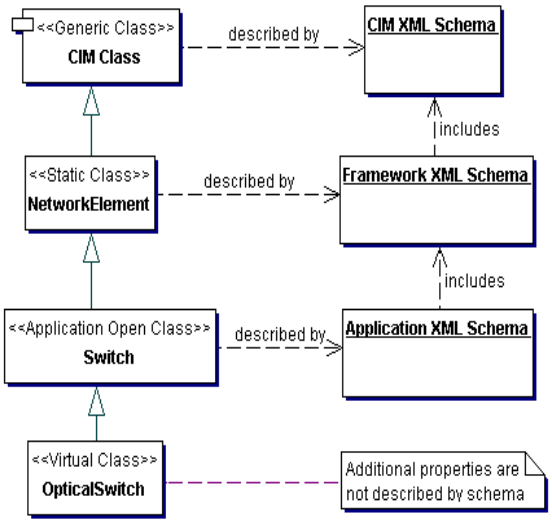

Figure 6. Class Hierarchy and associated XML Schemas

The hierarchy of XML Schema allows an easy validation. Furthermore if any change occur in the application the only thing that needs to be changed for the validation process is the XML Schema.

Moreover, since the schema is self-describing, the Classes of the Policy Manager Framework and the ones created for the Application can be used easily. Since the XML Schema of the Framework is built-in, the developer of new applications will have to extend the lowest XML Schema in the hierarchy to add new integrity constraints. These constraints will then be applied to the newly created classes or to the Framework extensions, making the validation process accessible to the Framework user.

Also as showed in figure 7, the properties of the virtual classes are not defined in any XML Schemas and their validation them becomes the sole responsibility of the LDAP Server. The Application Components will validate only the properties and relationships defined in the superclass. The new objects and their new properties transit through the Application Components without being validated. The validation occurs only when there is an attempt to store the information on the LDAP server: an exception can then be raised and returned.

5. Policy Platform Implementation

We have built the policy management platform using the framework presented and exploiting XML to perform validation. The policy platform is composed of the following elements: a policy user interface, a SQL database, a LDAP repository and three APIs. The first API allows creating new classes in the CIM-based information model. The second API allows reading and editing the information model instances. The third API allows mapping XML to LDAP. Figure 7 shows the Policy Platform components.

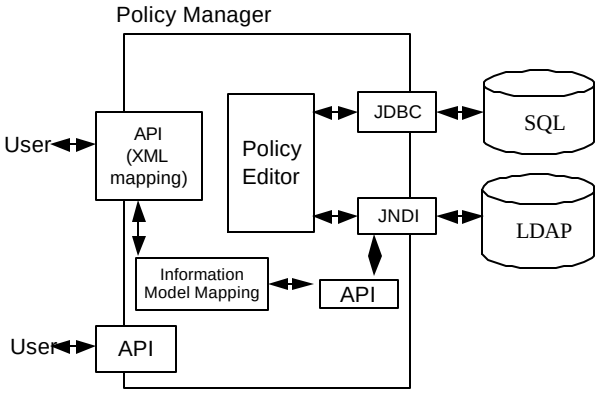

Figure 7. Policy Platform components

The Policy Platform allows creating new classes . The Policy Manager Class editor is show in Figure 8.

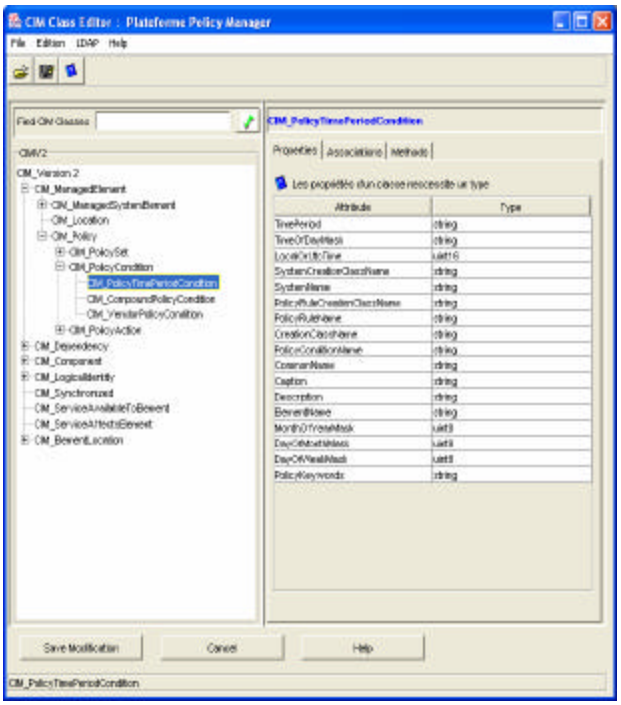

Figure 8. Policy Manager Class Editor

6. Experimentation: adding new policy services

We will present the experimentation using the MultiProtocol Label Switching (MPLS) virtual private network (VPN) policy manager. A VPN uses shared

facilities from a public network to provide the appearance and benefits of a private one, including continuous availability and reliability. VPNs offer organizations the network infrastructure needed to provide efficient communication channels for employees and corporate partners. For the customers, a VPN can extend the IP capacities of their enterprise to remote offices' and/or users' intranet and dialup services [15].

Policies provide rules of how network devices deliver services to users or sites at a given time. In the CIM model, policies are represented by a class that associates network devices, device profiles, sites, users and services, and a scheduler, all of which being also represented by classes. These are LogicalElement, OrganizationalEntity, AdminDomain and NetworkService. In order to manage VPN services, we need to extend these classes. Figure 9 presents the overall VPN model.

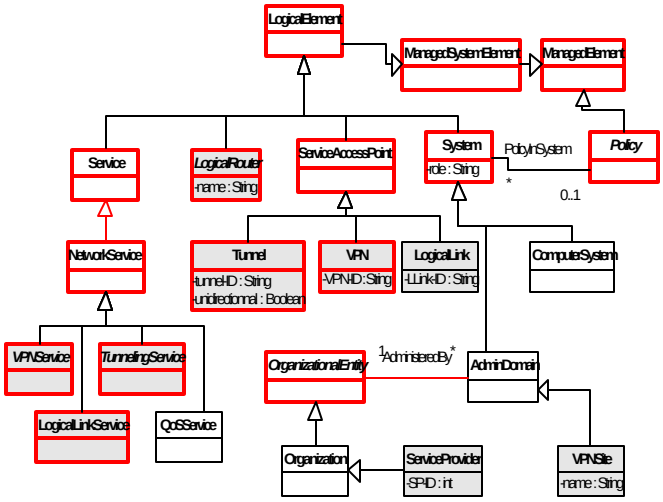

Figure 9. Overall VPN Information model

Policies provide rules of how network devices deliver service The extended model is mapped to XML schemas. The new data types corresponding to the new services are hence created. As per the defined framework, the extended derived objects are tagged "application". Application XML schemas describe those classes. The mandatory properties and relationships are validated through the use of those XML schemas.

New services can be incorporated in the policy platform in a transparent way. The Policy editor tool can hence adapt to the services added on the platform and the new policy created without the need to rewrite a large amount of code.

7. Conclusion

This article proposed a solution for evolving the network management platform to adapt to the constant changes imposed on the platform. The framework was presented for the case of the policy manager. Reducing the amount and the complexity of software changes is one of the biggest issues in any platform evolution. The proposed approach evolves the core information model and uses XML to communicate the model to all platform components.

Incorporating the description and validation of data on an XML Schema provides an easy way to make modification without having to go back to the code. Making XML Documents the data format for the Framework to communicate with other components ensure that proper validation will be done and that any component that want to use the Framework has to use the same XML Schema or extended versions of the Schema.

With the proposed framework, the experimentation demonstrates the ease of recreating the platform. We were able to adapt the policy platform to incorporate the new VPN policy management services by modifying the information model. The framework verifies the new model by validating the associations between the classes. Developers do not need to worry about relationships between classes shared between services nor do they need to modify the code. Finally the approach allows adapting to new configuration parameters as they are also described in XML schemas.

The proposed framework can deliver a more scalable platform as the information model is recreated with the needed extension to support the deployment of new services. It facilitates also the information sharing between services. The approach is less complex than the middleware solutions, as it enables an easy integration of new services or an easy removal of obsolete functionality. Finally, the approach can cope with the rapid changes in the network elements alleviating the burden of developers on the platform update task.

A management design pattern catalog can also be introduced in the framework [16]. It provides a set of patterns to ease the construction of CIM-based management models.

Fundamental to the approach is the in-depth understanding of the information model. The flexibility of the information models (CIM or other standard models) makes the analysis task for the developer non trivial.

Future work could focus on developing semantic tools that can help developers navigate and better use the model. The Web technology semantic [17] could be used to alleviate this problem. Network management ontology can also help define a vocabulary and a semantic that can clarify the model. An ontology will provide an explicit and formal representation of the domain knowledge expressing the objects belonging to that domain, the object properties and the relationship between those objects.

Other possible research avenue is to consider the combination of the component-based approaches with this solution in order to benefit from both approaches.

7. References

[1] Omar Cherkaoui, et all., Policy Management for Virtual Classroom over WAN network, SPIE ITCOM2001, Denver, August 2001.

[2] Cherkaoui Omar, Mounir Boukadoum, Alain Sarrazin, Managing Network-Based VPN with Policies, submitted paper.

[3] Cherkaoui Omar and Ahmed Serhrouchni, Managing Optical VPNs with Policies, APOC, Asia-Pacific Optical and Wireless Communications, 13-15 November, 2001, Beijing, China, IEEE and SPIE .

[4] Common Object Request Broker Architecture (CORBA/IIOP), Object Management Group (OMG), Revision 3.0, http://www.omg.org/technology/documents/spec_catalog.htm

[5] Joint Inter-Domain Working Group, X/Open and Network Management Forum, Inter-Domain Management Specifications: Specification Translation, April 1995.

[6] Implementing OSS Workflow: End to End to End, www.eftia.com/solutions/pdf/endtoend.pdf, March 2000.

[7] NGOSS: Development and Integration Methodology, Version 0.4, TeleManagement Forum, July 2001.

[8] Common Information Model (CIM) Specification, Version 2.x, DMTF. http://www.dmtf.org/standards/standard_cim.php.

[9] Policy Core Information Model version 1, rfc3060, IETF Network Working Group, B. Moore, E. Ellesson, J. Strassner, A. Westerinen, Nov. 2001.

[10] Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[11] Specification for the Representation of CIM in XML, Version 2.x, July 20th 1999, Distributed Management Task Force, Inc. (DMTF) http://www.dmtf.org

[12] IETF 3377: Lightweight Directory Access Protocol (v3): Technical Specification. J. Hodges, R. Morgan. September 2002.

[13] Lightweight Directory Access Protocol (v3): Attribute Syntax Definition, K. Dally, Editor, 19 June 2001 <draft-ietf-ldapbis-syntaxes-00> http://www.ietf.org/internet-drafts/draft-ietf-ldapbis-syntaxes-00.txt

[14] OpenGroup/NMF, "Inter-Domain Management: Specification Translation", The OpenGroup Prelim. Specification, March, 1997, http://www.rdg.opengroup.org/public/pubs/catalog/p509.htm.

[15] Fowler D., Virtual Private Network, Morgan Kaufmann Publishers, 1999.

[16] O. Mehl, M Becker, A. Köppel, P. Paul, D. Zimmermann and S Abeck, A Management-Aware Software Development Process Using Design Patterns, IM 2003, Colorado, March 24-28, 2003.

[17] Stephen Cranefield, UML and the Semantic Web, Proceedings of the International Semantic: Web Working Symposium, SWWS'01, July 2001