1998 International Computer Symposium
Workshop on Computer Networks, Internet, and Multimedia
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A MODEL-BASED OBJECT-ORIENTED TOPOLOGY SPECIFICATION FOR NETWORK MANAGEMENT

Jeng-Yueng Chen, Don-Lin Yang, and An-Chi Liu

Institute of Information Engineering
Feng Chia University, Taiwan, R.O.C.
Email: {jychen,dlyang,liu}@fcu.edu.tw

ABSTRACT

As the growth of Internet and web applications progresses repaidly, the network management becomes more and more important. Although most network management systems are able to collect network traffic data and alarms, they can not automatically analyze the collected data to find the cause of network faults, if any. As part of an ongoing project to develop an automated network faults diagnosis system, this paper presents a MODEL-based specification that can be integrated with NMS to provide the ability of describing the network topology. With the expanded MODEL class hierarchy, we can also show the OSI layer 2 devices in the topology map. To implement our work in the real world environment, we use the HP OpenView to display the processed topology data. Since relational databases are used in OpenView, we also derive suitable mapping rules to transform topology description objects to relational tables.

1. Introduction

In the recent years, a great deal of advancements and improvements on the network technology make the network applications more and more popular. When the network becomes more complex, the network management becomes more difficult. Among the challenges, the difficulty of network fault management lies in the alarm collection and the alarm analysis. A Network Management System (NMS) can be used to collect the network alarms, but one still needs experts to analyze the collected data and resolve the network problems. This semi-automatic process makes the network fault management ineffective and non-real-time.

Most of the current NMS products support visualized interfaces for network managers. They can automatically recognize the network environment by using the SNMP (Simple Network Management Protocol), and draw network topology maps based on the acquired information [1, 2, 3]. They also collect historical MIB (Management Information Base) information of the managed objects for trend analysis, and draw graphs for collected data. Most of the NMS store topology data and MIB data in flat files or relational databases. After configuring the managed objects, NMS can monitor the object's status, define event thresholds for MIB objects, and collect MIB information by polling [4]. Although network managers can manually add or delete topology elements, many attributes associated with the network elements still need to be acquired by SNMP. This makes it difficult for a network manager to change topology information.

By using the object-oriented technique, everything in the real world can be treated as an object. It is natural and convenient for us to model a system using the object concept. Therefore, object-oriented development can be used easily in network applications as well [5]. System Management ARTS (SMARTS) develops an object-oriented specification language, called MODEL (Managed Object DEfinition Language) to describe the network fault behavior and its propagation [6, 7, 8]. It is also very useful in the alarm correlation operation to assist a diagnosis system to automatically correlate the network alarms. Howevr, a diagnosis system needs the underlying network topology information to help identify the root causes. Unfortunately, the MODEL lacks proper support for network topology description. This may adversely impact the performance of fault management.

In this paper, we develop an effective object-oriented specification for describing the complete network topology. By integrating this function with the capability of alarm behavior description in [9], we can build an Alarm Specification Environment as part of an ongoing project that intends to develop an automated network fault diagnosis system.

The remaining of the paper is organized as follows. In Section 2, we introduce some related research including object-oriented specifications and techniques of mapping objects to relational tables. Section 3 specifies our object-oriented specification for network topology description. In Section 4, the transformation rules between objects and relational tables are explained. Section 5 describes our prototype implementation. Finally, in Section 6, we present conclusions and propose some possible directions of our future studies.

2. Related Research

The managed objects of a network environment can be effectively modeled with object-oriented concepts. A lot

of object-oriented specification languages are available, for instance, SMARTS MODEL, OMG IDL, and ITU-T SDL-92 can be used for network modeling. Each of the object-oriented specification languages mentioned above has its own special features and respective application domains.

The MODEL is an OMG IDL like specification language. It provides a simple syntax to describe network fault behavior [6]. It also provides instrumentation capabilities to automatically relate object attributes in the MODEL to SNMP MIB values. The MODEL can be integrated with the HP OpenView to provide a useful event correlation function. The OMG IDL is just a declarative language, not a full-fledged programming language [10, 11]. As such, it does not provide features like control constructs, nor is it directly used to implement distributed applications. The ITU-T SDL is used to model the structure and behavior of a communication system [12, 13]. The syntax of SDL is more complex than the other specification languages.

In summary, the MODEL is more suitable for cooperating with NMS to develop a network fault diagnosis system. The OMG IDL is more suitable for developing distributed systems. The SDL-92 is more suitable for modeling a telecommunication system.

Regarding the mapping for OpenView tables, many technologies have been developed to transform between objects and relational models. [14] uses a gateway mechanism to convert schemas between objects and relational models. [15, 16] present a useful mapping procedure to transform an object-oriented schema to a relational schema. [17] shows the basic issues of mapping objects into relational databases. [18] shows an ObjectShadow method to reflex the object concepts in relational tables. [19] uses the internal stub objects to be the interface between objects and databases. [20] presents an object-relational schema by adding type extension, complex objects, and inheritance to a relational DBMS. [21] uses three schemas architecture (external, conceptual, and internal schemas) for mapping objects into relational tables.

3. Object-Oriented Topology Specification

The MODEL language provides the function of describing network faults as part of the SMARTS' InCharge event correlation system [8]. Event correlation is the process of automatically grouping related events based on their underlying common cause that is defined and described by the MODEL language. It also automatically reduces the number of related events and identifies potentially hidden problems at the end.

The MODEL language enables a diagnosis system to automatically correlate the network faults and to provide network managers possible solutions. SMARTS emphasizes that their MODEL is independent to the physical network, but we believe that a correlation system needs the underlying network topology information to speed up its problem solving. Knowing the network topology, a network fault correlation process can easily identify the root causes. When the network topology is changed, we may observe different symptoms of the same network problem. In this circumstance, the event correlation system will be more difficult to find the root cause. Since the MODEL lacks proper support for network topology description, this may degrade the performance of fault correlation.

In order to provide effective network fault management, we expand the MODEL class hierarchy to have the ability of describing network topology. Fortunately, it is very easy to extend MODEL's object class hierarchy by adding relationships, attributes and events as needed [22].

In addition, because of the low cost and easy cabling, the star style network topologies that use hubs or switches to connect network nodes are more and more popular. However, most of the network management systems including the HP OpenView do not provide the detection of OSI layer 2 devices, such as hubs, switches or bridges in the network topology. They treat layer 2 devices as general network nodes. This will give network managers a network topology in a logical view rather than a physical view, such that the managers can not easily understand the real network configuration. If a network problem occurs, managers are not easy to recognize what's going on. The HP OpenView needs to cooperate with the third party products such as Onion Peel Solutions' Amerigo/L2 to provide the function of detecting layer 2 devices.

We devise two strategies for generating an OSI layer 2 topology map. One is to expand the MODEL specification to describe the complete layer 2 information. Here one needs to rewrite the MODEL specification and redraw the map when the topology is changed. The other strategy is to use additional procedures for the layer 2 map description. Standalone procedures can be employed to reorganize the topology map. In our research, we combine the two strategies. Our extended class hierarchy described in the next paragraph contains the classes to describe layer 2 topology and the rules like Amerigo/L2 to provide the ability of reorganizing layer 2 devices in the network topology [23]. The procedure to detect the layer 2 devices is shown in Fig. 1.

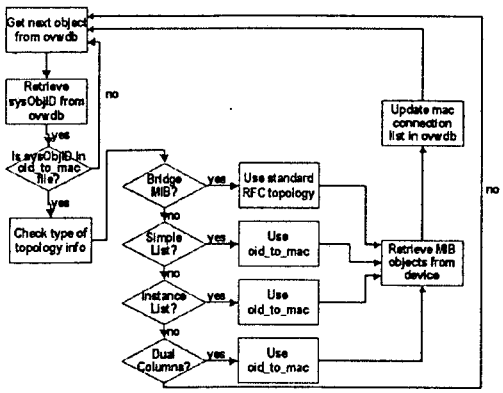

Fig. 1 Procedure to detect layer 2 devices

Our MODEL-based specification focuses on the description of network topology by expanding the MODEL's class hierarchy. We modify two original classes and add two new classes into the hierarchy. The expanded class hierarchy is shown in Fig. 2.

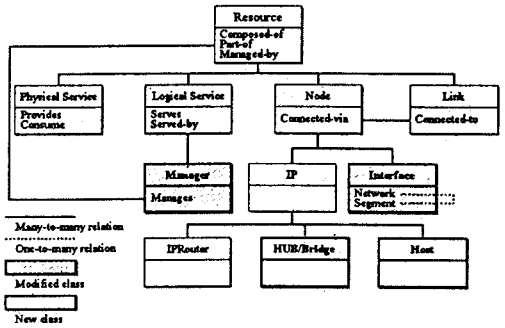

Fig. 2 Expanded MODEL class hierarchy

First of all, we modify the original **Interface** class to provide interface descriptions that include the network and segment descriptions. Second, we modify the original **Manager** class to provide a suitable capability of network management station description. Finally, we add two new classes called **HUB/Bridge** and **Host** to represent the OSI layer 2 devices and generic network nodes respectively.

```
interface IF: Node {
    instrumented attribute integer ifNumber;
    instrumented attribute integer ifType;
    instrumented attribute char[36]
            ifPhysaddr;
    instrumented attribute char[256] ifDescr;
    instrumented attribute integer ifOperStatus;

    attribute char[256] ifName;
    attribute char[36] topo_id;
    attribute char[15] ip_address;
    attribute char[15] ip_subnet_mask;
    attribute integer seg_type;
    attribute char[256] ip_network_name;
    attribute char[15]  ip_network_address;
    attribute char[15]
            ip_network_subnet_mask;
    attribute integer net_id;
    attribute integer seg_id;
    attribute char[256] seggment_name;
    attribute integer node_id;

    attribute long *Ref_Value_ifInOctets;
    attribute long *Ref_Value_ifInUcastPkts;
    attribute long *Ref_Value_ifInNUcastPkts;
    attribute long *Ref_Value_ifOutOctets;
    attribute long *Ref_Vallue_ifOutUcastPkts;
    attribute long *DEV_ifInOctets;
    attribute long *DEV_ifInUcastPkts;
    attribute long *DEV_ifInNUcastPkts;
    attribute long *DEV_ifOutOctets;
    attribute long *DEV_ifOutUcastPkts;

    instrument      SNMP;
}
```

Fig. 3 Modified Interface class

The **Interface** class provides the information about the network interfaces of managed objects. At the top of the modified **interface** class in Fig. 3, we add instrumented attribute commands to get the related information from the SNMP MIB data directly. The other added attribute commands are used in topology instances to provide the interface information such as interface name, IP address, network mask, network type, and so on. Each instance has designated values for some attributes because they are needed before using SNMP commands. For example, we can retrieve the IP address of an interface from ipAddrTable MIB, but first we must supply an IP address in the SNMP command to get its MIB data. We directly assign values of those attributes in the instances. Some attributes listed in this class such as Ref_Value_ifInOctets and DEV_ifOutUcastPkts are used for calculation purposes and are not related to network topology description.

```
interface Manager: LogicalService {
    attribute char *NMSName;
    attribute char[36] topo_id;
    attribute char[256] station_name;
    attribute char[256] station_description;
    attribute integer station_type;
    attribute integer access_mode;
    attribute char[256] snmp_community;

    NMSName = "HP OpenView";
}
```

Fig. 4 Modified Manager class

The **Manager** class provides the network management station's information to be used in the topology description. A management station is also a network node, so we may treat it as a general host and omit this class. The HP OpenView supports a delegation architecture that can manage a large network environment via cooperated management stations, and each management station maintains a local domain. We keep the **Manager** class as shown in Fig. 4 to record network management stations. The attributes of the **Manager** class are generic attributes used to describe a network management station. The topo_id field is used as a primary key, and the station_name is used to keep the hostname of this network management system. Each instance has designated attribute values because they are SNMP related attributes whose values must be provided in the SNMP commands. General speaking, SNMP MIB data only contains information of a managed object which acts as a management client. We can recognize if a managed object is a management station or not via the **Manager** class.

```
interface HUB: IP {
    instrumented attribute integer ifOperStatus;

    attribute char[36] topo_id;
    attribute char[256] hubname;
    attribute char[15] snmpaddr;
    attribute integer rootport;
    attribute integer port_number;
    attribute integer vendor;

    instrument SNMP;
}
```

Fig. 5 HUB/Bridge class

In general, we can treat a hub as a multi-port repeater and a switch as a multi-port bridge. The functions of hubs, bridges, and switches are similar to each other. The only difference between them is that switches and bridges may ignore unused packets by their filtering or forwarding algorithms. Because we focus on topology operations, hubs and bridges can be treated in the same manner. In order to reduce the complexity, we simply use the **HUB** class in Fig. 5 to describe all the layer 2 devices including hubs, bridges and switches in the topology. This class isn't provided in the original MODEL classes. The **HUB** class inherits attributes from the **IP** class because it needs to provide port information that is retrieved by SNMP commands. We first use instrumented attribute description to retrieve ifOperStatus MIB from the managed object. This MIB data is used to record the current operating status of the device. Other attribute commands are used to assign the HUB information such as name, port number, vendor, and so on.

```
interface Host: IP {
    instrumented attribute integer ColdStart;
    instrumented attribute integer ifOperStatus;

    attribute char[36] topo_id;
    attribute char[256] ip_hostname;
    attribute char[15] snmpaddr;
    attribute integer vendor;
    attribute integer topm_interface_count;

    instrument SNMP;
}
```

Fig. 6 Host class

The **Host** class provides the generic network node's information to be used in the topology. The **Host** class inherits attributes from the **IP** class. In Fig. 6 the attributes of the **Host** class include two instrumented and some generic attributes. Instrumented attribute ColdStar is used to check rebooting condition of a node, another attribute ifOperStatus is used to store the operating status of a network interface. These instrumented attributes are retrieved by SNMP commands. The attribute topo_id is an UUID identifier, and is the primary key used in the OpenView database. The attribute ip_hostname indicates the hostname of a node, and snmpaddr contains its IP address that is used to get SNMP MIB data. The vendor field stores an indexed integer used to indicate the manipulator of the node. The topm_interface_count indicates the number of interfaces the node has.

4. Objects to Relations Mapping

It is natural and easy for us to model a network system with the object concept. This is the reason why we choose the object-oriented specification language - MODEL to describe the network topology. Although the object concept is suitable for developing network applications, most of the commercial NMS products including the HP OpenView support relational databases only. The network topology described by using the MODEL-based specification needs to be converted to the formats that OpenView can recognize. They are stored in relational databases. We have to work not only on the object-oriented specification language MODEL, but also the corresponding relational database structure used to store topology information.

There are four types of databases used by the OpenView NNM. They include the OVW (OpenView Windows) object database, map database, topology database, and trend data database. The OVW object database manages all objects and field information for the HP OpenView. The map database contains presentation information specific to a topology map. Each map uses one map database in an HP OpenView environment. The topology database contains the topology schema used by

IP discovery and layout processes. This topology schema consists of tables for various types of IP objects or other related information. The trend data database contains the SNMP trend schema that consists of tables for NNM SNMP data collector facility.

The OpenView topology database consists of seven tables: **nnm_objects, nnm_interfaces, nnm_network, nnm_segments, nnm_nodes, nnm_stations,** and **nnm_topology** [24]. The **nnm_objects** is the most important table in the schema, and the other tables are all related to it. All the seven topology tables use the same primary key "topo_id." In addition to the seven tables, there is an **nnm_classes** table which contains an entry for each of the seven tables.

Mapping objects into relational model seems to be a big problem because the relational model cannot represent the object properties nicely in an object model. This is frequently called an impedance-mismatch [17, 18]. In order to avoid the impedance-mismatch, we have to derive rules for mapping objects to relational databases.

In our research, the strategy for mapping objects to relational databases is based on [17] as well as some ideas from [16, 21]. The transformation rules are divided into the following types:

- Mapping attributes to zero or more columns. If an attribute is used by instances for calculation purposes, it is not required to be stored in a database. These attributes are not mapped to any columns of tables.
- Mapping classes to one or more tables. There are three basic methods in mapping classes to relational tables.
  - — One table to store the entire class hierarchy. This solution is very simple. It is also easy to maintain the table because all of the data we need is in one single table.
  - — One table to store each concrete class [17]. That is, we can ignore unused data and store the data we really need in one single table.
  - — One table to store each object class. This approach conforms to object-oriented concepts better than the other approach.
- Mapping for relationships.
  - — One-to-one relationship: By intuition, one-to-one relationship is simple and easy to transform. To implement one-to-one relationships in a relational database, we just have to include the key of one table in another table. In other words, we only need to add a foreign key in one table.
  - — One-to-many relationship: To implement one-to-many relationships, we have to include the key of one table in another table. It is similar to one-to-one relationship mapping. The only difference is that the foreign key has to be added in the "many" side of the relationship.
  - — Many-to-many relationship: To implement many-to-many relationships, we have to use the associative table [17]. An associative table is a table whose purpose is to maintain the relationship between two or more tables in a relational database. The attributes in an associative table are traditionally the combination of the keys in the tables involved in the relationship. It is easy to implement associative tables by treating them as just another type of table. We assign associative tables their OID key field, and then add the necessary foreign keys to maintain the relationship.
- Mapping single inheritance instances to tables. Three approaches can be used:
  - — Each of superclass and subclass is mapped to a separate table.
  - — No superclass table: superclass attributes are replicated for each subclass.
  - — No subclass table: bring all subclass attributes up to the superclass level.
- Mapping disjoint multiple inheritance instances to tables.
  - — Each of superclass and subclass is mapped to a separate table.
- Mapping overlapping multiple inheritance instances to tables.
  - — Each of superclass, subclass, and generalization relationship is mapped to a separate table.

5. Implementation

In our system, we begin by constructing a MODEL script file that describes the network fault propagation model and network topology of our testbed environment as shown in Fig. 7. Then a Topology and Alarm COmpiler (TACO) is developed to read the MODEL script file and to generate a topology data file. The system flow is shown in Fig. 8.

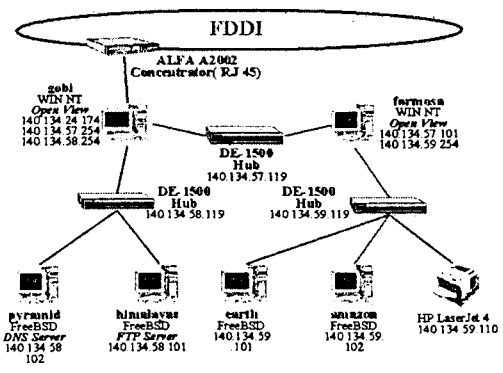

Fig. 7 Network management testbed

The TACO compiler analyzes the MODEL script to identify the propagation model and topology information.

The compiler generates an intermediate topology data file that can be stored in any types of databases. In our implementation we use a PERL program to read the intermediate file and put the data into the relational databases that an OpenView can access. Different PERL programs can be developed for other types of databases.

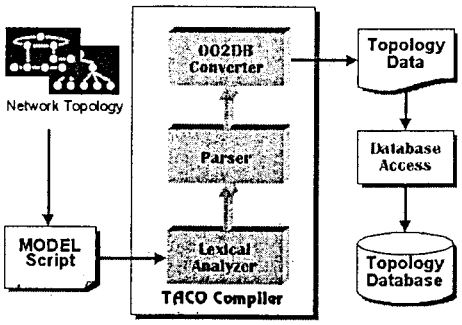

Fig. 8 System flow

Our MODEL-based specification is used to describe a network topology with four classes: **Host, HUB/Bridge, Interface**, and **Manager**. After constructing the MODEL description file for our testbed topology, we convert it to the relational tables. The HP OpenView stores topology data in seven tables: **nnm_objects, nnm_topology, nnm_networks, nnm_segments, nnm_nodes, nnm_interfaces, nnm_stations**. We analyze and decompose the MODEL's four classes to form the OpenView's seven tables. After being processed by the lexical analyzer and parser, the MODEL script is converted to table entries. Then the entries are inserted into OpenView's relational database. Finally, the OpenView displays an OSI layer 2 topology map as shown in Fig. 9 based on our MODEL descriptions.

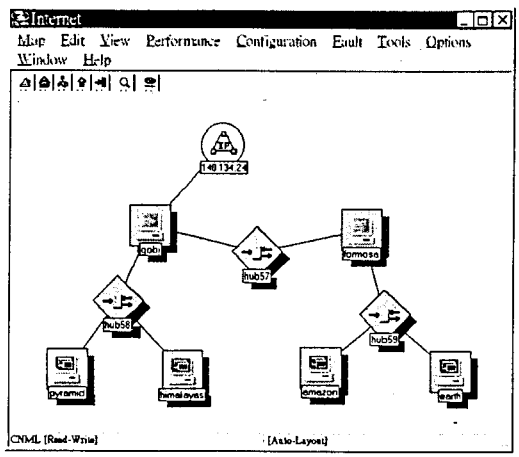

Fig. 9 Testbed topology map from TACO

6. Conclusion and Future Work

In this paper, we expand the SMARTS MODEL class hierarchy to provide the ability of describing network topology. With the network topology information, a network fault correlation system can identify the root causes much easier. We also develop a TACO compiler to convert the MODEL file to have the format that OpenView can recognize and display in its map windows. The TACO compiler contains the rules for mapping objects to relational tables that allow us to convert object-oriented MODEL specification to standard relational tables.

Our system has the following advantages. First, it provides an effective topology specification that can be used to describe network topology. For a non-existent network, it can be used to do network simulation. Second, our system provides the ability to present OSI layer 2 network maps. The topology map becomes more precise and easier to maintain. Third, we have shown an organized way to modify class hierarchy as needed. Fourth, the TACO compiler generates an intermediate file without directly accessing the database. Thus, the compiler is independent to the database used in the NMS. Finally, our topology information can greatly improve the correlation of netwrok faults when it is integrated with fault propagation model.

One of our future work is to develop a visualized GUI interface for constructing the MODEL script. The GUI can provide a better way for users to configure their network topology, and automatically generate a corresponding MODEL description file. As new network technologies continue to blossom, we would like to be able to expand the MODEL to display these new technologies more easily.

References

[1] HP, *HP OpenView: Using Network Node Manager*, HP, Mar. 1997.

[2] IBM, *NetView for AIX: Administrator's Guide Version 3*, IBM, Sep. 1994.

[3] Sun, *Solstice Enterprise Manager 2.1: A Technical White Paper*, Sun, 1997.

[4] A.C. Liu, *Network Management*, Information and Computer Magazine, 1995.

[5] R. Gupta and E. Horowitz, "An Object-Oriented Model for Network Management," *Object-Oriented Database with Applications to CASE, Networks, and VLSI CAD*, Prentics-Hall, Inc., pp. 283-95, 1991.

[6] A. Mayer, S. Kliger and S. Yemini, "Event Modeling with the MODEL Language," System Management ARTS, 1997

[7] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, "High Speed and Robust Event Correlation", *IEEE Communication Magazine*, pp.82-90, May 1996.

[8] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, "High Speed & Robust Event Correlation,"

System Management ARTS, 1997.

[9] E.T. Liang, "TCP/IP Alarm Transformation in a Network Management Environment," 1998, submitted to ICS '98.

[10] J. Rosenberger, "Mastering the Interface Definition Interface (IDL)," *Sams' Teach Yourself CORBA in 14 Days*, Sams Publishing, pp. 37-61, 1998.

[11] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, pp. 46-55, Feb. 1997.

[12] ITU-T Recommendation Z.100, "CCITT Specification and Description Language (SDL)," *Recommendation Z.100*, Mar. 1993.

[13] ITU-T Recommendation Z.105, "SDL Combined with ASN.1 (SDL/ASN.1)," *Recommendation Z.105*, Mar. 1995.

[14] C. Lee, et al., "An OODB Gateway to RDB," *Proceeding of 7th Workshop on Object-Oriented Technologies and Applications*, pp. 367-76, Sep. 1996.

[15] M. Blaha, W. Premerlani and J. Rumbaugh, "Relational Database Design Using an Object-Oriented Methodology," *Communications of the ACM*, Vol. 31, No. 4, pp. 414-27, Apr. 1988.

[16] S. Y. Hsieh, C. K. Chang, P. Mongklowat, W. Pilch Jr. and C. C. Shih, "Capturing the Object-Oriented Database Model in Relational Form," *IEEE*, 1997.

[17] S. Ambler, "Mapping Objects to Relational Databases," *http://www.ambysoft.com/mappingObjects.html*, Apr. 1998.

[18] M. Fussell, "Foundations of Object Relational Mapping," *http://www.chimu.com/publications/objectRelational/index.html*. Jul. 1997.

[19] H. Vogelsang and U. Brinkshulte, "Presisent Objects In A Relational Database," *http://www-ima.ira.uka.de/mitarbeiter/vogelsan/ps/woon96.ps.gz*

[20] M. Stonebraker, *Object-Relational DBMS: The Next Great Wave, Morgan-Kaufman Publishers*, 1995.

[21] J. Rumbaug, M. Blaha, W. Premerlani, F. Eddy and W. Lorenson, *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

[22] A. Dupuy, S. Sengupta, O. Wolfson, and Y. Yemini, "NetMate: A Network Management Environment," *IEEE Network Magazine*, 1991

[23] Onion Peel Solutions, "Amerigo for HP OpenView UNIX: User and Installation Guide," Onion Peel Solutions, 1998

[24] HP, *Using Relational Databases with HP OpenView Network Node Manager*, HP, Apr. 1997.