Network Management for the 90s

Luca Deri
IBM Zurich Research Laboratory[1], *University of Berne*[2]

The increasing complexity and heterogeneity of modern networks has pushed industry and research towards a single and consistent way of managing networks. The effort to define a single industry-standard API for network management basically failed because it did not address aspects like complexity and ease of programming. Recently, a common approach is to map established network management standards into another object model, often based on the emerging Corba standard. Unfortunately even this approach has shown many drawbacks mostly related to the significant amount of code that has to be linked with the final application and to the many limitations and imperfections of the mapping itself.

This paper describes a new approach to inter-domain management that attempts to overcome the limitations of current solutions. The goal is to allow people to write hybrid CMIP and SNMP based network management applications, using a single and simple object model. Relevant characteristics of this approach are: light, extensible, object-oriented, language-neutral, built upon software-components, string-syntax based, Internet-ready. This demonstrates that it is feasible to implement simple and light applications for inter-domain management without the need to use expensive or complex technologies.

Keywords: Network Management, Object-Oriented Programming, Software Components, Java.

1. Introduction

The increasing complexity and heterogeneity of modern networks and the advent of distributed computing are making network management both more important and complex. In this decade many companies and research institutions have attempted to simplify the scenario by defining a single and consistent way for managing heterogeneous networks based on both CMIP [8] and SNMP [2]. In this view, X/Open has defined an industry standard C-based API called XOM/XMP [17][18], able to unify these two dominant network management protocols. The idea was to allow people to write applications using a single API in order to simplify the integration of code written by different people.

Recently, the increasing popularity of the Corba [9] industry-standard pushed many people to write mappings between CMIP/SNMP and Corba [1][6] based on the assumption that Corba will become the network management standard of the future and that everybody will use it instead of CMIP and SNMP. Despite their efforts, today there are many different mappings available that usually do not fully support CMIP/SNMP. Another drawback is related to the significant amount of code that must be generated for implementing these mappings and that has to be linked with the final application. Other than this, a network management expert that intends to write a management application must learn Corba, IDL (the language used to specify the Corba interfaces), how the mappings have been defined, and must have an ORB (Object Request Broker) installed somewhere. It is clear for instance that the initial vision of SNMP to be simple and light has been jeopardized.

1.1. What to do then ?

This work is based on the idea that so far network management has been considered like a special software engineering problem where solutions must be built ad-hoc and cannot reuse widely established concepts. Today most of the network management people come from the "Unix and C" school and ignore new concepts and innovations like software components [14], and truly object-

---

[1] Luca Deri, IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland. Email: lde@zurich.ibm.com, WWW: http://www.zurich.ibm.com/~lde/.

[2] Universität Bern, Institut für Informatik und angewandte Mathematik, Software Composition Group, Neubrückstrasse 10, CH-3012 Bern, Switzerland. Email: deri@iam.unibe.ch, WWW: http://iamwww.unibe.ch/~deri/.

oriented software development (most of the code is object-based but not object-oriented[3] [16]). Additionally, it is a common belief to pretend to solve a problem generating code for all the possible situations (for instance XOM/XMP and many CMIP/SNMP to Corba mappings generate a class for each data type) instead of trying to define a way to simplify the problem. The advent of Java [12] and TCL [10] demonstrated that the short reign of native-code-generating-object-oriented compiler is about to be over [15]. Internet and the market demand light, machine-independent applications capable to roam from machine to machine. This requires light applications simple enough to be downloaded from the network and that do not need excessive system resources.

These days programmers want to use programming concepts instead of protocol concepts. All the modern programming languages support exceptions and programmers are used to them, therefore it is time to replace protocol errors with exceptions and to avoid to execute a lot of code or to convert information many times just to get the value of an integer attribute.

In the next section a new approach to inter-domain management is described that attempts to overcome limitations of current solutions. The goal is to allow people to easily write light network management applications that fully support both CMIP and SNMP using a single and simple object model. These application are Internet-ready and can be integrated with the world-wide web using the Java bindings here described. The guidelines and the code examples have been drawn from implementation experience and in the course of designing and implementing commercial products and research prototypes. Familiarity with the object-oriented terminology and the Java programming language, thought not strictly required, it is certainly useful.

2. Merging Network Management Standards

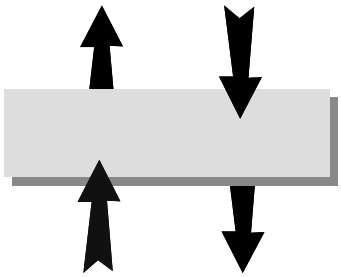
This section shows how the process of integrating network management standard such as CMIP and SNMP is done. First of all the difference between the two standards is hidden, then an infrastructure is built. At last the integration is done.

2.1. Hiding Differences

Network programmers need a single way to manipulate instances of various object models. The main problem arises from the data types that have to be managed. In SNMP this is easy to handle because the different data types are about ten. CMIP is a lot more flexible in this respect and it allows the user to define new data types. Due to this, the number of data types that a network management application has to handle is not determined a priori. Therefore a way has to be defined to handle different data types of arbitrary complexity.

The solution proposed here is based on string notation [5]. In this view, every data type is represented using strings. Aggregate data types like sequences or sets are a composition of basic data types like integer or boolean. The fact to have a single data type makes things simple and allows applications written in whatever language to use it, even if this representation slows down the system code that uses data types to speed operation. Nevertheless experience derived from using string representation on various commercial applications has shown that this inefficiency is rather limited especially on Unix™ systems that are able to handle string very efficiently. Despite the advantages of a string-based notation, some users may want to define information using a different object model. For this reason, utilities for handling aggregate types are been provided in order to make the conversion smooth and efficient. Programmers define data values using the string representation and then the encoder/decoder module converts this string to BER (Basic Encoding Rules) and back.

[3] In 1987, Peter Wegner of Brown University introduced some order to the OO community. In Wegner's terminology, systems that have classes and support implementation inheritance can be properly called object-oriented, while those without implementation inheritance are characterized as object-based.



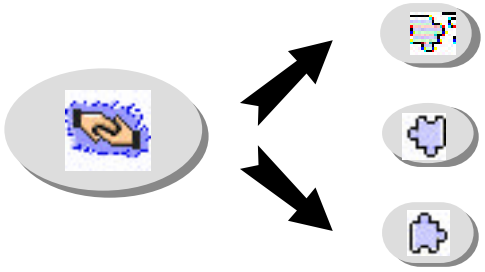
The conversion is based on metadata information. In the IBM stack [7], the ASN.1 and GDMO compilers, compile input documents into a data file that is read by the encoder/decoder at startup time. These data files contain information about the data types and object-model dependent information. In the case of CMIP, data files contain information about managed object classes, name binding, actions and notification. In case of SNMP information concerning object identifiers and the textual description of the various attributes are stored in separate files. At runtime it is possible to access this information not only for encoding/decoding purposes but also for querying information about a particular attribute or action. This kind of information is very useful in browser applications or to help during their work preventing for instance the request of wrong operations. Once the problem to define a single format to represent various syntaxes is solved, the difference between connectionless and connection oriented protocols has to be hidden.

In SNMP there is no concept of connection and every message is sent independently usually over UDP. In CMIP every protocol request travels over an association that has to be established first and then closed when the communication is over. Users should not be concerned about associations and they should think only in terms of objects. In the IBM stack, associations have been implemented transparently. In a simple directory service, similar to the one defined by many XMP implementations, there are stored information about known peers and about the instance tree they manage. Every time a request is sent to the stack, the object instance is analyzed and the correct agent managing that instance is identified and an association is opened. An association stays alive until it is closed either by one of the partners or when an error occurs (for instance if the connection goes down).

Thanks to the string representation and to the automatic association handling, it is now possible to transparently manipulate remote instances using both SNMP and CMIP in a single and uniform way.

2.2 Building up the Infrastructure

In an effort to integrate the network management world with the web, the author developed an application named Liaison [4]. This application, based on a special type of software components called Droplets [3], enabled web users to access network resources based on CMIP and SNMP. Droplets have the ability to be replaced and added at runtime allowing to dynamically modify and extend the behavior of the application that contains them. The core element of Liaison is the Proxy server.



Each droplet, built upon shared libraries, implements one or more services. These components cooperate through the Proxy that takes care of the communication with the outside world. Proxy implements the HTTP protocol hence remote web users can access it directly without the need to have,

for instance, CGI[4] applications that interface the HTTP server with the Proxy itself. This solution presents several advantages in terms of performance and configuration. Proxy comes with droplets that implement all the CMIP and SNMP operations, a basic directory service and a metadata repository for SNMP. Additionally there are a couple of droplets that have the ability to query the metadata information contained inside the OSI stack. Thanks to the flexibility of droplets it has been easy to develop some more and to integrate them with the Proxy itself. The idea is to implement a droplet for each management CMIP/SNMP operation and then cooperate with the existing droplets in order to reuse the services they provide especially with respect to the metadata access. This demonstrates how powerful software components are and how they allow to reuse existing services and then to incrementally build applications instead of starting from scratch every time.

Each droplet communicates with remote requesters through the Proxy that acts like a tunnel application. Being the droplet code reentrant, concurrent requests can be served. The Proxy communicates with remote clients over HTTP and it exchanges messages with the droplets. The usage of the HTTP protocol has been preferred because it is simple, well established and it allows to flow through firewalls allowing to manage hosts everywhere on the network. Since each droplet exploits existing services, the average size of a droplet that implements either a CMIP or SNMP primitive is about 10 Kb. The memory footprint is very small too because droplets allocate only temporary memory needed to process the request and they do not store any state. All the Proxy structure is stateless and it is up to the Proxy clients to maintain state information. This contributes to keep the code simple and it allows to easily replace droplets at runtime.

Management requests, received by the Proxy via HTTP, have been designed in order to reduce the amount of information exchanged with the remote client and to be flexible enough to allow to handle not only CMIP and SNMP but even new protocols. For this reason the client sends the information like a sequence of `{name, value}` elements. Suppose for instance to set the attribute `systemTitle` of the instance `systemId=(name Telco)` of class `system` to `(nothing NULL)`. In this case the client issues the following request:

```
/CMIP/SET/MIBCTL/?objectClass=system&objectInstance=systemId%3d%28name+
Telco%29&systemTitle=%28nothing+NULL%29
```

[5]

The order is not significant. In CMIP the `context` field, if present, contains the AE-Title of the stack that has to be used for communications (if absent the default one is used), whereas in SNMP it contains the TCP/IP address of the SNMP agent. The other entries contain information related to the operation. Proxy, based upon the request type (for instance CMIP SET), sends the HTTP request received by the remote client to the corresponding droplet. Such droplet is responsible executing the operation and returning the HTTP response over the same socket that is still connected with the requester. If the operation is successful the droplet returns a document of type `Content-type: text/plain` containing the response where each response entry is separated with a carriage return. In the example above it returns

```
HTTP/1.0 200 OK
Server: IBM ZRL Proxy Server
Date: Fri, 28 Jun 1996 12:30:16 GMT
Content-type: text/plain
Content-length: 73

currentTime
Fri, 28 Jun 1996 12:30:16 GMT
systemTitle
(nothing NULL)
```

In case of error the type of error and the error information, if any, is returned. This way of exchanging information allows to avoid sending unneeded fields like `scope` or `filter` in case they are set to default, and it allow to express cases like setToDefault, in the example above the value for `systemTitle` would have been omitted, without the need to introduce additional complexity. Additionally, since the response is in a human readable form, it is simple for simple tools as shell scripts or text processing tools to handle it efficiently. It is important to remark that for SNMP the

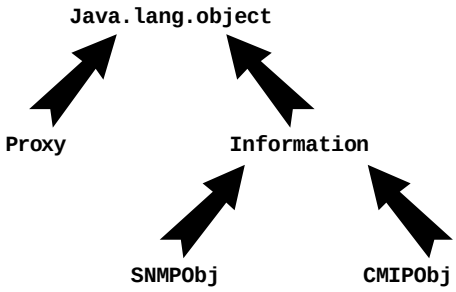
[4] CGI (Common Gateway Interface), defined by NCSA, allows to interface applications to HTTP servers.

[5] This URL is built automatically by the Proxy and absolutely no knowledge about the HTTP protocol and the encoding rules is required by the user.

information is expressed exactly the same way allowing to have a consistent information exchange format.

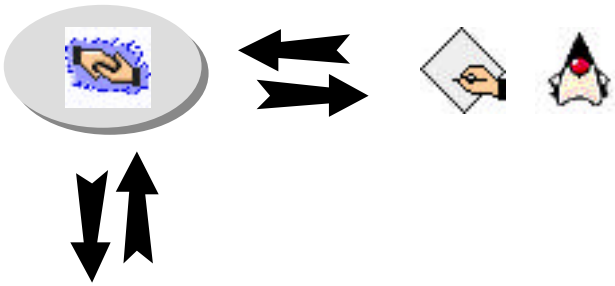
2.3 Application-side Bindings

Because clients communicate with the Proxy over HTTP and because the data exchange type is based on strings, it is easy to write bindings in whatever programming language either object-oriented or not. For the sake of simplicity bindings described in this section are written using Java™. Similar considerations can be done for other languages such as C++ or TCL. The class hierarchy is quite simple.



The class `Proxy` is responsible for handing communications with the Proxy application. It transparently sends the requests and receives the responses. The class `Information` contains the information relative to the request and to the response(s), stored in an object of class `java.util.Hashtable` that are passed as input parameter to an instance of class `Proxy`. Subclasses `SNMPObj` and `CMIPObj` implement some high level manipulation functions for manipulating the input/output information and invoking `Proxy` methods whenever a request has to be issued. These subclasses have been provided to further simplify the access to the `Information` and `Proxy` classes and have to be considered like pure facilities.

Requests can have single or multiple responses returned in case of a CMIP scoped request or a SNMP walk. When multiple responses are returned they are insert in a `java.util.Vector` that is returned as output parameter. In case of single response, the returned values replace the actual ones in the input `SNMPObj` or `CMIPObj` object. In this way the input object is transparently updated with the return values. The example below clarifies this situation.



```
Proxy p;
CMIPObj cmip;
try {
  p = new Proxy("adl.zurich.ibm.com" /* Host where Proxy is running */);
  cmip = new CMIPObj(p, "MIBCTL" /* AE-Title */);
  cmip.SetObjectClass("system");
  cmip.SetObjectInstance("genericNe   tworkId=Net1@systemId=(name Telco   )");
  cmip.SetAttribute("systemTitle", "");
  cmip.CMIPGetAttributes(); // Issue the CMIP M-GET request
  System.out.println(  “systemTitle is:”+cmip.GetAttribute(“systemTitle”));
} catch(Exception e) { System.out.println(“Error: “    +e); }
```

When the `CMIPGetAttributes()` method is called, the Proxy sends back the CMIP response containing `objectClass`, `objectInstance`, `currentTime` and `systemTitle`. `CMIPObj` receives those values and puts them in the `cmip` instance itself. In case of `systemTitle`, the original empty value is replaced with the one returned by Proxy. `currentTime`, not present in the request, is added to the input object. This approach allows to easily get and set attribute values other than allowing to issue operations in a few lines of code. If a request fails for whatever reason an exception of class `ProxyException` is raised: users should not deal with protocol errors but they should interact with remote objects only using programming constructs. This is very important because programmers do not have to change their programming style using familiar concepts like exceptions. When an exception is raised, an error code is returned together with the receiver error response that does not affect the input object which remains unmodified.

The `Information` class and its subclasses `SNMPObj` and `CMIPObj`, greatly simplifies and reduces the code users have to write:

- a `SNMPObj` or `CMIPObj` object represents a hook to an instance or attribute independently from the operation that will be issued: this allows to issue different operations using the same input object
- parameters such as scope, filter, sync (CMIP) or community (SNMP) are handled transparently: if not present or set to default they are not sent to the Proxy that will then use the defaults
- default values are expressed using empty ("") values instead of using special flags or data structures.

Additionally, this solution allows to save bandwidth because only the needed attributes are exchanged between the Proxy and the Java application and because unmodified attributes, for instance objectClass in a CMIP response, are not transmitted. Classes `SNMPObj` or `CMIPObj` other than issuing protocol requests, allow to retrieve metadata information and to convert object identifiers that can be expressed in both numeric or symbolic form.

```
public class Information extends java.lang.Object {
  [ … ]
  public void SetAttribute(String name, Object value)
                                    throws IllegalArgumentException       { … }
  public Object GetAttribute(String name)                                 { … }
  public void RemoveAttribute(String name)                                { … }
  public Enumeration GetAttributeValues()                                 { … }
  public Enumeration GetAttributeNams()                                   { … }
  public  void RemoveAllAttributes()                                      { … }
}

public class CMIPObj extends Information {
   [ … ]
   public void Se  tObjectClass(String val)                               { … }
   public String GetObjectClass()                                         { … }
   public void SetObjectInstance(String val)                              { … }
   public String GetObjectInstance()                                      { … }
   public Information GetActions() throws ProxyException                  { … }
   public Information GetNameBindings() throws ProxyException             { … }
   public String GetSyntaxInfo(Strin   g syntax) throws ProxyException    { … }
   public String ConvertOID(String oid) throws ProxyException             { … }
   /* Management Operations */
   public void CMIPCreateObject() throws ProxyException                   { … }
   public void CMIPDeleteObject() throws ProxyException                   { … }
   public Vector CMIPDeleteContainedInstances() throws ProxyException       { … }
   public void CMIPGetAttributes() throws ProxyException                  { … }
   public Vector CMIPGetContainedInstances() throws ProxyException        { … }
   public void CMIPSetAttributes() throws ProxyException                  { … }
   public Vector CMIPSetContainedInstances() throws ProxyException        { … }
   public void CMIPPerformAction() throws ProxyException                  { … }
   public Vector CMIPPerformActionContainedInst() throws ProxyException             { … }
   public int NotificationsAvailable() throws ProxyException              { … }
   public Information WaitForNotific(int timeout) throws ProxyException             { … }
   public void DeleteEFD() throws ProxyException                          { … }
   public void CreateEFD(String inst, String fltr) throws ProxyException            { … }
}
```

```
public class SNMPObj extends Information {
   [ … ]
   public String SNMPGetAttributeInfo(String sntx    ) throws ProxyException      { … }
   public String ConvertOID(String oid) throws ProxyException           { … }
  /* Management Operations */
  public Vector SNMPWalk() throws ProxyException                        { … }
  public void SNMPGetAttribute() throws ProxyException                  { … }
  public void SNMPGetNextAttribute() throws ProxyException              { … }
  public void SNMPSetAttribute() throws ProxyException                  { … }
 }
```

Respectively:

- `GetActions` returns the CMIP actions that can be performed by the object
- `GetNameBindings` returns all the name bindings of the object, useful for creating managed objects
- `GetSyntaxInfo` returns the requested ASN.1 syntax in HTML format
- `ConvertOID` is responsible for converting object identifiers
- `SNMPGetAttributeInfo` returns the attribute description specified in the RFC.

Metadata information is either contained in the Proxy (the object identifier mapping information and the RFC information) or it is retrieved from the stack (ASN.1 information).

The class `Proxy` is responsible for handling the communications with the Proxy.

```
public class Proxy extends java.lang.Object {
   public Proxy(String hos  t) throws UnknownHostException,  IOException  { … }
   public void dispose() throws IOException                             { … }
   public Vector SendRequest(Str  ing operation, String context,
             Information input) throws ProxyException                   { … }
   public String SendOffLineRequest(String operation,
             String context, String in  put) throws ProxyException      { … }
   […]
}
```

Respectively:

- `SendRequest` sends a protocol request to the Proxy using as input `Information` that contains information related to the target managed object
- `SendOffLineRequest` sends an off-line request to the Proxy (for instance OID mapping).

The `context` parameter contains protocol-related information. In case of CMIP it contains the agent AE-Title whereas for SNMP it contains the TCP/IP address of the SNMP agent. It is worth to remark that the `operation` parameter is a string (for instance “CMIP_Get”) used by Proxy to identify the droplet that implements such operation. This approach will allow in the future to support further protocols and object models such as Corba or Tina without the need to modify the classes `Proxy` and `Information` hence to define a new object model. In fact it is sufficient to add some new droplets and define some new values for the `operation` parameter (for instance “Corba_Get”).

Like it has been said already, classes `SNMPObj` and `CMIPObj` are pure facilities and both CMIP and SNMP requests can be based directly on `Information`. Looking at their definition is quite simple to see this. The method `SetObjectClass(String val)` is defined like `SetAttribute("objectClass", val)` or the method `CMIPGetAttributes()` internally calls `proxy.SendRequest(“CMIP_Get”, agentAET, super/* Information */)`.

The decision to base this work on the Proxy derives from the fact that, especially with the advent of Internet, applications have to be as light as possible. It does not make sense to duplicate part of the functionality of the Proxy on each network management application. Also, in case of CMIP, the Proxy should be installed by the ones who install the stack and the OSI agent, if any, and Proxy users should not be responsible for configuration or maintenance tasks. Nevertheless, the Proxy is quite compact and thanks to its open droplet-based architecture it allows to be installed, replicated on various hosts and tailored easily using very little space. In total, the Java classes just described are about 8 Kb in total. This allowed to easily use them inside applets that are downloaded by remote web browsers.

2.4 Related Research

The table below compares this work with similar efforts being undertaken in this area. All compared models have runtime bindings and are runtime type-checked.

	Proposed Solution	**Tcl-MCMIS [13]**	**Scotty [11]**	**XMP [18]**	**GOM [1]**
Object-Oriented	Yes	No	No	No	Yes
Application Size	Light	Medium	Medium	Medium / Large	Large
Ease of Use	Easy	Easy	Easy	Difficult	Easy
Typing	Weak	Weak	Weak	Strong	Weak
Currently Supported Object Models	CMIP / SNMP	CMIP	CMIP / SNMP	CMIP / SNMP	CMIP / Corba
Language Bindings	Java / C++	TCL	TCL	C	C++
Data Representation	String	String	String	XOM	GOM (11 types)
Metadata Access	Yes	No	No	Impl. Dependent	Yes
Pre-requisites	Java VM	Tcl	Tcl	XOM / XMP	Obj. Broker

This table shows that the proposed solution is preferable over the listed alternatives in many important aspects like application size and ease of use. Other solutions based on TCL, despite their simplicity and their similarity with the approach here described, have a bigger application size and hence cannot run unmodified on different platforms due to their use of C / C++ libraries that interface TCL with CMIP / SNMP resources. Finally, the proposed solution thanks to the Java application bindings and to its limited size enables the construction of a new class of network management applications that can be easily integrated with the world-wide web and Internet.

3. Conclusion

This paper shows a new approach to inter-domain management that overcomes limitations of many current solutions. Main characteristic are: ease of use, language neutral bindings, based on established technology like HTTP, small size, open to the integration of additional protocols. Bindings for the Java language have been developed and then enabled the creation of a new class of Internet-ready network management applications that can be built upon applets and integrated on web browsers. Usage of droplets demonstrated that it is possible to easily extend existing applications, like the Proxy, and then to add services and extensions not planned in the initial design. This will likely allow in the future to support new protocols and object models other than CMIP and SNMP.

This work demonstrates that it is possible and feasible to easily implement network management applications. The complete compliance with both SNMP and CMIP allows to write hybrid an powerful applications using a single object model and greatly reduces the development efforts for the production of final applications. Despite its broad field of applications, this solution does not introduce additional costs nor require the usage of relatively expensive technology like object brokers, and results are also attractive in terms of both efficiency and size of code.

4. Acknowledgments

The author would like to thank Bela Ban and Dieter Gantenbein for their suggestions and valuable discussions other than the users of *Webbin' CMIP*[6] who have greatly stimulated with all their comments and suggestions.

[6] A publicly accessible demo is located at http: / / misa.zurich.ibm.com / Webbin /

5. References

[1] B. Ban, *Towards An Object-Oriented Framework for Multi-Domain Management*, IBM Zurich Research Laboratory, December 1995.

[2] J. Case, M. Fedor, M. Schoffstall and C. Davin, *The Simple Network Management Protocol (SNMP)*, RFC 1157, May 1990.

[3] L. Deri, *Droplets: Breaking Monolithic Applications Apart*, IBM Research Report RZ 2799, September 1995.

[4] L. Deri, *Surfin' Network Management Resources Across the Web*, Proceedings of 2nd Intl IEEE Workshop on Systems and Network Management, Toronto, June 1996.

[5] D. Geiger, W. Allen, A. Majtenyi and P. Reder, *IBM cmipWorks: Technical Paper* IBM, March 1994.

[6] J. Hierro, *Architectural Issues For Using Corba Technology in OSI Systems Management*, Append of draft to XoJDM forum, August 1994.

[7] IBM Corporation, *Agent User's Guide for the IBM NetView TMN Portable Agent Facility, Release 2, Version 1*, IBM TMN Products *GC31-8209-00*, October 1995.

[8] ISO/IEC, CCITT, *Information Technology - OSI, Common Management Information Protocol (CMIP) - Part 1: Specification* ISO/IEC 9596-1, CCITT Recommendation X.711, 1991.

[9] *The Common Object Request Broker: Architecture and Specification*, 1.2 edition, Object Management Group, 1993.

[10] J. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, ISBN 0-201-63337-X

[11] J. Schönwälder and H. Langendörfer, *Tcl Extensions for Network Management Applications*, Comp. Science Dept., Univ. Of Braunschweig, May 1995.

[12] Sun Microsystems, *The Java Programming Language*, Addison-Wesley, 1995, ISBN-0-201-63455-4.

[13] T. Tin, G. Pavlou and R. Shi, *Tcl-MCMIS: Interpreted Management Access Facilities*, Proceedings of DSOM '95, October 1995.

[14] J. Udell, *ComponentWare*, Byte, May 1994

[15] P. Wayner, *Net Programming for the Masses*, Byte, February 1996.

[16] P. Wegner, *Dimensions of Object-Based Language Design*, Proceedings OOPSLA '87, ACM SIGPLAN, vol. 22 no. 12.

[17] X/Open Company Ltd. *OSI-Abstract-Data Manipulation API (XOM)*, X/Open Document C315, February 1994.

[18] X/Open Company Ltd. *Systems Management: Management Protocol API (XMP)*, X/Open Document C306, March 1994.