A Component-based Architecture for Open, Independently Extensible Distributed Systems

Inauguraldissertation der Philosophisch-naturwissenschaftlichen Fakultät der Universität Bern

vorgelegt von

Luca Deri

von Pisa, Italien

Leiter der Arbeit: Prof. Dr. Oscar Nierstrasz
Institut für Informatik und angewandte Mathematik
Universität Bern

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, den 18. Juni 1997

Der Dekan:
Prof. Dr. Horst Bunke

Every effort has been made to ensure that the information contained in this book is accurate. The author is not responsible for typographical errors.

Author's contact addresses:

IBM Research Division
Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon
Switzerland
Email: lde@zurich.ibm.com
WWW: http://www.zurich.ibm.com/~lde/

Universität Bern
Institut für Informatik und Angewandte Mathematik
Software Composition Group
Neubrückstrasse 10
3012 Bern
Switzerland
Email: deri@iam.unibe.ch
WWW: http://iamwww.unibe.ch/~deri/

Table of Contents

Abstract

Object-oriented programming (OOP) has significantly changed the way software applications are produced and has allowed many problems that affected traditional programming to be overcome. Unfortunately many object-oriented systems misused OOP techniques, failed to address issues such as application extensibility, and produced monolithic systems that are difficult to manage and tailor. Especially in the context of communication networks, some companies have developed huge management systems very powerful and rich in terms of functionality and tools, but that do not address problems relating to application development. The natural consequence is that management applications are monolithic, difficult to tailor and configure, and are system resources-hungry, preventing them from running on hosts having limited power.

These problems have been the driving force to define Yasmin, a new architecture for applications. Yasmin attempts to overcome problems that affect traditional applications by defining an application framework characterised by the following properties: 1) light and simple kernel, 2) based on a new type of pluggable software components called droplets, 3) built entirely on object-oriented technology, as well as 4) extensible, easy to tailor and distribute. The idea behind Yasmin is to build component-based applications that can be composed by the user, who can add or replace components at runtime. By enforcing the component boundaries, Yasmin prevents components from making assumptions concerning other components, hence reducing component interdependency and making them easy to reuse in different contexts. Liaison, a Yasmin-based application, has been developed in order to validate the architecture. Liaison has been applied to selected network management problems and clearly demonstrated that Yasmin-based applications efficiently overcome problems that affect conventional management applications. In addition, Liaison proved that common beliefs such as management applications need a long development time, need large computing resources, are difficult to scale and tailor, and have a poor performance, are not at all correct.

In conclusion, Yasmin a) allowed traditional problems to be overcome, 2) demonstrated that droplets can be effectively used to create distributed applications that are easy to extend, scale and tailor, and 3) proved to be a feasible architecture for management applications and in general for open distributed application systems.

Sommario

La programmazione orientata agli oggetti ha cambiato profondamente il modo di produrre applicazioni ed ha permesso di risolvere problemi riguardanti applicazioni scritte in modo tradizionale. Purtroppo molti sistemi orientati agli oggetti hanno abusato di alcune tecniche programmative, non sono riusciti a fornire una soluzione a problemi quali l'estendibilità delle applicazioni ed hanno finito col produrre sistemi monolitici difficili da gestire e configurare. Specialmente nel campo dei sistemi comunicativi, alcune aziende hanno sviluppato enormi sistemi per la gestione di rete assai potenti e ricchi di funzionalità che però non risolvono i problemi inerenti il loro sviluppo. L'ovvia conseguenza è che questi sistemi sono monolitici, difficili da gestire e configurare, necessitano di molte risorse che quindi possono essere utilizzati solo su potenti sistemi. Tali problemi hanno portato alla definizione di Yasmin, una nuova architettura per applicazioni. Yasmin risolve i problemi visti prima definendo un framework che gode delle seguenti proprietà: 1) kernel semplice e leggero, 2) basato su un nuovo tipo di componenti software chiamati droplets, 3) construito interamente con tecnologie orientate agli oggetti, 4) estendibile, facile da configurare e distribuire. L'idea dietro Yasmin è di produrre applicazioni basate su componenti software che possono essere manipolati dall'utente in modo da aggiungere o togliere tali componenti a tempo di esecuzione. Creando una ben definita interfaccia tra componenti, Yasmin è stata capace di evitare che componenti facessero assunzioni su altri componenti quindi riducendo dipendenze tra componenti in modo da facilitare il loro riutilizzo in altri contesti.

Liaison è un'applicazione basata su Yasmin ed è sviluppata per validare l'architettura. Liaison è stata applicata ad alcuni problemi inerenti la gestione di rete dimostrando chiaramente che le applicazioni basate su Yasmin hanno risolto efficientemente i problemi presentati da applicazioni sviluppate con metodi tradizionali. Inoltre Liaison ha anche fatto vedere che è possibile sviluppare applicazioni per la gestione di rete senza richiedere un lungo tempo di sviluppo e molte risorse di calcolo. Tali applicazioni sono facilmente configurabili, scalabili e dotate di buona performance. In conclusione Yasmin 1) ha permesso di risolvere molti problemi inerenti lo sviluppo di applicazioni, 2) dimostrato che i droplet possono essere usati per creare applicazioni distribuite scalabili, efficienti, facili da estendere e configurare, 3) dimostrato di essere una buona architettura per sviluppare qualunque tipo di applicazione e non solo sistemi per la gestione di rete.

Summary

Object-oriented programming (OOP) has significantly changed the way software applications are produced and has allowed many problems that affected traditional programming to be overcome. Unfortunately many object-oriented systems misused OOP techniques, failed to address issues such as application extensibility, and produced monolithic systems that are difficult to manage and tailor. Especially in the context of communication networks, network management applications are usually built following the craftsman principle (i.e. everything has to be custom-built for a certain task) without exploiting the power of OOP or adopting a clean application architecture or framework.

Some companies have developed huge management systems composed of several applications and libraries that address every network management need. Although these systems are very powerful and rich in terms of functionality and tools, they do not address problems relating to application development. In fact in order to build network management applications based on those systems, developers need detailed knowledge of many different libraries that usually have not been designed to work together and that very seldom are based on OOP concepts. In addition, due to the interdependency among those libraries, user applications require the installation of a large subset of the application system in order to run.

The natural consequence is that applications are monolithic, difficult to tailor and configure and are system resources-hungry, preventing them from running on hosts of limited power. Moreover, network management applications quite often have to support different management protocols and object models, and be open to extensions and updates as the network technology changes. In addition, applications must be built in such a way that it is possible to add new pieces when new hardware devices have to be supported or when users demand additional services.

These problems that affect many management applications, have been the driving force to define *Yasmin*, a new architecture for applications. Yasmin attempts to overcome those problems mentioned before by defining an application framework characterised by the following properties:

1. light and simple kernel;
2. based on pluggable software components;
3. built entirely on object-oriented technology;
4. extensible, easy to tailor and distribute.

The idea behind Yasmin is to build component-based applications that can

be composed by the user, who can add or replace components at runtime. By enforcing the component boundaries, Yasmin prevents components from making assumptions concerning other components, hence reducing component interdependency and making them easy to reuse in different contexts. Additionally, Yasmin loads the components on demand only when they are really needed and unloads them when no longer in use according to a policy defined by its developer. The efficient use of system resources is quite important because it enables complex applications to run on hosts of limited computation power such as mobile computers.

An effective way to limit the application size is cooperation. This is because components provide services that can be used by other components instead of reimplementing them in different flavours when needed. Every component that implements a service of general use makes it available through a well-defined interface. This is very important in the management world because large network management systems must use a common set of services that sometimes require significant resources.

Yasmin's components are called droplets. This term derives from their ability to be activated and replaced by simply being dropped into a certain directory. A droplet is characterised by the following properties:

1. it is not statically linked to the application but is loaded at runtime;
2. it has the ability to be replaced (i.e. a new version of the droplet can replace a previous one) at runtime while the application is running;
3. it has a well-defined interface, the so called droplet interface, that enables it to communicate with other droplets independently from the type of the services provided;
4. it is reentrant, hence the droplet can process concurrent requests.

The following illustration shows Yasmin's components.

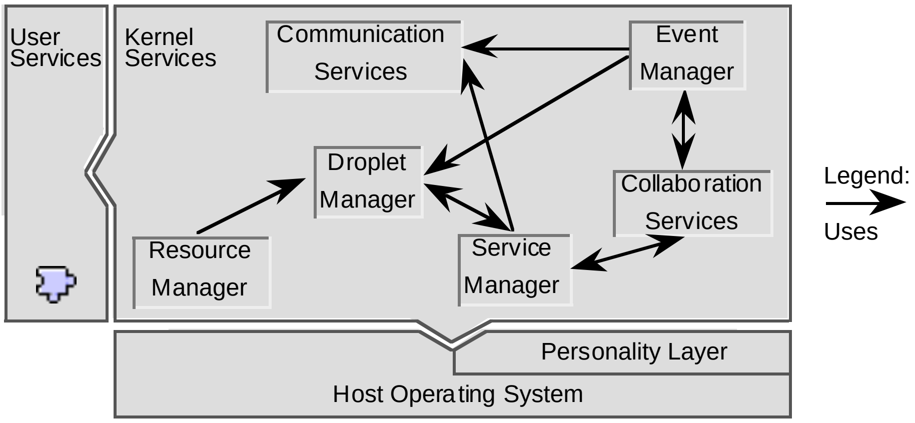


Figure 1. Yasmin's Components

User services are implemented using droplets, which change from application to application. Kernel services, part of each Yasmin-based application, provide services and functionality on which droplets rely. Kernel services sit on top of the operating system and a personality layer. This layer hides the differences among various operating systems insulating the operating system-dependent code in order to simplify the porting of Yasmin-based applications on different operating systems. The droplet manager is responsible for handling droplets and it collaborates with the service manager, informing it of newly available services. The event manager delivers events to the various components and allows different droplets and services to cooperate and interact by means of the events they exchange. The service manager interacts with the droplet manager to handle the services provided by the droplets, and it allows local or remote service calls to be issued, transparently handling the necessary communications. The resource manager cooperates with other managers to use system resources (for instance memory, threads or disk space) efficiently, making sure that system resources are not wasted and that resources no longer needed are purged. Collaboration services provide facilities for sending requests in multicast/broadcast mode to other components, collecting results, and synchronising tasks by means of events. Finally, communication services allow droplets to communicate with remote entities (local communications are performed by means of events or service requests).

Liaison, a Yasmin-based application, has been developed in order to validate the architecture. Liaison has been applied to selected network management problems such as interdomain management in order to demonstrate that Yasmin can efficiently solve these problems. Besides this, Liaison had to face with new challenges derived from new technologies such as the web and from the efforts in the network industry to unify the various management models (OSI, Internet and CORBA) in a single uniform one. Liaison has a very light kernel, that does not include any management functionality, whereas the management functionality is fully implemented inside droplets which are loaded on demand only when necessary.

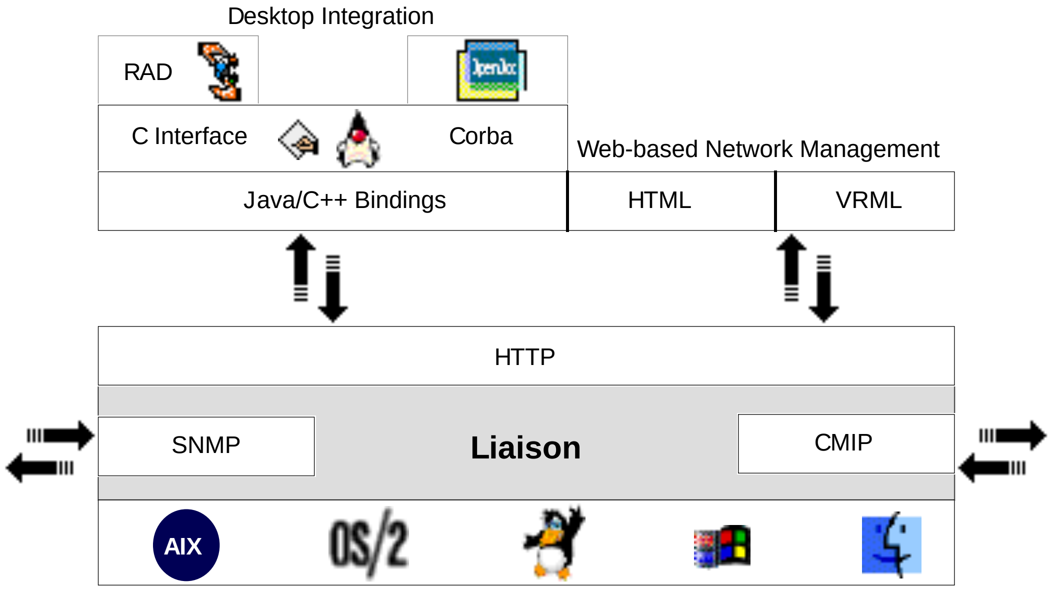


Figure 2. Liaison's Components

Liaison comes with droplets that implement:

1. web-based full OSI (CMIP) and Internet (SNMP) management;
2. a basic directory service for locating management resources and other instances of Liaison running on local or remote hosts;
3. a metadata repository used only by SNMP because the metadata information relative to CMIP is retrieved by Liaison directly from the OSI stack;
4. Java/C/C++ external bindings, which enables the development of client/server management applications to exploit Liaison's services;
5. CORBA interfaces, which allow pure CORBA applications to be created while being able to manage existing CMIP and SNMP resources.

As depicted in the previous figure, Liaison acts as a proxy (mid-level manager), which allows CMIP and SNMP resources to be managed using HTTP and CORBA. Since Liaison provides services accessible from external applications using the external bindings and the CORBA interfaces, powerful and light client/server applications exploiting Liaison's services can be created. Additionally, thanks to the large variety of interfaces offered by Liaison, it has been possible to build management applications rapidly at various levels of sophistication, overcoming a common belief in the management world that management applications require a large amount of time and highly skilled developers. Therefore, Liaison has been the very first management application that:

- allowed people to manage networks using HTML/VRML, hence enabling network resources to be managed with a Web browser instead of using complex, expensive and platform-dependent client applications (see "Web-based Management" on page 111);
- allowed people to deliver to people a real seamless integrated inter-domain management applications supporting CMIP, SNMP and CORBA, that overcomes limitations of the current generation of CORBA-based network management applications (see "CORBA Interfaces" on page 129);
- enabled people to rapidly create simple, light, and efficient management applications by combining services provided by Liaison with tools for rapid application development (see "Rapid Network Management Application Development" on page 134);
- brought software technologies such as software components into the management world solving problems that affect conventional management applications such as scalability, monolithic structure, and limited tailoring and extensibility (see "Evaluating Liaison" on page 203);
- introduced HTTP-based management that allows network resources to be managed using simple client/server applications such as Java applets which communicate with Liaison using the HTTP protocol (see "HTTP-based Management" on page 121);

- demonstrated that CMIP and CORBA management no longer have to be considered a challenge because Yasmin-based applications such as Liaison are easy to modify, use very limited computing resources, are small in size and offer good performance (see table “Liaison at a Glance” on page 141 and “Evaluating Liaison” on page 203);
- thanks to Yasmin, it has been possible to bring management capabilities to operating systems that previously were considered unsuitable for this task (for instance MacOS™).

In conclusion, Yasmin has proved to be a feasible architecture for management applications and in general for open distributed application systems. Moreover, Yasmin has exploited novel technologies such as the web and CORBA, which permitted it to go beyond the initial goals of simple CMIP/SNMP management.

1 About this Book

1.1. The Road Behind

My interest in computer networks has been gradually instilled in me by a friend who was employed as a system manager at Scuola Normale Superiore based in Pisa, Italy. Watching my friend, I was amazed to see him connect to sites throughout the world and exchange mail with distant users. In early 1991, I was looking for a master's thesis while studying at the university. I was growing increasingly interest in networks and so I decided to follow the suggestion of my advisor and contact Tecsiel, a networking company based in Pisa. During the interview, I was interrupted by the interviewer who told me more or less this: "It's amazing to see that you, like me, have inside you the 'holy fire' for computer science. Soon you will encounter many difficulties that will extinguish your holy fire". After almost six years I have to say that this man was both right and wrong. He was right because computer networks are incredibly complicated (and network management is even more complex). He was wrong because the holy fire is still burning in me. I hope it will never abate.

This thesis is my tribute to all the people who instilled in me the interest in computer science, to those who constantly encouraged me, to those who raised me and taught me that it does not matter what I do as long as I do it with passion. This book is a product of the need and the enjoyment I felt while trying to imagine it, while suffering and struggling to realise it. It is here to contain all the best I have been able to achieve during the past years: sacrifice, struggle, success, defeat, discussion, suffering, disappointments. It is my humble and tiny contribution to computer science research, made in the hope that all my efforts to develop those ideas may be useful to somebody. In any case this thesis is here: it is a software application and a book sitting on your desk.

1.2. Acknowledgments

Infinite thanks to all those who love me, and to those who loved me but who are not longer here.

Bern, 15 June 1997. Luca Deri

1.3. Structure of this Book

This thesis has been written in an incremental fashion: an argument is first introduced, and then covered in detail. This approach should make the reading easier even for the non-expert.

1.3.1. Special Font

All code listings, reserved words, and the names of data structures, classes constants, fields, parameters, methods, and functions are shown in Courier (this is `Courier`).

1.4. Types of Notes

There are two types of notes used in this book, which are formatted like the following two paragraphs.

NOTE

A note formatted like this contains information that is interesting but possibly not essential to the understanding of the main text.

IMPORTANT

A note like this contains information that is particularly important.

1.4.1. Coding Conventions

The following are conventions that apply to the code fragments contained in this book. The listings that appear in this book embody certain naming conventions designed to indicate the type and usage of identifiers. These conventions and examples of each are as follows:

classes begin with uppercase letters	`Proxy`
variables begin with lowercase letters	`proxy`

uppercase letters identify each word	`HashTable [class]`
	`hashTable [variable]`

2 Introduction

The aim of this research is to demonstrate that open distributed systems can be built with relatively little effort if an appropriate software architecture is used. In particular, by using a component-based architecture, it is possible both to satisfy the typical requirements of open systems and to produce applications that are not affected by common problems such as having a monolithic structure or being difficult to extend and to configure.

This chapter introduces the issues addressed in this thesis and defines the terms and concepts used throughout. This is necessary because merging open distributed systems concepts with other methodologies from the software engineering world requires the definition of a common terminology. Additionally the requirements and the scope of this thesis are identified.

2.1. Architectures and Framework Basics

The structure of software applications has been recognised by many people to be an important issue of concern. However, only in the past few years has software architecture emerged as an explicit and important field of study in software engineering. Although the term “software architecture” is commonly used in the computer community, there is not a universally accepted definition of it. Some people use this term to denote, a style of building applications, whereas other mean the high-level organisation of computational elements and the interactions among them. Other people may use different definitions. The worst aspect of this lack of definition is that in some cases architectural concepts are mixed with implementation concepts with the result that architectures and frameworks are sometimes mixed.

A good definition of *software architecture* is the following [Garlan93] [Abowd93]: the structure of the components of a program/system, their interrelationships, and principles governing the design and evolution over time. Components perform the primary computations of the system and interact using high-level communication abstractions such as message passing and event broadcasting. An *architectural instance* refers to the architecture of a specific system, whereas an *architectural style* defines constraints on the form and structure of a family of architectural instances. Architectural styles range from abstract patterns (such as “client-server” or "layered" organisation) to reference architectures such as the OSI (Open Systems Interconnection) basic reference model [ISO7498-4].

An architecture is defined by means of a framework, which specifies and restricts the way components interact.

NOTE
Do not confuse the term component used here with the term software component used later.

The main difference between an architecture and a framework lies in their nature. An architecture is a conceptual description of a system used to understand it at a level of abstraction at which the system's high-level design can be understood. A framework specifies:

- the structure of the components, the services they provide, and their responsibilities with respect to other entities of the system (contracts);
- the component interface, which specifies the component characteristics that have to be visible from the outside;
- the way components are glued together;
- the communication mechanisms used by the components.

Therefore a *framework* is an extensible set of cooperating components that make up a reusable design solution for a given problem domain. Therefore an architecture is an abstract system description, whereas a framework is a software library that enables the creation of components, which are guaranteed to respect the constraints imposed by their architecture. The less freedom the framework leaves the developer, the better the framework is. This is because the developer should not be responsible for design but only for application development.

2.2. Software Components

The software component field is quite new and has not yet reached a certain degree of maturity. Some problems arise from the fact that there is no unique definition of the term software component. This section is an attempt to define a software component, to establish what kind of components are currently available and to determine which of them are relevant for this thesis work.

What is a Software Component?

The term software component has different meanings depending on the community that uses it [Caldiera91]. In addition, its definition may differ even inside within one company. According to [Microsoft93a], a component is "a piece of compiled software which is offering a service", whereas in [Microsoft93b] this definition becomes "a reusable piece of software that can be plugged into other components from other vendors with relatively little effort". Although the first definition is true for components, it is too weak because, according to it, a compiled library is also a software component. The second definition is too vague and does not define any component properties.

In [Waskiewicz95] a component is defined as "the minimal piece of functionality in a system or subsystem that can be removed without affecting the integrity of the system or subsystem". This definition is somehow incorrect because according to it components have to be minimal, hence an aggregation of components is not possible. This statement thus excludes the construction of large applications.

[Ciupke96] defines components as "software units that are context independent both in the conceptual and technical domain" where the conceptual context is the modelling environment of the program being created and the technical context is given by the properties of the system such as programming language and the operating system. Although this definition emphasises another aspect of components, namely their independence from the context and hence their reusability in different contexts, it is not very clear and it does not define the component properties.

A more precise definition can be found in [Pfister96] where it is defined as "a collection of cooperative objects with a clearly defined boundary to other objects and components". This definition says more about the component properties although it defines components in terms of objects (components do not necessarily have to be object-oriented) and restricts the component size, which cannot be smaller than an object.

In [Nierstrasz95], a component is a "static abstraction with plugs". The word "static" highlights the fact that components are long-lived entities that can be stored in a software database independently of the applications that have used it. Abstract means that the component shields the software it encapsulates from the outside by putting an opaque boundary around it. The plugs are the communication channels that allow the component to interact and communicate with the outside world (messages, ports, etc.).

The definition of *component* used in this thesis is a refined version of the definition above: *a static abstraction with bidirectional plugs*. Bidirectional emphasises the fact that other components can communicate with the component but also that the component can communicate with other components, i.e. peer-to-peer vs. client-server mode. If the bidirectional constraint is relaxed, then the component is called *plug-in* because other components can communicate with it but not the other way round.

Component Granularity

Although the definition given above is quite precise, it does not specify the component granularity, which is a very important issue for the decomposition of a system into components. According to the above definition, software component granularity can range, for instance, from an iterator class to a component able to manage multimedia data such as QuickTime™ (see "Apple QuickTime Component Manager" on page 46). In order to better specify and compare different components, it is necessary to classify them into fine and coarse-grained components. The boundary which divides them concerns component properties. A component is *fine-grained* if one of the following properties hold:

- the component needs other components in order to be usable, i.e. is not self-contained (for instance an iterator component needs a list or a stack in order to be useful);
- suppose the component is coded using a compiled language, the component has to be distributed in source form in order to be used (for instance a C++ template).

A component that does not satisfy the above statements is a *coarse-grained* component. Seen the definition above, a droplet is a coarse-grained component.

Fine-Grained Components	Coarse-Grained Components	Plug-in
• STL (Standard Template Library) [Jazayeri95] • ET++ components [Gamma91] • Smalltalk Collection Classes [Cook92] • TCL (THINK Class Library) [Symantec93b] • Darwin Components [Magee92]	• OpenDoc [Apple95] • OLE [Box95] [Brockschmidt93] [Microsoft93a] • VisualBasic VBX/OCX [Microsoft92] • Java Beans [JavaBeans] • Flexible components [Leeb96] • QTCM [Apple93] • Droplets [Deri95c]	• CGI applications [CGI] • Shell Commands

Table 1. Some Fine and Coarse-Grained Components

2.3. Open Systems: What are They?

The definition of open distributed systems is not unique in the computer world. The differences among the various definitions derive from the emphasis put on the various aspects involved in a given definition of the term.

In the context of distributed systems a system is considered open if it implements open standards. Standards fall into two categories: “de facto” (a.k.a. industry standard) or “de jure” standards. In the first case a standard is such if it has been so widely adopted that virtually everyone has to deal with it (for instance the IBM PC computer, which become a de facto standard because thousands of manufacturers have chosen to clone them). A de jure standard is such if it has been standardised by some authorised standardisation body such as ISO (International Standards Organisation) or IETF (Internet Engineering Task Force). In some cases, an industry standard can be considered an open standard if:

- it has been defined by a consortium composed of a (relatively) large number of companies and institutions;

- companies and institutions that are not part of the consortium can choose to join the consortium, i.e. the consortium is not closed;
- documents produced by the consortium can be used not only by the consortium members but also by third parties;
- access to such documents is at no or nominal cost.

NOTE

As some standardisation organisations are self founded, in some cases the standards have a price or a membership is required in order to gain access to the standards (this is the case of ITU, International Telecommunications Union). This does not necessary mean that these organisations produce proprietary standards.

It is important to note that, in the definition above, the properties of the systems are not considered relevant.

In the software engineering community a system is open if it:

- does not rely on features peculiar to a specific platform, hence it can be ported to various platforms;
- has an open topology, i.e. software applications can run in a physically distributed environment;
- is open to changing requirements (evolution).

Throughout this thesis, an *open system* is such if it satisfies the following requirements:

- it implements open standards according to the previous definition;
- it does not implement or use proprietary technologies that are not publicly available or are protected by patents;
- it is proved to be portable, i.e. it is not strongly tied to a certain architecture, operating system, or manufacturer;
- it does not rely on a specific physical topology, either local or distributed;
- it is open to extensions and changing requirements.

This definition of an open system is basically the intersection of the previous definitions. In addition it does not imply that an open system has to be distributed but that it should not rely on a specific topology and hence that it is open to extensions (from local to distributed topology) and to restrictions (from distributed to local topology).

In the definition of an open system given above, the notion of system extensibility is rather vague and needs to be further defined. A system that allows functionality to be added at runtime is called an *extensible system* [Szypersky96]. An extensible system loads only the functionality currently used and adds further functionality only when needed. A more precise definition of extensibility has to take into account mutually independent extensions. A system is called *independently extensible* [Weck96] if it can cope with the late addition of extensions, possibly developed by different people in complete ignorance of each other, without requiring a global integrity check. This thesis deals with open, independently extensible distributed systems.

2.4. From Open Distributed Systems to Network Management Systems

Open distributed systems are a very broad area of computer science. This area includes very different systems ranging from e-mail systems (for instance Internet mail or OSI X.400) to distributed multimedia services like Internet MBONE.

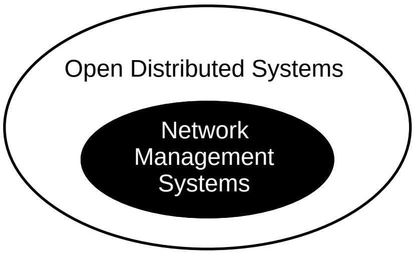


Figure 1. Open vs. Network Management Systems

Among those systems are network management systems, which are used to manage communication networks ranging from LANs (Local Area Networks) to telephone networks. As distributed open systems are a very broad area which cannot be covered properly within the scope of a PhD thesis, this work will focus on a subset there of, namely on network management. Network management has been selected because:

- this is the area of primary interest to the author;
- management systems are far from being perfect, and some problems and open issues present on this area have stimulated most of this research work;
- recent technological innovations in computer science may be profitably applied to this field, hence there is a need to define a new way to build applications that solves common problems and that exploits these innovations.

The author’s interest in network management is based on:

- the many yet unsolved problems which range from the integration of heterogeneous systems to scalability issues present in large networks;
- dissatisfaction with the usual way to approach the field, namely from the protocol side, which neglects aspects such as ease of use and elegance of design methodology in favour of raw performance;
- the absence of an architecture that focuses on the seamless integration of various protocols and services into a homogeneous and human-friendly environment.

Network management not only significantly stimulated the author’s interest but it has also been the testbed of the prototypes presented later which are based on the architecture proposed here.

2.5. Merging Network Management with Component Technology

In the past decade, the computer world has changed significantly. Powerful and centralised computers are currently being replaced with many connected mid size computers in an effort to decentralise and distribute intelligence on the network. The consequence of this trend is the need to move data reliably between computer systems running different operating systems, physically located at distant sites. The problem of managing network devices has become increasingly important with the evolution of networked operating systems and with the need to integrate management packages from different vendors [Qwerin91]. The aim of *network management* is to control, coordinate, and monitor resources for the purpose of communication. In the past, network management was relevant only for computer manufacturers, who needed tools to control the network devices they produced. Today, network management is perceived as an integrated set of tools able to manage efficiently a network made up of products from different vendors.

The need to manage different computer devices pushed the industry to produce network management tools based on open standards [Rose90]. The two predominant standards for network management, SNMP and CMIP, allow network resources to be managed independently from the vendor and the device type. Especially in the CMIP world, network management standards are quite complex. This and their relatively slow definition by standardisation bodies are two of the reasons for their delayed and limited adoption. These days, emphasis is shifting from agents, applications that physically manage network resources, to managers, applications that implement user-defined management policy by issuing requests to the agents. This is because SNMP agents are often implemented inside network devices and because modern CMIP agent development toolkits allow agents to be built in a relatively easy way. Agents run on powerful hosts usually located close to the managed resources, whereas managers often run on user machines connected to remote agents. Agents are transparent to user which interact exclusively with managers. For this reason, managers need to be resource-savvy because they have to coexist with user's applications. They should also be simple to use and to configure because modern users are accustomed to these facilities.

The accelerating pace in the development of object-oriented open network management standards, combined with the complexity and heterogeneity of CMIP and SNMP, has enhanced the interest in new object-oriented frameworks and architectures to facilitate the rapid and accurate implementation of such standards. The most interesting one seems to be OMG's (Object Management Group) CORBA (Common Object Request Broker

Architecture), which fulfils all the requirements: object-oriented, platform neutral, and able to merge CMIP and SNMP object models into a simpler, fully object-oriented CORBA object model. Unfortunately so far CORBA has not been able to deliver all this, mostly because it has been employed, in the network management field to merge network management standards into a new object model, which means that it unifies all the problems, idiosyncrasies, and limitations of CMIP and SNMP.

Most management applications are difficult to use, to configure and to extend. Worse yet they are notoriously resource hungry. One of the reason for this situation is that developers of existing network management applications have emphasised communication aspects more than application efficiency. This is because most of these developers come from the "protocol school" and do not consider such important issues as real object-oriented design and implementation, or a user-friendly graphical user interface. Instead they are accustomed to dealing with "bits and bytes" and character-based cryptic user interfaces.

In this arena the Internet recently came into the play. The explosion of Internet-accessible resources, the low cost of network equipment and the consequent proliferation of network devices have contributed to make the situation even more complex. Users demand fast and reliable networks; networks are getting bigger and even more heterogeneous; mobile computing is replacing conventional site-based processing. Therefore network management applications have to scale up in order to manage more and more devices, and hence to become more efficient and less resource-hungry. Moreover management applications have to leave their familiar environment composed of powerful hosts and expert users, becoming "visible" from the Internet as HTTP-aware distributed network resources.

2.5.1. From Class Libraries to Component Factories

The popularity of object-oriented programming (OOP) has increased considerably in the past few years. OOP offers many advantages over traditional programming, such as allowing programmers to define objects that can be easily extended and composed in order to build a software application [Brown90] [Budd91]. Although powerful, OOP lacked a standard framework through which software objects created by different vendors can interact with one another. The major result of this trend has been the production of a "sea of objects" that cannot interact across application boundaries in a meaningful way. At the beginning of this decade, the software industry realised that the ability to tie objects together to create a closer unit would result in a much more powerful system [Champeaux93]. For this reason, many frameworks and architectures have been developed to address this problem. Unfortunately applications based on such frameworks often had a monolithic structure mostly because object-oriented techniques such as inheritance have been misused by introducing cross dependencies among classes. The obvious consequence has been that objects were so tightly coupled that even the simplest application had to link the entire system [Joymer92]. This has been the reason for the decline or limited diffusion of such celebrated applications systems as Symantec Bedrock™ [Symantec93a] and Taligent CommonPoint™ [Taligent95a].

Software components [Nierstrasz92] [Joch96] [Udell94] seem to be the answer to all these problems. As its name suggests, component software is based on the notion of a component, which is a reusable piece of software that can be "plugged into" components from other vendors with relatively little effort [McIlroy69] [Staringer94].

Traditional applications are built around a block that performs all necessary functions. These applications are satisfactory and still do their job reasonably well. Nevertheless this paradigm is reaching its limits in terms of code reusability, extensibility, configuration, and especially speed and size. One reason is that such a paradigm specifies that the application must contain the entire functionality even if much of the functionality is required only for very particular tasks. This paradigm also requires that every extension of the initial design be integrated by the application developer, who is the only person able to access the source code. But the user, not the developer, is often the one who best knows the requirements. The natural consequence of this would be to provide an interface that allows the user to add new functionality to the monolithic block, and that defines a migration path towards compound applications.
Software components seem to be the answer to all these problems. Like a child does with Lego™, one can build a compound application by using several simple blocks - called components - rather than a monolithic entity. Once the application has been built, it works like a monolithic one but it has the great advantage that it can easily be modified and extended and improved by adding pieces or replacing components. Component software creates a system in which it is easy and inexpensive to tailor individual needs. The old Latin proverb "Divide et impera" can now be changed to "Build 'n play".

2.6. Thesis Motivation

It is the author's opinion that the problems and limitations of the current generation of network management systems could be solved using an appropriate architecture. This architecture should exploit the latest innovations in software engineering such as software components which seem to be a viable solution to well-known problems that affect traditional applications as well as network management systems. This is because network management places the emphasis on the communication issues rather than on architectural issues, since the customers of network management applications are large institutions or big carriers which can afford expensive computer systems. In addition, the advent of mobile computing and the Internet has highlighted the limitations of these applications systems especially with respect to scalability. For instance, using architectures employed by traditional network management systems, it is quite difficult to produce applications able to exploit distributed and heterogeneous environments and that run on computing devices of limited power (see “Do We Really Need Yet Another Architecture?” on page 68).

2.7. The Vision: Component-based Open, Independently Extensible Systems

The objective of this thesis is to define Yasmin, a new architecture for open, independently extensible systems that can be applied to selected network management problems. Yasmin is a component-based, object-oriented architecture for extensible software applications which need to accommodate extensions and updates [Rumbaugh94]. The need to define a new architecture derives from the fact that most of the ones available today are tailored for graphical user interface applications and are almost useless for network management.

Yasmin defines a new style of building applications based on established technologies such as OOP [Wirfs-Brock90], software components and novel concepts such as cooperation, delegation [Goldszmidt93], and subcontracting. This is part of the effort to make computer software easy to use and develop in addition to overcoming typical problems that affect network management applications such as scalability, monolithic structure, and limited tailoring and extensibility [Wayt94]. In addition, Yasmin allows distributed environments and Internet technologies to be exploited because Yasmin-based applications:

- are composed of droplets that can roam around and be distributed among different applications according to a certain policy (mobile agents);
- scale by replicating services in multiple locations and making them accessible transparently using a trader;
- provide services accessible from remote in a way similar to what happens with Internet services (for instance ftp);
- droplets provide services using a naming convention based on Internet naming [Deri95d].

Hence one can prove that it is feasible to develop simple yet powerful Internet-aware network management applications by using an appropriate architecture.

2.8. Thesis Scope

This thesis includes the design of a component-based architecture and the implementation of an application based on it, which can be used for open, independently extensible systems [Tsichritzis89]. As author’s interest lies mainly in network management, the architecture should take into account open issues and current trends in this field. Despite the focus on open distributed systems, the proposed architecture can be applied in other contexts when there is a need to create extensible applications that are easy to tailor and to configure, and that make efficient use of system resources. Nevertheless this work does not define an architecture for general-purpose applications, nor does it cover all the relevant aspects of managing network systems.

In order to validate this work, an application based on Yasmin has been developed and applied to the network management field. Although the application described here is fully functional and currently in use in commercial and research projects, very specific implementation issues are covered only in the appendix (see "Implementation Issues" on page 203) because they do not enhance the overall quality of this research work nor constitute integral part of it.

2.9. Thesis Requirements

In the course of the problem analysis, many requirements have been identified and divided into three categories according to their relevance. In "Thesis Validation" on page 143 the requirements listed below will be used to validate this thesis work.

Mandatory	Optional	Secondary
1. extensibility; 2. evolution; 3. ease of use and development; 4. distributed environment support; 5. use of software components; 6. promotion of reuse; 7. efficient resource utilisation and ability to run on environments of limited resources; 8. portability and genericity; 9. based on open standards; 10. Internet-awareness; 11. slim and efficient architecture; 12. full support of (management) standards; 13. scalable and performant applications; 14. independence from specific technologies and languages.	1. full distributed environment support (for instance in terms of replication and mobility); 2. security support; 3. ease of installation and tailoring 4. support for visual application development.	1. exploitation of specific platform features; 2. integration with commercial products/frameworks and ability to embed components into commercial frameworks;

Table 2. Architecture Features

Mandatory Issues

These issues are mandatory because they constitute the minimal set of requirements a modern architecture for open systems must satisfy.

1. Extensibility

   The architecture must support application extensibility. This is necessary in dynamic fields such as open systems where new standards and technologies are introduced frequently, and must be supported not only by new applications but also by existing ones. For this reason it is important to create applications that not only are able to fulfil today's requirements but can also accommodate future extensions.

2. Evolution

   Evolution is the ability to change/extend the behaviour of an application. Especially in open systems, application evolution is a must because applications have to be adapted to the changing requirements and to the different environments in which they run. For instance a management application that manages an analogic network will have to be significantly modified/extended when the network migrates to digital technology. This is because digital networks support new characteristics such as quality of service and bandwidth reservation, which were not present on analog networks.

3. Ease of use and development

   Open systems are often labelled complex systems that can be operated only by highly skilled specialists. This is because open standards are rather complex and the need for an application to support a few of them exponentially increases the system's complexity. Ease of use concerns issues such as installation, configuration, and tailoring. Ease of development concerns the development language, the various libraries and APIs which are used to build up the application, and their integration. In fact many problems are derived from the loose integration of the various object models and APIs, which exposes to the developer all their idiosyncrasies.

4. Distributed environment support

   As this thesis deals with open distributed systems, it is necessary to exploit distributed environments. This ranges from basic communications between remote peers to the exploitation of distributed environment, for instance, increasing application performance and robustness.

5. Use of software components

   Component technology appears to be a very promising way to overcome many problems that affect most software applications. This is especially true in some fields of open systems such as network management, where applications are often monolithic and difficult to extend and configure. In addition software components have many other advantages in terms of code reuse and application design, which can become necessary to create a new generation of modern management applications.

6. Promotion of reuse

   Reuse of code and design are becoming increasingly important in the software industry. Software components are a way of achieving code reuse, whereas a good architecture guarantees design reuse. Reuse is an important issue because it saves development time. Moreover, reusing well designed and already tested pieces significantly reduces testing time, hence minimising the "time to market".
   In the open systems field, reuse of existing (legacy) code is very important because it is definitely not feasible to implement a new system from scratch. Key part of it may take a significant amount of development time. In addition, it is quite often mandatory to integrate legacy binary code, which is necessary to interface the system with existing applications and resources.

7. Efficient resource utilisation and ability to run on environments of limited resources

   It is a common belief that, because open systems are complex, they need a significant amount of resources in order to run. For instance, the best selling system/network management platforms (see "Classic Management Platforms" on page 63) require at least a mid range Unix™-based computer equipped with no less than 64-128 Mb of memory and a large hard disk in order to install and run the daemons and applications that make up the management environment. This example shows that, although management applications are complex, an architectural problem has led to the creation of such monstrous applications. A modern architecture for open systems must be able to produce applications that do not demand a great amount of resources and that can run, within certain limitations, in environments of limited computing power. This has become even more important since the advent of mobile computing because many people have replaced their desktop computers with mobile ones with limited computing resources which must be used very wisely.

8. Portability and genericity

   Code/design portability is a must in the field of open systems because the same application may have to run on very different operating systems. This imposes constraints on the architecture, which must be generic enough to be adaptable to different platforms but still able to exploit most of the facilities offered by a platform such as threads or multiprocessing.

9. Based on open standards

   Although open systems must be based on open standards, often proprietary protocols are often used to implement secondary functionality instead of using existing standard protocols. For instance only a few large SQL databases can be managed using a standard management protocol such as SNMP, and most of them are managed with proprietary tools. This design decision prevents such databases from being managed like any other system/network resource and also introduces additional costs. For this reason, an architecture for open systems must define new protocols if and only if there is not a suitable open protocol. In any case a proprietary protocol (i.e. an undocumented black-box protocol) must be avoided.

10. Internet-awareness

    Internet protocols are becoming the default way to access information. Therefore it is necessary to leave room for these protocols in the architecture, hence to support at least one of them (for instance HTTP).

11. Slim and efficient architecture

    As open systems have to implement various protocols, it is mandatory that the architecture be kept slim and efficient in order to avoid adding further complexity. In many cases the architecture has merely to provide the glue between the application and the protocols it implements.

12. Full support for (management) standards

    When a standard protocol is implemented, it has to be implemented in full, not just in part. In general the architecture must fully support a given standard although some functionality may not yet be implemented. This is an important point because quite often architectures have been designed to support a limited standard subset due to architectural limitations that preclude full support. This is significant for management standards where vendors have to specify in a PICS (Protocol Implementation Conformance Statement) which standards have been implemented and to what extent they comply with the standard specification.

13. Scalable and performant applications

    The architecture must support scalability because in open systems it is quite difficult value a given problem size. In the management world for instance, network devices and services increase rapidly every year, making it necessary to produce management applications that can be scaled up according to current needs.
    Furthermore final applications must perform quite well in order to be used not only for everyday but also for real mission-critical tasks. This is necessary in order to validate the architecture and to show that it can be used for real application development.

14. Independence from specific technologies and languages

    The concepts in this thesis have to be general enough to be implemented using different programming languages. This means that this work should not depend at all on a specific technology or programming language. In particular this applies to Yasmin components, which must be implementable on different platforms using different languages without relying on specific features such as garbage collector or multithreading.

Optional Issues

1. Full distributed environment support

   The full exploitation of a distributed environment, for instance in terms of mobility and replication, is an interesting problem. The effort necessary to cover it is, however, huge due to the wide variety of the mobile systems/applications and to the challenges associated with a complex topology. For this reason, the architecture must not impose additional constraints on the topology nor make assumptions concerning the location of certain services/resources. This will facilitate extending the architecture, should this become mandatory.

2. Security support

   Security in the context of this thesis means identifying and implementing mechanisms that allow the distributed environment to be exploited in a safe way by preventing possible intruders from interfering with the application. Security should also prevent unauthorised users from changing critical network parameters, which can affect the global system stability.

3. Ease of installation and tailoring

   An application that is difficult to install and tailor has not fulfilled the requirements, even if such an application will be used only by experts. It would be useful to provide facilities that make installation and tailoring easy and allow it to be done not only by experts but also by novice users.

4. Support for visual application development

   The use of software components has stimulated the creation of applications that can be built visually using a predefined set of basic components. It would be interesting to understand what kind of support has to be provided by an architecture in order to tackle this issue.

Secondary Issues

1. Exploitation of specific platform features

   The exploitation of specific platform features allows applications based on the architecture to perform and behave exactly as other applications tailored for the platform. As the main objective of this work is to be general, however being both general and able to exploit platform-specific features is contradictory. Although this requirement may be relevant for commercial products, it is worth noting that this thesis deals with open systems, for which specific platform issues are irrelevant.

2. Integration with commercial products/frameworks and ability to embed components into commercial frameworks

   This issue is interesting from both the commercial and the technical point of view. As the author is not aware of component-based architectures for open systems, the integration of this work with other commercial applications can be done only at a communication level. As one of the requirements of this work is the use of open standards, the only issues involved are related to interoperability, i.e. to validating the implementation of open protocols.

2.10.Research Goals

Besides fulfilling the above requirements, the goals of this research are as follows:

1. The demonstration that Yasmin overcomes typical problems that affect open systems, such as being monolithic and not open to extensions (see "Conceiving Yasmin" on page 73).
2. Yasmin's components, called droplets, are effective for developing applications:
   - whose behaviour can be extended and modified at runtime by adding/replacing droplets (see "Droplets" on page 80);
   - that integrate both legacy and heterogeneous code, a typical situation in the field of open distributed systems, where different object models need to be integrated in the same application (see "Heterogeneous Code Integration" on page 87);
   - whose components can migrate when necessary. Suppose there is a remote application that implements a heavily used service inside a droplet. Thanks to the droplet paradigm, it is possible to migrate such a droplet and integrate it into the local application, which will then avoid making remote calls (see "Component Migration" on page 102).
3. The proof that cooperation and delegation, when combined with a clean component interface, promote reuse and resource sharing because this will prevent the duplications of services and functionalities (see "Reuse" on page 78). Thus when a resource/service is of general interest, it must be exported by the component through the droplet interface. This allows the resource/service to be usable from other components and hence prevents them from reimplementing it.

Specifically with respect to network management, the goals of Yasmin-based applications are the following:

1. The implementation of selected network management applications, currently considered a difficult task and described in "Introduction" on page 105, no longer is a challenge when Yasmin is used (see "Rapid Network Management Application Development" on page 134).
2. The demonstration that network management applications can profitably exploit Internet technologies and then be visible using conventional Internet tools such as a web browser (see "Web-based Management" on page 111). This is a very important demonstration in the network management world because it proves that management applications can be integrated and accessed using common tools rather than having to run special applications on special hosts to perform management.

IMPORTANT

Management applications can usually be used only on hosts where the management platform is installed and thus cannot fully exploit client/server environments. Internet visibility is also important because it allows management applications to exploit at no cost facilities such as security (for instance SSL and firewalls) and platform independence (for instance through the use of HTML and Java as described in “Web-based Management” on page 111 and “HTTP-based Management” on page 121). The use of Internet technologies therefore allows management applications to leave their highly specialised environment in favour of a more standard, simple, and platform-independent one.

3. Rapid development of distributed client-server applications that communicate using standard protocols (see “HTTP-based Management” on page 121 and “Rapid Network Management Application Development” on page 134). This proves that contrary to a common belief, management application do not necessarily need a long development time.
4. Identification of a new management approach that harmonises the various network management object models, a.k.a. interdomain management (see “CORBA Interfaces” on page 129), and overcomes the limitations of current solutions (see “Interdomain Management” on page 66).

NOTE

With the advent of object-oriented distributed computing models (for instance CORBA), there are efforts in the industry and research to manage networks using a single and uniform object model although the managed devices may support various object models. The goal of interdomain management is to create a bridge between emerging models such as CORBA and existing management protocols such as CMIP and SNMP. Future management applications will be based exclusively on these emerging models (see “Interdomain Management” on page 66).

5. Reuse and sharing of common services (see “Reuse” on page 78) that prevent their replication whenever a new application has to access to them.

Finally, the proposed architecture and the application built using the concepts defined here, are compared with similar state-of-the-art approaches (see “Comparison with Other Architectures” on page 101) in order to:

1. show how this work solved the problems that affect the current generation of network management applications (see “Final Remarks” on page 141);
2. propose solutions to partially solved network management problems such as interdomain management (see “CORBA Interfaces” on page 129);
3. demonstrate that Internet standards/tools can be profitably used for the purpose of network management (see “Web-based Management” on page 111 and “HTTP-based Management” on page 121) without the need to design additional standards/tools uniquely for the purpose of management.

2.11.Research Contributions

The main contributions of this thesis are the definition of a new type of software components and the novel adoption (and integration) of well-known concepts/techniques such as software components, cooperation, and delegation into a new field, namely open distributed systems. In particular this research work:

1. defines droplets, a new type of software components that are replaceable and modifiable at runtime (see "Droplets" on page 80), hence which allow application behaviour to be extended and modified at runtime as the application requirements change;
2. presents Yasmin, an original architecture for distributed open, independently extensible distributed systems that:
   - effectively exploits object-oriented and software component techniques (see "Yasmin at a Glance" on page 75);
   - addresses well-known problems in the field of software engineering such as monolithic application structure and runtime application evolution (see "Yasmin's Design Choices" on page 98);
   - enabled the creation of a new generation of management applications accessible using Internet technologies.
3. demonstrates the superiority of software components over the techniques currently being used for open system development (see "Final Remarks" on page 141) which can lead to large monolithic applications that are difficult to modify and extend;
4. proves that, contrary to common belief (see "Introduction" on page 105), efficient network management applications can be built rapidly and simply (see "Rapid Network Management Application Development" on page 134) without having to use specialised tools but just reusing existing tools and techniques;
5. introduces a new implementation technique used by Liaison that demonstrates that it is possible to:
   - overcome limitations of the current generation of management applications such as being large and monolithic, not portable across different platforms, resource hungry, and difficult to extend and modify (see table "Java/C++ Bindings vs. Similar Solutions" on page 128 and table "Comparison of Techniques for Rapid Application Development" on page 137);
   - achieve interdomain management, Internet protocol support, and true platform independence (see "Welcome to Liaison" on page 109) without having to give up ease of development and application performance (see "Evaluating Liaison" on page 203);

6. defines a new technique for real seamless interdomain network management, an open problem in the management community, and demonstrates its practical feasibility (see "CORBA Interfaces" on page 129);
7. defines an Internet Draft which explains how management applications can benefit from using Internet technologies [Deri96c].

Although implementation issues should not belong in a PhD thesis, it is worth mentioning that Liaison is a true milestone in the field of HTTP-based network management because it has been the very first application that:

1. allowed network resources to be managed with a Web browser [Deri95a] instead of using complex, expensive and platform-dependent client applications (see "Web-based Management" on page 111);
2. employed VRML to represent network management information in 3D [Deri97b] going beyond classic 2D visualisation techniques that are not suitable for modern network topology (see "VRML: Adding 3D to Network Management" on page 117);
3. allowed CMIP and SNMP network resources to be managed using the Java language [Deri96b] enabling the creation of simple, cheap, platform-independent management applications (see "HTTP-based Management" on page 121);
4. implemented full CMIP and SNMP network management capabilities using CORBA, demonstrating that interdomain management is feasible [Deri97a] by overcoming limitations of the current generation of CORBA-based network management applications (see "CORBA Interfaces" on page 129);
5. enabled development of slim, client-server network management applications in a visual and rapid way [Deri97c] by combining the services provided by Liaison (see "Rapid Network Management Application Development" on page 134) with tools for rapid application development;
6. implemented network management in a real portable way, showing that it is possible to achieve performance, conformance to standards, and portability (see table "Liaison at a Glance" on page 141 and "Evaluating Liaison" on page 203).

2.12.Thesis Outline

This thesis is divided into four parts:

1. collection of requirements, open issues, and limitations of current generation of network management applications;
2. definition of a component-based architecture that satisfy the requirements;

3. design and implementation of a modern network management application based on the defined architecture;
4. validation of the work and comparison with relevant architectures and network management systems.

This chapter has covered the basic concepts and terminology necessary to understand and evaluate this thesis, and defined the scope and the goal of this work in addition to having established external constraints and requirements. The main achievements and contributions of this research have been also identified.

Chapter three covers relevant research efforts undertaken in the areas covered by this thesis: software components and network management architectures. For simplicity, the chapter has been divided into two parts. In the first part, state-of-the-art component-based architectures and software components are covered in detail. The second part instead, covers the three standard management architectures, highlighting their differences and similarities (see “Comparison of Network Management Architectures” on page 60). In addition to commercial and research applications, this comparison is necessary because the three management paradigms are quite different in terms of goals and functionality.

Chapter four covers Yasmin in detail. It describes the concept behind Yasmin, why it has been necessary to define yet a new architecture for open systems, which issues Yasmin addresses that are not covered by the available architectures. Additionally, Yasmin’s main components are covered in detail and Yasmin is compared with the component-based architectures shown in Chapter three (see “Comparison with Other Architectures” on page 101).

Liaison, a Yasmin-based application for network management, is covered in Chapter five. First, the scenario is shown in which Liaison was conceived, including the open problems that Liaison solved. Then, Liaison components and internals are described according to the functionality they implement and compared with similar approaches undertaken in this area. Finally, it is shown how Liaison can be used profitably to build interdomain management applications rapidly.

Chapter six evaluates and validates this work comparing it with relevant efforts undertaken in the areas of component-based architectures and open systems.

Chapter seven summarises the lessons learned in the course of this work, and demonstrates that Yasmin and Liaison have fulfilled the initial goals. Finally, open issues requiring future work are identified.

3 Related Research

This chapter covers the relevant standard, commercial, and research efforts undertaken in the area of software components and network management. In the next two chapters, these efforts described here are compared with the work in this thesis.

3.1. Component-based Architectures

In this section, five component-based architectures are presented. These architectures differ considerably: the first one is used for document-centric application, the second one for generic software applications, the third one for compound Java applications, the fourth one for simple applications, and the last one for sharing multimedia components. The first four architectures have been selected because of their acceptance in the industry, whereas the last one is relevant for this thesis because the components it defines are similar in some aspects to the droplets.

Because droplets are coarse-grained components, the following sections will only cover-coarse grained components, in order to better position droplets.

3.1.1. OpenDoc

OpenDoc [Apple95] is a set of shared libraries designed to facilitate the construction of compound, customisable, collaborative, and cross-platform documents. To do this, OpenDoc replaces today's application-centred user model with a document-centred one. The user focuses on constructing a document or performing an individual task, rather than using a particular application. The software that manipulates a document is hidden, and users feel that they are manipulating parts of the document without having to launch or switch applications. This document-centred model does not mean that OpenDoc supports only those kinds of data found in paper documents. An OpenDoc document can contain data as diverse as navigable movies, sounds, animation, and database information such as networked calendars or virtual folders as well as traditional spreadsheets, graphics, and text. In OpenDoc, every kind of medium can be represented as a part of any document. Thus, an OpenDoc document is automatically able to contain future kinds of media, even kinds not yet envisioned, without modification. Although OpenDoc lends itself readily to complex and sophisticated layouts, its usefulness is by no means restricted to page-layout applications or even compound documents. The scripting and extension mechanisms allow communication among parts of a document for any imaginable purpose.

OpenDoc consists of six functional service layers:

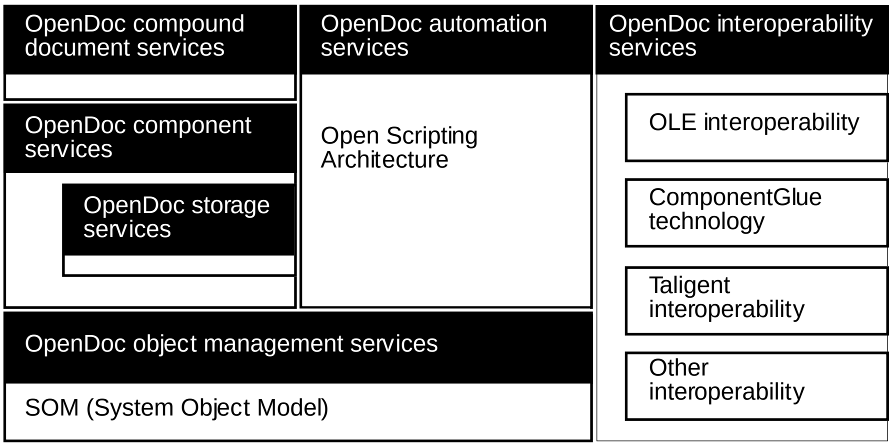

OpenDoc compound document services
OpenDoc automation services
OpenDoc interoperability services
OpenDoc component services
Open Scripting Architecture
OLE interoperability
OpenDoc storage services
ComponentGlue technology
Taligent interoperability
OpenDoc object management services
Other interoperability
SOM (System Object Model)

Figure 2. OpenDoc Building Blocks

1. OpenDoc Compound Document Services

   OpenDoc Compound Document Services manage display and user-interface aspects in such a way as to ensure a unified document model that supports multiple data types, while providing users with a smooth and single look and feel when working with a variety of contents. These services consist of a set of libraries that allow editors to work together to display and manipulate the contents of an OpenDoc document.

2. OpenDoc Component Services

   OpenDoc Component Services support the integration of multiple software components into both seamless compound documents and custom applications. They allow cross-platform support, scriptability, replaceability, and extensibility. These services consist of a set of libraries designed to allow components to work together by providing methods for negotiating resources, registering objects for cooperative use, and persistently storing components.

3. OpenDoc Storage Services

   OpenDoc Storage Services are designed to solve the problems inherent in storing multiple content elements by providing a standard mechanism for storing such elements as objects. These services are based on Standard Interchange Format (formerly known as Bento), a published and widespread model for document storage. OpenDoc Storage Services consist of a portable object storage library and format that allow OpenDoc to store and exchange compound documents and multimedia.

4. OpenDoc Automation Services

   Based on the Open Scripting Architecture (OSA), OpenDoc Automation Services (OAS) consist of an automation and scripting API that supports application-independent scripting. Not a scripting language, but rather a standard for the coexistence of multiple scripting systems, OAS provide full functionality with compliant scripting languages, allowing them to plug in transparently. This lets arbitrary scripting systems be built as shared libraries, so that any component application can be scripted by any scripting system. In this way, OAS free developers from the need to develop a scripting language or system of their own.

5. OpenDoc Object Management Services

   OpenDoc Object Management Services are based on IBM's System Object Model (SOM) [DSOM] [IBM94a], a highly efficient dynamic linking mechanism for objects, which supports multiple languages and provides a gateway to distributed object services. SOM technology allows different object-oriented programming languages, such as SmallTalk [Goldberg83] and C++ [Stroustrup91] [Ellis90], to create objects that can work together on a single desktop. Through SOM, developers can take advantage of the OMG's CORBA standard for distributed object messaging (see “OMG Network Management” on page 57). This provides a path to distributed cross-platform, networked applications and a means to link desktop applications to information services, computing resources.

6. OpenDoc Interoperability Services

   OpenDoc Interoperability Services solve the issue of interoperability between OpenDoc and other technologies such as Microsoft OLE, Microsoft Corporation's proprietary API for application integration (see “Microsoft Object Linking and Embedding” on page 38). The ComponentGlue technology lets any OpenDoc component be directly embedded within any OLE object or application. OpenDoc components appear as OLE objects which can be embedded in OpenDoc documents. This capability allows OLE-enabled desktop objects to communicate easily with distributed OpenDoc components. An OLE object embedded in an OpenDoc document remains an OLE object and it does not inherit such additional OpenDoc features as support of irregular objects. But, within the limitations of the functionality that OLE provides, the OLE object works inside the OpenDoc application.

Component Model

OpenDoc's basic elements are documents, their parts, their frames, and the part editor code that manipulates them. Those elements, represented in a set of object-oriented class libraries, define a number of object classes. The classes provide interoperability protocols that allow independently developed software components to cooperate in producing a single document for the end user. Through the class libraries, these cooperating components share user-interface resources, negotiate document layout on-screen and on printing devices, share storage containers, and create data links to one another.

Documents, not applications, are at the centre of OpenDoc and individual documents are not tied to individual applications. In creating OpenDoc documents, collections of software components called *part editors* replace the conventional monolithic applications in common use today. Each part editor is responsible only for manipulating data of one or more specific kinds in a document. The user does not directly launch or execute part editors, however. The user works with document parts (or just parts), the pieces of a document that, when executing, include both the document data and the part-editor code that manipulates it. Each part editor must: display its part, both on screen and when printing, edit its part by changing the state of the part in response to events caused by user actions, and store its part, both persistently and at runtime. At runtime, a part is the equivalent of an object-oriented programmatic object in that it encapsulates both state and behaviour. The part data provides the state information, and the part editor provides the behaviour; when bound together, they form an editable object. As with any programmatic object, only the state is stored when the object is stored. Moreover, multiple instantiations of an object do not mean multiple copies of the editor code; one part editor in memory serves as the code portion for any number of separate parts that it edits. OpenDoc dynamically links part editors to their parts at runtime, choosing an editor based on the kinds of data that a part contains. Dynamic linking is necessary for a smooth user experience because any sort of part might appear in any document at any time.

All OpenDoc part editors are represented to OpenDoc by a subclass of *ODPart*, which is a SOM class as are all the classes in the OpenDoc class library. The part editor interface is written using an extended version of OMG's Interface Definition Language (IDL) [OMG95]. Using SOM it is possible to create subclasses of existing part editors simply having access to the IDL file, which defines of the part editor interface (no access to the part editor code is necessary). Unlike the object models found in formal object-oriented programming languages [Rumbaugh91], SOM is language-neutral. It preserves the key OOP characteristics of encapsulation, inheritance, and polymorphism without requiring that the user of a SOM class and the implementor of a SOM class use the same programming language. Therefore, it is possible to write part editors using one of the languages supported by SOM (currently C, C++, FORTRAN and SmallTalk). This also implies for instance that a base part can be written in C++, an inherited part can be written in Pascal, and a double-inherited part can be written in C++ or Pascal or C or Fortran etc. OpenDoc components can be parameterised in a totally different way that common constructs such as the C++ template. Each component has an associated part kind, i.e. a typing scheme analogous to file type, to determine which part editor to associate with a given part in a document. Because OpenDoc documents are not associated with any single application, a file type is insufficient in this case; each part within a document needs its own "type", or in this case, part kind. Common part kinds are "TEXT", "PICT" and "SND". The substitution of part editors is facilitated by defining a part category, a general description of the kind of data manipulated by a part editor. Part categories have broad designations, such as "plain text", "styled text", "bitmap", or "database". Both part kind and part category are specified as ISO strings.

OpenDoc components are black-boxes and because they are implemented using SOM, they behave like SOM objects derived from a class named `ODPart`. Developers who need to specialise a component can override some of its methods and extend the interface by subclassing the component. Thanks to SOM, there is no need to have access to the component's source code but simply to the component interface defined in IDL. If the interface has not been made available by the component developer, the component can be treated like a subclass of `ODPart` and then subclassed. OpenDoc components are first-class values.

Component semantics is defined by OpenDoc. Hence every part has a well-defined behaviour (at least at the `ODpart` level) and an interface which guarantees plug-compatibility and versioning through facilities offered by SOM. Composition can be obtained in two ways:

1. Runtime

   Objects can be composed using OSA or by creating a document template (c.f. below). In both cases composition is dynamic, i.e., it is possible to modify the relation ship among components and the way they are glued together. A *document template* is a file that contains information on how the various components have been glued together in order to define the document. For instance, a word processor document template specifies how the various components (text editor, spelling checker, drawing tools) have been glued together to build the word processor. A document generated out of a document template is an instance of the template and it stores only pure data (e.g. text and pictures). A document template can be further composed by adding components or by changing the links among them. In any case a document template it is not yet a new `SOMObject`, hence it does not own an interface and cannot be subclassed.

2. Compile time

   Components can be composed by defining a new component. In order to do this a new IDL file has to be defined and the component has to be implemented using a programming language. In this case the composition is static because the result of the composition is a new component whose behaviour can be modified only through recompilation.

Application Domain

OpenDoc has been tuned to document-centric applications, but this does not mean that OpenDoc can be used exclusively for document-centric applications. As SOM is an integral part of OpenDoc, it is possible to build CORBA-savvy applications that are not based on documents, such as servers or daemons.

Users can build applications on top of the basic classes, on top of ODF (OpenDoc Framework provided with OpenDoc for MacOS) or they can define a new framework. In any case, developers have limited freedom because the architecture specifies how components have to interact and how they can be composed. This is certainly an advantage because it guarantees interoperability and component composition. Heterogeneous components are supported thanks to SOM, which also supports versioning and provides functions to query metadata information contained inside components. This does not guarantee that there is a binary representation for components but simply that components built and running on different platforms will be able to interoperate and be composed into a document as soon as an upcoming version of OpenDoc based on DSOM (Distributed-SOM) is released. OpenDoc specifies how the messages exchanged among components should look, leaving to (D)SOM the task to mask differences between different operating systems and architectures. Finally CI (Component Integration) Labs, a non-profit organisation, ensures that OpenDoc components follow the OpenDoc guidelines. Hence it certifies that components are able to interoperate and be composed.

Concluding Remarks

OpenDoc is the first (and so far the only) architecture that allows compound document-centric applications to be built. Based on SOM, it allows components to be created and composed at both compile and runtime. The use of IDL-defined object interfaces simplifies the creation of black-box components and clearly separates component's interfaces from their implementation. Although OpenDoc has been tuned to document-based applications, it is possible to use some of its components such as Bento and SOM to build more general-purpose applications which, unfortunately, cannot take full advantage of OpenDoc.

At the moment, OpenDoc has not yet reached maturity owing to its limited availability (MacOS and OS/2, and recently Win32 and AIX) and to its lack of many applications and parts. In addition, development tools are still primitives and do not offer facilities to easily debug compound applications. These problems are expected to be solved very soon, and a recent agreement between Apple and Sun will shortly allow people to use Java to write OpenDoc components, and hence to combine the flexibility of Java with the power of OpenDoc.

3.1.2. Microsoft Object Linking and Embedding

Microsoft's Component Object Model (COM) is a language-neutral binary interface specification for Windows Objects and a set of runtime functions for instantiating them. Microsoft's Object Linking and Embedding (OLE) [Microsoft93a] [Brockschmidt93] is based on COM. It defines a set of COM object interfaces, which provide a variety of ways to integrate application components, and specifies their binary storage representation.

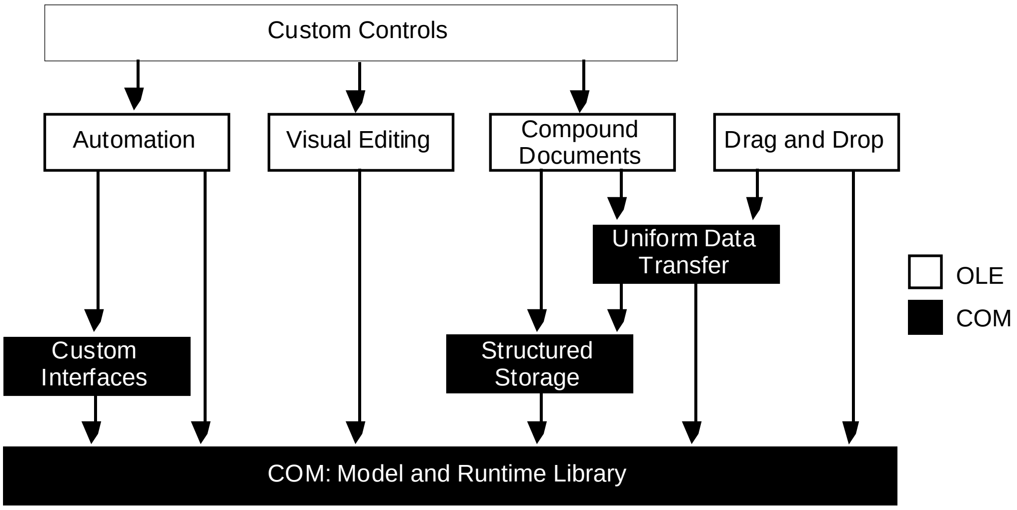


Table 3. COM/OLE Layering Model

In particular OLE defines:

1. Binary Object Representation

   As COM is a specification and does not provide an object representation, OLE has to specify the binary representation of OLE components [Box95] [Petzold95]. Unlike other object models which are language-independent, OLE embraces more or less closely the C++ way of thinking objects which is used to provide language independence. Interfaces make use of structures of pointers and function calls be means of pointers stored in a pointer table built by the developer. Once an OLE object has registered its interface with the OLE object registry, applications can query the object interface and use the provided interface using a C++ call style.

2. Collection of Interfaces

   OLE provides visual editing, drag and drop between applications, OLE Automation, and structured storage for objects. Visual editing allows two or more applications to cooperate in the editing and display of compound documents. Drag and drop allows users to select an application object such as a document and drop it into another application window, where it can be copied or moved. *OLE Automation* provides a standard means for macro and script languages for applications to view and manipulate a set of internal application-level objects and for altering the object's state. Finally, structured storage allows applications to cooperate in the creation of compound files supporting a variety of native data types stored as nested objects within a standard file format.

3. System Object Model

   OLE's system object model based on COM provides object versioning and evolution in a safe way. It offers interfaces that allow developers to query the object interface, safely downcast objects and aggregate them.

4. Distributed Capabilities

   The current version of OLE permits components running on different object spaces inside the same host to be integrated. The coming version of OLE for WindowsNT will include a networked version of OLE that allows an application to integrate OLE components located on a host connected via a network. Networked OLE will be based on LRPC (Light RPC) a lightweight version of the standard Unix RPC and will include support for security designed to be upwardly compatible with future implementations of OLE.

Component Model

COM is, according to Microsoft, a language-neutral binary interface specification but is primarily a set of rules that must be implemented in every application implementing interoperable *Windows Objects*. It provides no formal representation of an object, hence developers are responsible for providing the representation of Windows Objects. They are arbitrary as long as one obeys the rules of how COM expects its objects to behave. COM is object-based and not object-oriented and is based on aggregation rather than inheritance. Therefore, applications using COM obtain and manipulate only interface references and not objects at all. A byproduct is that it is possible to manipulate COM objects from relatively simple and non-object-oriented languages such as Basic.

A component interface is a strongly typed contract between software components that is designed to provide a small but useful set of semantically related operations. An interface is associated with a unique identifier called an *interface ID*. The way to obtain an interface from an object is to query it directly from the object using a method called *QueryInterface*. When an object is to be cast, a new interface has to be obtained. If the interface cannot be returned then the object does not support that interface. QueryInterface is a safe way to cast objects; it eliminates the error-prone situations that arise in many programming languages like C++.

COM does not support inheritance but *aggregation*, which is essentially a manual technique entirely implemented by user-written code. Rather than have one class derived from another, COM allows a new aggregate object to be built. Using this model, a set of objects can work together in a well-defined manner to appear to other software components as a single object: the container object delegates to the component object in much the same way as a derived class passes messages to its base class. As the COM interface is very generic, it is possible to support genericity, i.e., to parameterise COM objects. However COM provides no support, so developers are responsible for implementing it themselves.

Components are black-box entities. The component interface is the only way to interact with the component and can also be used to query information about other supported interfaces. Developers have no access to component internals nor to additional information. Unlike real objects, COM objects can externalise information only by means of their interface. An advantage of this scheme is that aggregate components cannot rely on side information but only on the interface. Hence COM prevents the “fragile base-class problem", which is well documented in the literature [Nackman94] [Szypersky96].

A component can be of arbitrary granularity, so it can be smaller or larger than an object. Applications usually consist of a main program and additional code which glues components together. However it is possible to have OLE servers, implemented as DLLs (Dynamic Loadable Library) and without a visual appearance, which can be embedded in an existing document or application. OLE allows components to be embedded at runtime into existing applications. Hence they can be considered first-class values.

Composition can be performed at:

1. Runtime

   Objects can be composed dynamically using OLE Automation, which is essentially a set of interfaces that allow some object characteristics to be exposed to other components. This serves to connect components to applications. Languages such as Microsoft VisualBasic™ [Microsoft92] or Borland Delphi™ [Pacheco96] thus allow binary stand-alone application to be created through composition. Notice that components will not be integrated as part of the application code but will still be delivered as DLLs.

2. Compile time

   At compile time it is possible to compose objects using aggregation. In this case the composition is static because the result of the composition is a new component whose behaviour can be modified only through recompilation.

Component semantics is partially defined by OLE, which specifies a minimal behaviour of a component. Nothing can be said about other component properties because OLE guarantees very minimal properties such as plug-compatibility via an object interface.

Application Domain

OLE components come in different flavours including OCX (OLE Control eXtension) and OLE servers. OCX have a visual appearance and are suitable for document-centric applications. OLE servers may have a visual appearance (only if they have been embedded in an application) and are usually used to provide services. Although OLE pretends to be suitable for document-centric applications, it lacks important features such as document storage or real support for scripting. OLE is suitable for building applications that make use of components (for instance OCXs) where the interaction among components is rather limited.

OLE specifies how components should look from the outside. Developers can use the standard Microsoft framework or build their own framework to develop the component. In both cases developers have limited freedom because OLE specifies how components have to interact and how they can be composed. Although OLE specifies several characteristics of components, many Windows developers have experienced problems with third-party components. Unlike OpenDoc, in the Microsoft world there is no organisation which tests components for conformance with OLE. Hence embedding components can sometimes become quite an ordeal.

Components are supposed to be language-neutral but the use of pointers and function pointers make them easy to build only with C++-like languages. COM does not define a definition language such as OMG's IDL, so there is no standard language-independent way to describe component interfaces. The lack of a definition language is fairly important from the programming point of view because COM does not provide an automatic tool, such as an IDL compiler, that facilitates the composition and the definition of new components. Heterogeneous components are supported but have to be developed with languages that support C++ pointer arithmetic.

COM provides facilities to implement versioning and to query object interfaces. *Windows OLE Registry* keeps track of the registered components and provides facilities to register/deregister them. As OLE is very complex and error-prone, Microsoft has released code generators and other tools which simplify the development of components. Unfortunately most of those tools are not part of the standard OLE distributions but usually come with Microsoft development tools.

Concluding Remarks

Although OLE has many limitations, is based on proprietary technology, is difficult to program and not object-oriented, it is a reality. Hundreds of companies have embraced this technology and offer components for many different purposes. It is likely that a component one needs is available, so it can simply be reused. Aside from general acceptance, the OLE world is overly complex. Old and widespread components called VBX (VisualBasic eXtensions) [Microsoft92] are going to be replaced with OCX, OLE does not specify all the implementation details, so component reuse implies that components come with low-level implementation details. Although it is clear today that OpenDoc/SOM is technically far better than OLE, the market has not yet accepted this new technology. Hence it is likely that OLE will continue to be an important player in the near future.

3.1.3. Java Beans

A *Java Bean* [JavaBeans] is a reusable software component written in the Java programming language [Sun96a] such that it can be manipulated visually in a builder tool. Builder tools may include web page builders, visual application builders or server application builders. The goal of JavaBeans is to define a software component model for Java in order to enable the composition of Java software components by end users. Beans range from small controls to simple compound documents. Thus the beans’ API (called an interface, i.e. a collection of method signatures, in Java terminology) is similar to the OLE Control API but does not provide the full range of high-end document API provided, for instance, by OpenDoc.

A bean is a collection of Java classes describing the behaviour of the objects that make up the bean and a serialised (i.e. marshalled) version of the initial object configuration of the bean. Therefore a bean is not a class that has to be instantiated in order to obtain a runtime component, and not even an object but rather a collection of objects, each having an initial state. A bean has to implement two well-separated interfaces:

1. Design Interface

   This interface is used to compose beans and is not used at all at runtime. The interface exposes the beans' properties (for instance the foreground colour), the methods that other components can call on the bean, and the set of events it emits.

2. Runtime Interface

   This interface is used only at runtime. It consists of methods that allow the bean to be manipulated, interact with the environment (e.g. through cut and paste), and handle events.

Beans are black-box entities. The only way to access beans' internals is through the introspection interface, which can expose only the information the developer decided to make public. Because beans can be passed as values at runtime, they can be considered first-class values and can also be used to build new beans. Composition can be performed at the object or class level. Two bean objects are composed by passing each other's reference whereas class composition is performed by subclassing. Object composition is dynamic and can be done by writing Java glue code, by using a scripting language or through property editors and composition wizards. In the case of a compiled composition, the structure of the new component can be serialised and stored in a persistent file called a *pickle*.

Contrary to OpenDoc, beans are specified in Java, lack a formal interface definition language, and do not have a solid semantic foundation. Nevertheless the fact that beans are based on Java has several advantages:

- they have a binary standard representation (Java byte-code);
- beans can be distributed and accessed remotely through the Java Remote Method Invocation (RMI) API;
- thanks to the Java serialisation mechanisms, beans can be easily moved on the network along with their state

NOTE

This facility is not available even of on mature architectures like CORBA.

Conclusion

Although Java Beans are a simple technology in comparison to other commercial components, they have many unique features which make them very attractive, especially in a distributed environment:

- flexible network access mechanisms: HTTP, Java RMI, Java IDL (Java's CORBA implementation) and JDBC (Java DataBase Connectivity);
- ability to move beans around easily, by exploiting the serialisation/deserialisation mechanisms;
- platform-independent component model;
- persistence and security facilities (provided by Java).

The ability to run beans unmodified on any platform that supports Java makes beans even more attractive. This is because the beans' component model is currently the only way to produce components, which run unmodified on different heterogeneous platforms.

3.1.4. Flexible Components

[Leeb96] presents a software architecture which enables end users to build applications by modifying and extending software components. These components are designed for building small, simple exploratory applications such as simple board games or mathematical models. The architecture is based on *flexible components*, which are executable pieces of software that can be developed, distributed, and reused independently. They constitute the application building blocks by being:

1. Programmable

   It is possible to modify and extend component functionality dynamically.

2. Interoperable

   Components can communicate with each other through standardised interfaces managed by a system component called the *component engine*.

3. Composable

   two or more components can be composed to form a new component by exploiting the component engine, which provides functionality for coordination, relation, and composition of components.

Flexible components can share parts and also be combined dynamically to build applications or compound components. They cooperate by communicating with each other and may be combined in hierarchy or grouping relations. Flexible components simplify programming by virtue of:

1. Reusability

   Components can be modified in order to be reused for different purposes, and to build a new application mostly by reusing existing building blocks.

2. Runtime modification

   Components can be modified, extended, and replaced in-place, i.e., within their application environment, in order to observe the effects of modifications immediately.

Most of the flexibility and power of flexible components derive from the fact that they have been implemented uniquely using dynamic object environments (a.k.a. *prototype-based programming environments*) [Wegner87] such as Self [Ungar87], Obliq [Cardelli94], or Cecil [Chambers93]. In these environments objects are called *prototypes* which exist on their own (i.e. they are not instances of a class) and are created by cloning existing objects. A prototype is a collection of data or methods members called *slots*. Prototypes usually offer a user interface that supports direct manipulation of their structures (i.e. the number, the name, and the type of their slots) and their behaviour (i.e. the functionality of their method slots). As prototype-based systems do not support most of the mechanisms present in class-based systems, they have introduced some alternatives such as:

1. Delegation

   This establishes a parent-child hierarchy among objects. A child can delegate some unknown slot to its parent which, if owner of the slot, will handle the message on behalf of the child.

2. Shared information

   As there are no objects which derive from a common base class, groups of methods that need to be used by several objects are shared among those objects and externalised in a *trait*. Unfortunately the of shared objects parts lead to a break down of the correctivness principle in prototype-based environments by introducing objects that do not behave like other objects.

Besides these advantages, prototype-based environments have several disadvantages with respect to object-oriented ones:

1. Object Interdependency

   Owing to information sharing, interdependency in a dynamic system can become complex, and the basic advantage of concreteness (object) being simpler to understand than abstractness (class) vanishes.

2. Interoperability

   Prototype-based environments are closed, because they offer little or no interoperability with other applications, making them unsuitable in cases where legacy code or objects created using other development languages have to be integrated into the application.

Component Architecture

Flexible components are binary objects written using a prototype-based language, whose functionality is encapsulated in methods and data members. The component interface allows the component behaviour to be controlled by executing methods, and its functionality to be modified by adding/deleting/modifying methods or data members. Each object has:

- a system-wide, unique, immutable identity;
- a modifiable name;
- a visual representation, i.e. an icon;
- data and methods, some of them being modifiable or removable depending on the component;
- a command interface (i.e. object's method and data at a certain time) used to access the component functionality, and a programming interface necessary to modify it.

The component engine is responsible for providing global services necessary to manage component interaction, such as:

- creation and destruction of components,
- component location and identification,
- component communication through message passing,
- component grouping in a parent-child or sharing (i.e. all the components that share data or methods) hierarchy,

- component composition in a cluster of components.

Conclusion

Although flexible components have some interesting capabilities, they are not suitable for developing real applications owing to the following limitations:

- Complexity

  The more flexible a component is, i.e. the more options it presents to the user, the more complex it is.

- Performance and Scalability

  Owing to the addition of several levels of indirection and to the interpretation of methods and message, the general application performance is degraded. Additionally, because component sharing is implemented replicating common data in each object and updating it every time the data change, this results in suboptimal memory usage and further performance degradation whenever common data is modified.

- Robustness and Reliability

  Components with multiple interfaces can be modified in multiple ways that may damage the global system integrity owing to the complex component composition scheme, which does not facilitate the immediate identification of problems due to interface modification.

In conclusion, although flexible components are a promising alternative to more traditional components such as OpenDoc or OLE, their numerous limitations do not permit their use for real application development but only for simple, experimental applications with a small number of components which users combine in many different ways and where flexibility and programmability are much more important that performance and reliability.

3.1.5. Apple QuickTime Component Manager

Apple QuickTime is a set of libraries and services for the manipulation of multimedia data, such as images, audio, and video. A component is a piece of code that provides a defined set of services to one or more clients. Applications, libraries, as well as other components can use the services of a component. A component typically provides a specific type of service to its clients and multiple components can provide the same type of service. All components of the same type support the same basic interface. Hence applications can use different components in order to obtain different levels of service.

The *Component Manager* (QTCM) [Apple93] provides access to components and manages them by keeping track of the currently available components and routing requests to the appropriate component. The Component Manager classifies components by three main criteria: the type of service provided, the level of service provided, and the component manufacturer. The Component Manager uses a component type to identify the type of service provided by a component.

Every component must have a single entry point that returns a value of type `ComponentResult` (a long integer). Whenever the Component Manager receives a request for a component, it calls that component's entry point and passes any parameters, along with information about the current connection, to a component parameters record. When a component receives a request, it examines the parameters to determine the nature of the request, performs the appropriate processing, sets an error code if necessary, and returns an appropriate function result to the Component Manager.

The component parameters record is defined by a data structure of type `ComponentParameters`.

```
TYPE ComponentParameters =
PACKED RECORD
   flags: Char; {reserved}
   paramSize: Char; {size of parameters}
   what: Integer; {request code}
   params: ARRAY[0..0] OF LongInt; {actual parameters}
END;
```

The `what` field contains a value that specifies the type of request. Acceptable values are for instance `kComponentTargetSelect` and `kComponentUnregisterSelect`, which are used respectively to register and unregister a component. The `params` field of the component parameters record is an array that contains the parameters specified by the application that called the component.

Whenever an application requests services from a component, the Component Manager calls the component and passes two parameters: the application's parameters in a component parameters record and a handle to the memory associated with the current connection. The component parameters record also contains information identifying the nature of the request. There are two classes of requests: requests that are defined by the Component Manager and requests that are defined by the component. When an application closes a connection to a component, the Component Manager issues a `close` request to that component. The component disposes of the memory associated with the connection and closes any files or connections to other components that it no longer needs.

The developer defines the interfaces supported by the component being developed by declaring a set of functions for use by applications. These function declarations specify the parameters that must be provided for each request. Once a component has been created, it has to be registered with the Component Manager in order to be used by other components and applications. There are two mechanisms for registering a component with the Component Manager. First, during startup processing, the Component Manager searches the components in a directory contained inside the directory where the operating system is stored. Second, an application can register the component and can specify whether the component should be made available to all applications (global registration) or only to the application (local registration).

Conclusion

The QuickTime Component Manager is a very simple mechanism to register components either application-wide or operating system-wide. In any case components can only be used by local applications and are not available to remote applications. A component is basically a function that provides a certain service which is identified by a four character long name. The Component Manager takes care of component registration and deregistration. In addition it is responsible for routing the client requests to the appropriate component. Unlike CORBA, there is not an interface definition language which can be used to specify the component interface. This restricts the component reuse because an application can use only those components whose interface has been made public, similar to OLE.

3.2. Plug-in Software Components

In the past few years, software developers have recognised the need to change the application development cycle. The necessity to support many media types and file formats, to add new tools, and ultimately to facilitate application tailoring, has contributed significantly to the creation of new types of software components. These software components do not usually constitute the core part of the application architecture (c.f. section 3.1) but they are used for side activities. For this reason they are often called *plug-ins* (see “Software Components” on page 14).

The following sections cover some types of software components that fall in this category. Although they have been developed for a very precise task, they have been selected because of the technology used to implement them.

3.2.1. CGI Applications

CGI (Common Gateway Interface) [CGI] is an interface defined by NCSA (National Center for Supercomputing Applications) between a web server and Unix™ applications. Before the advent of CGI, web server were able only to return static documents stored on the files system. With CGI, it has become possible to start applications when a specific URLs is requested to the server. CGI passes arguments, such as the request, to the application using the Unix environment. Basically, before launching the application, the web server ‘enriches’ the environment with information relative to the URL and then starts the application, which can fetch the information it needs from the environment. The output generated by the application is then returned to the requestor through the web server. Owing to the way the web server interacts with the CGI applications, it is not possible to qualify a CGI application as a component because it has only monodirectional plugs (see "Software Components" on page 14):

- a CGI application (in general) cannot interact with other CGI applications;
- the CGI application cannot send data to the web server, unless the web server issues a request to the CGI application.

3.2.2. Shell Applications

Shells are command-line interfaces that allow commands to be executed. Some commands may be hard-coded in the shell whereas others are stored somewhere on the file system and are executed when needed. The shell interacts with the applications it starts in two ways, through the parameters it passes to the application and through the environment. The shell is always the entity that initiates the operations. Shell applications usually return a return code to the shell in order to indicate whether the operation was completed successfully. Like in the previous case, shell applications cannot be considered components but rather plug-ins because they interact with the shell and with other components in a client-server mode rather than a peer-to-peer mode.

3.2.3. Plug-in Components

Several commercial applications [Adobe96] [Claris93] [Metrowerks96] use programming techniques similar to the component manager in order to achieve extensibility and customisation by allowing users to plug in components, often called plug-ins. Like the QuickTime Component Manager, these techniques define a component interface and a jump table containing functions pointers called by the application. Usually a plug-in cannot communicate with another plug-in while the application can access all of them. Additionally the plug-ins have quite limited capabilities and cannot access all the services and resources accessible by the application.

IMPORTANT

A comparison of the architectures and software components covered in the previous sections can be found in the "Comparison with Other Architectures" section on page 101.

3.3. Network Management Standards

Networks are used to interconnect heterogeneous systems and devices. In the past few years, hardware manufacturers have acknowledged the need to build software applications able to manage the different network equipment they produced. Unfortunately the interconnection of devices from different manufacturers has shown the limitations of ad hoc software for a specific device and it has been one of the reasons that pushed the network community towards a standard way to manage networks. For historical reasons network management is split into two distinct parts: OSI network management and Internet management. The former deals with the management of large public (tele)communication networks, the latter with management of devices attached to the Internet. In the past few years, a new management paradigm defined by OMG and based on CORBA is has been very successful. This is due to its ability both to operate (e.g. TCP/IP) and manage (e.g. SNMP, CMIP) networks, activities which instead are separated in OSI and Internet management.

The following sections introduce OSI, Internet, and OMG management, and cover the various predominant architectures and frameworks currently in use.

3.3.1. OSI Network Management

The OSI Network Management [Jeffree92] [Klerer88]:

- defines how management information has to be collected, represented, and transferred among open systems [Collins89] [ISO10165-1];
- provides a common terminology [ISO7498-4] necessary to flatten the differences among different systems and create new management standards;
- specifies the protocols used to exchange management information;
- defines the services and operations relevant for network management;
- identifies and defines the tools necessary to monitor, control, and coordinate open systems activities.

The *OSI Environment* is the set of resources that allows open systems to communicate according to OSI protocols and services. The *OSI Management Environment* is a subset thread that deals with the tools and services necessary to control and supervise interconnection activities and the resources relevant for management. The systems part of the OSI Environment can perform management activities themselves or they can cooperate with other open systems for the purpose of management. In the latter case OSI Management offers tools that:

- enable system managers to plan, organise, and control interconnection services;
- help maintain reliable connections among systems;
- protect management information through the authentication of senders and receivers of management information being exchanged.

Resources contained in the open systems part of the OSI Management Environment are represented by *managed objects* (MO) [ISO10040] [ISO10165-1] [ISO10165-2] and are managed using the protocols defined by OSI Management. An MO is an abstract representation of a physical (for instance a bridge or a router) or logical (for instance a communication protocol) entity. It is characterised by some attributes which represent its properties, its characteristics that need to be visible from the outside, the operations (methods) to which the MO responds, its behaviour, and the events (notifications) the MO emits asynchronously. For instance a counter can be represented by an

MO containing an attribute which stores the counter's current value and which can be manipulated using the get, set, and reset methods. These methods manipulate only the MO attributes. It is the MO's responsibility to synchronise the modification of the attribute with the current counter value, which may be stored in a physical network device. An MO is a member of a certain *object class* that characterises all the objects which have the same attributes, support the same operations, and emit the same notifications. MOs are defined using *GDMO* [GDMO] notation, defined by ISO in order to harmonise management information definition. GDMO is a formal language which allows users to specify MO characteristics. Unfortunately it is not powerful enough to specify MO behaviour which is defined in plain English. This limitation prevents GDMO from being a suitable fully automatic processing because a compiler cannot interpret the behaviour clause, thus making management application toolkits complex and unable to generate applications automatically without humans to code MO behaviour [Deri95b] [Halsall90].

OSI management model defines two hierarchical relations for MOs: inheritance and containment. Inheritance is a relation between hierarchical classes. Each object class inherits the characteristics (attributes, operations, notifications) of its superior class adding new characteristics or specialising some of them. For instance the object class "packet switching network" is derived from the class "network" in the inheritance hierarchy. Containment is applied to single MOs. An MO can contain other MOs, which can contain further MOs. For instance an MO representing a router can contain MOs representing the router communication ports, which can contain MOs representing the devices connected to such ports. Although inheritance is a concept present in object-oriented languages, containment is a characteristic of OSI management because in object-oriented languages there is no means of object containment and hierarchy. In conclusion, every resource relevant to the OSI Management is represented by an MO having some peculiar characteristics.

MOs can be specified for a single management layer (*N-layer MO*), for multiple layers or for system management (*system MO*). The following picture shows the OSI Stack (see "OSI Reference Model" on page 184) with the hierarchy of managed objects and it highlights the difference between N-layer MOs and system MOs.

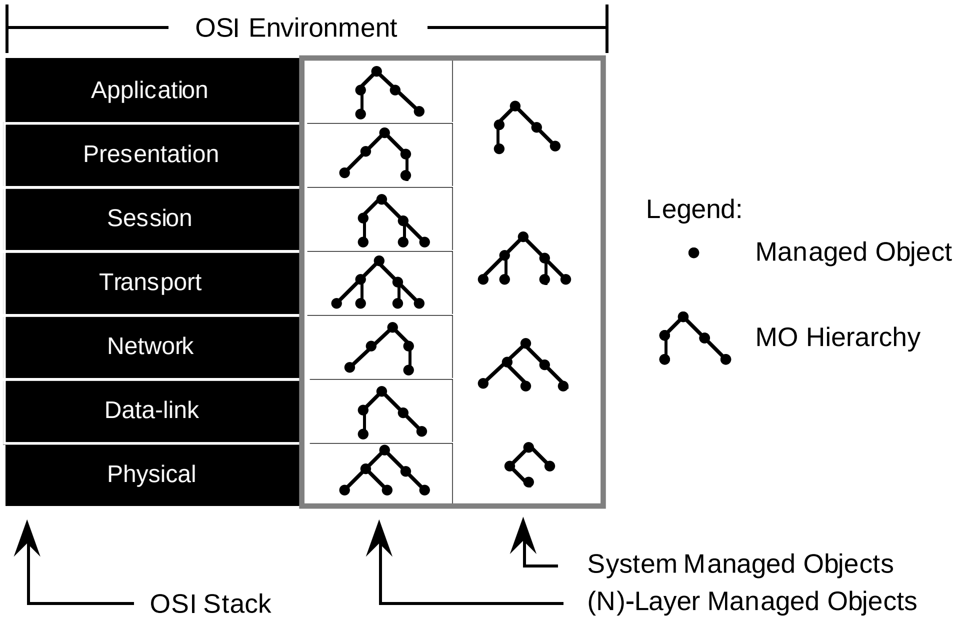


Figure 3. OSI: Environment: Managed Objects Hierarchy

In an open system the information relative to MOs is stored in the Management Information Base (MIB). Such information can be modified or transferred only by using OSI Management protocols. A consequence of this is that data contained in the MIB are logically organised according to the OSI Management specifications. Nevertheless OSI Management imposes no restrictions on data types, on the storage method (for instance database or flat files), on the abstract syntax, or the semantics used during data exchange. The formal language defined by OSI to define abstract syntaxes is *ASN.1* (Abstract Syntax Notation One) [ASN1], which provides a wide variety of types ranging from simple bit strings to complex structures (see "ASN.1" on page 186).

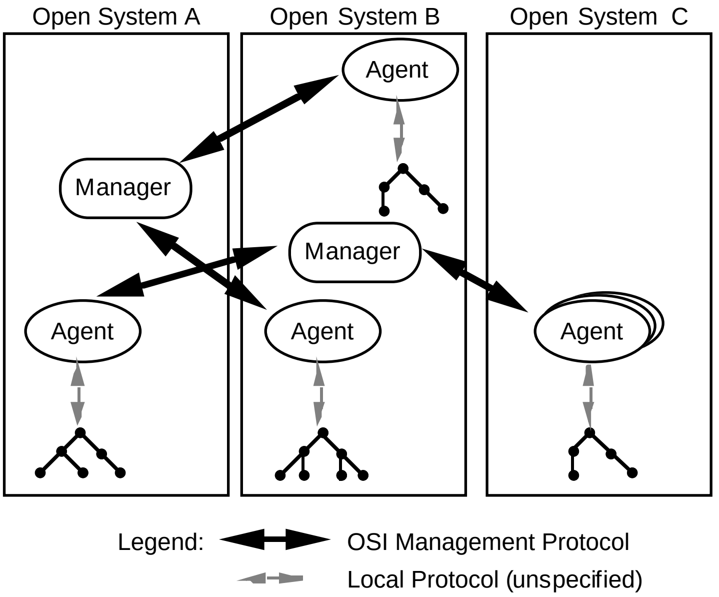


Figure 4. OSI Management Paradigm

The OSI Management paradigm is characterised by two processes: managing process (manager) and agent process (agent) [ISO10040]. The agent manipulates MOs directly by performing operations requested by managers. Agents are also responsible for sending events to managers when a certain situation occurs. An OSI open system can implement one or more agents or managers.

The *OSI System Management* defines the mechanisms used to monitor, control and coordinate MOs contained in an open system. It allows MOs contained in one or more layers to be managed and it is the only way, within OSI Management, to operate on different layers. System management communications are performed by the *System Management Application-Entity* (SMAE), which define the services offered by OSI management applications. SMAEs send requests and receive responses and notifications from/ to open systems. They are composed of one or more *Application Service Entity* (ASE), namely: SMASE (System Management Application Service Element), CMISE (Common Management Information Service Element), ROSE (Remote Operation Service Element) e ACSE (Association Control Service Element).

SMASE [ISO10040] is responsible for identifying and defining specific management aspects concerning the information which has to be exchanged among open systems.

CMISE (see “CMISE” on page 187) is composed of CMIS and CMIP, and permits OSI Management systems to share management information and to handle requests received from other systems.

CMIS [CMIS] is the general service employed by OSI Management to handle communications concerning the management of MOs and the transmission of notifications. It defines a set of service primitives, their parameters and the information necessary to describe each primitive semantically. The CMIS primitives allow:

- notifications to be received (M-EVENT-REPORT primitive);
- attributes to be read (M-GET and M-CANCEL-GET) and modified (M-SET);
- MO instances to be created and deleted (M-CREATE and M-DELETE);
- operation requests to be sent to another CMISE user (M-ACTION).

CMIP [CMIP] specifies the protocol used by CMIS and by N-layer MOs to exchange management information. It defines the procedures for exchanging management information between applications (CMIP abstract syntax), for the correct information control protocol interpretation and for the conformance tests employed to validate CMIP implementations.

ROSE [ISO9092-1] [ISO9092-2] allows remote operations to be requested and the corresponding responses to be received. ROSE can be considered the OSI service that corresponds to the Unix™ Remote Procedure Call (RPC). CMIP exploits the ROSE transactional services in order to send requests and receive responses.

ACSE [ACSE] provides services which establish (A-ASSOCIATE primitive) or release in a normal (A-RELEASE) or abnormal (A-ABORT) way logical associations between two application entities. It also provides the means to control an *application association*, which is a cooperation relationship between two application entities that communicate using a connection established in an open system environment.

OSI has identified five functional areas within OSI Management:
- fault management,
- configuration management,
- performance management,
- accounting management,
- security management.

The *fault management* provides the facilities necessary to:
- identify, diagnose, and correct network faults;
- notify the system managed about errors and abnormal situations;
- analyse error logs;
- perform diagnostic tests used to verify network activities.

The *configuration management* provides facilities necessary to:
- set key parameters of open systems;
- initialise and remove MOs;
- collect information about the system and notify state changes;
- modify the system configuration.

The *performance management* provides facilities for monitoring, controlling, analysing, and tuning system activities. The *account management* provides the facilities necessary to:
- collect information on the users and the resources they have used;
- calculate the costs involved in the utilisation of some MOs necessary to perform a certain activity;
- collect statistics concerning errors and anomalous situations.

The *security management* provides facilities necessary to:
- authenticate users who performed management requests;
- control and maintain the tools used for security;
- handle encryption keys;
- control and maintain access rights;
- manage and maintain security logs.

3.3.2. Internet Network Management

SNMP (Simple Network Management Protocol) [SNMP] is the management protocol used to manage Internet network devices. It has been created for research purposes by four professors who needed to manage the devices attached to their campus network. The main requirements were that SNMP be:

- a simple protocol with low impact on the managed devices;
- able to continue working when the network is collapsing;
- capable of managing both current and future networks.

SNMP is able to accomplish many tasks including accounting, configuration, testing and fault detection/correction. Although this protocol was born in an academic environment, it has been widely adopted by many computer manufacturers. Since then it has rapidly become the management protocol for network devices connected to LANs and is by far the most used protocol, whereas CMIP is employed primarily on telecommunication networks. SNMP is a protocol part of the application level of the TCP/IP stack (TCP/IP is the protocol used in the Internet) and it defines the formal structure of communications between devices attached to different interconnected networks. The content and the structure of SNMP data is defined in ASN.1, which has been selected because it is a standard language and also because it is able to separate data cleanly from their effective representation for transmission.

SNMP's network architecture comprises three components:

- the SNMP protocol;
- the management information base (MIB) [RFC1156] [RFC1158] [RFC1907];
- the structure of the managed information (SMI) [RFC1155] [RFC1902].

SMI specifies how the information relevant for management has to be represented inside the MIB. The *MIB* is a set of MOs (in SNMP they are usually called *MIB variables*) which describe the logical and physical network resources, and a set of test points and controls which a management system based on SNMP has to support. The MIB also names the MOs according to the hierarchy used by OSI for network object naming using object identifiers (see "Management Naming Scheme: Object Identifiers" on page 191). Following this naming convention, SNMP retrieves the MOs contained in the MIB, which are logically interconnected in a table-like structure.

The software for network management is usually contained in the network devices part of the *Network Management System* (NMS), a system which controls and monitors network devices produced by different vendors, sends requests, and receives asynchronous events (traps) from the SNMP agents contained in the devices shown in the picture below.

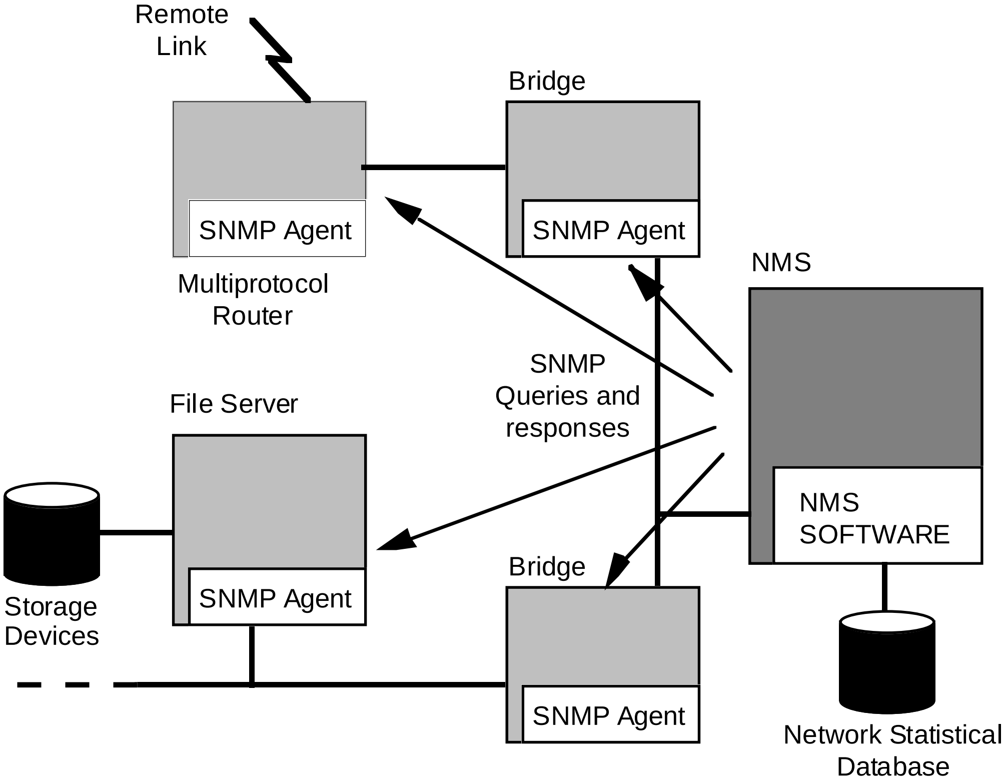


Figure 5. SNMP Network Management System

SNMP agents are typically small software applications that maintain information regarding specific MOs which control critical resources (for instance the state of an Ethernet interface of the number of errors received). Agents are also responsible for processing requests sent by managers in a very similar way to OSI Management, for generating alarms when a critical condition is verified, when certain network resources are not available or when devices are powered up/down. As SNMP agents are usually inside the network devices (hence they are often stored in some persistent memory), they necessarily have to be quite simple and able to continue working even when the network is collapsing (this is one of the fundamental SNMP paradigms). Other than being simple, SNMP is an extensible protocol. Instead of using complex data types to represent network resources like CMIP does, SNMP uses only a few simple types (about ten), which are composed to build more complex types. As the types are simple, it is quite easy to extend the MIB by adding new variables. In order to maintain a consistent variable naming convention that leaves room for new variables, SMI divided the variables into four parts (see “Management Naming Scheme: Object Identifiers” on page 191):

- directory: identifies the MOs reserved for future use;
- management: identifies the MOs that need to be implemented by every SNMP implementation;
- experimental: contains the MOs used for research purposes and which may become part of future MIB extensions;
- private: contains MOs defined by hardware manufacturers in order to manage the devices they produce.

Thanks to these classifications, SNMP has become a very long-lived protocol. In fact, as soon as new MOs become available, new software applications can use them while old applications can continue to work unmodified without being influenced by the new MIB variables.

3.3.3. OMG Network Management

In order to describe OMG management it is necessary to first define the terms and concepts proper of this management paradigm. An *object model* provides an organised presentation of object concepts and terminology [OMG95]. It also defines a partial model for computation that embodies the key characteristics of such objects. The object model describes concepts useful to client application such as use of the objects, object creation and identity, requests and operations, types and signatures. It then describes concepts related to object implementations such as methods, execution engines and activation. An *object system* is a collection of objects that isolate the requestors of the services (clients) from the provides of services (servers) by a well defined encapsulating interface.
In the CORBA object model [OMG92] [OMG91] [Yang96], *objects* are identifiable, encapsulated entities that provide one or more services which can be requested by clients. Objects encapsulate state (attributes) and behaviour (operations) that can be accessed through an *interface*, which is a description of a set of possible operations that a client may request of an object. Object interfaces are specified in OMG IDL, a language used to define the object interface but not the object behaviour. IDL, like ASN.1, is not a programming language, so it has to be complied in a programming language such as C++ or FORTRAN. Supposing we use C++, then the CORBA interfaces for some specific objects are processed as follows:

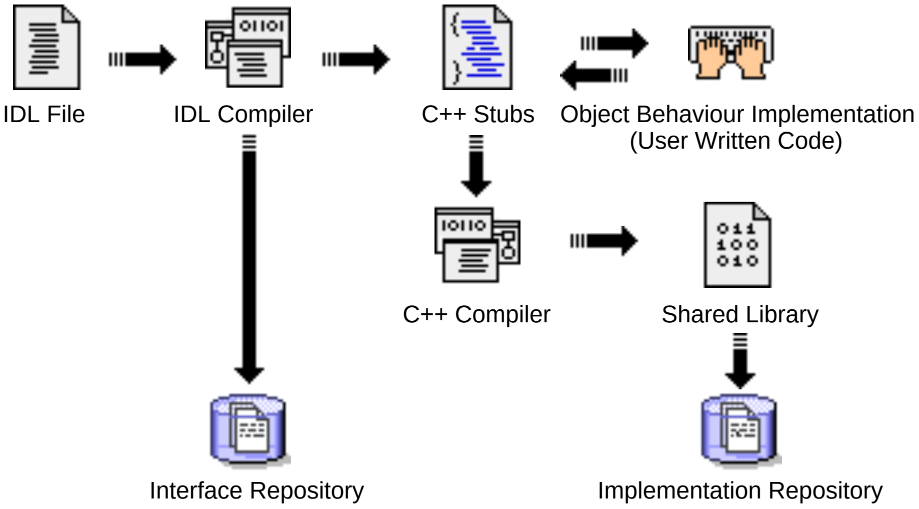

IDL File
IDL Compiler
C++ Stubs
Object Behaviour Implementation (User Written Code)
C++ Compiler
Shared Library
Interface Repository
Implementation Repository

Figure 6. CORBA Interfaces Implementation using the C++ Language

The IDL compiler compiles the IDL file containing the object interfaces and updates the *interface repository*, a persistent-type repository of objects representing the elements of interface definitions, created and maintained based on information supplied in the IDL source file. In addition the IDL compiler generates C++ stubs for each interface method. The stubs are empty, i.e. the object behaviour has still to be implemented. The user has to write this code: there is no way for an automatic tool to do it because the objects represent a

real resource whose behaviour is not specified in the IDL file. Once this step has been done, the stubs are compiled and a shared library (usually the preferred format) is generated. Such a library is then added to the *implementation repository*, a database which contains the implementation definitions of CORBA objects, i.e. the shared information on the location of the libraries that implement the CORBA objects and their relative object classes.
On each host on which CORBA instances are to be created there is an *Object Request Broker* (ORB), a.k.a. a CORBA server, responsible for handling client requests, locating the requested objects and their implementations, performing the request on the objects, and returning the results, if any, to the client.

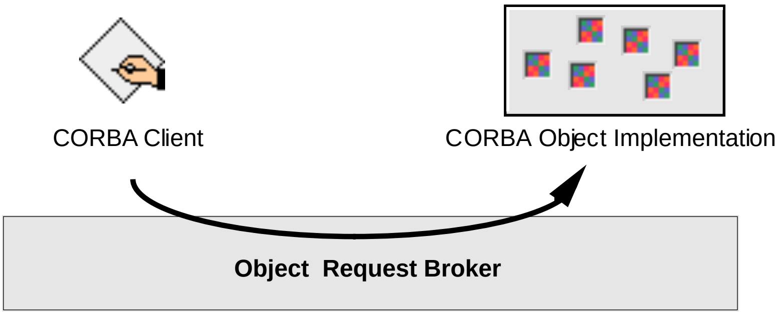


Figure 7. Client Access to CORBA Objects through the ORB

The ORB is also responsible for creating and deleting objects on behalf of clients. In order to serve such requests, the server accesses the interface and the implementation repository. Whenever a client application creates a remote object, the ORB uses the implementation repository to gain access to the code that implements the requested CORBA object. Once the code which implements the object has been located, it instantiates a new object instance locally. The client application receives from the object creation an object reference, allocated in the client's address space, which identifies the real instance created in the server application. Owing to this mechanism, instances cannot be manipulated directly by clients but they do access them transparently through the ORB. Instead, if the instances are created locally, i.e. in the client address space, there is no interaction with the server, and the instances behave like a normal non-CORBA instance. Nevertheless if the instance is created locally, the instance is private to the application and there is no way for external clients to access it.

The goal of NMF-X/Open Joint Interdomain Management task force (*XoJIDM*) is to reconcile the OSI and Internet models using CORBA. Specifically it focuses on the management of OSI and SNMP resources using CORBA and vice versa. Assuming that future network management applications will be written using CORBA, it is still necessary to access network resources managed existing object models in order to preserve large investments. In order to reconcile the models, XoJIDM follows three approaches:

- model alignment by identifying of the core elements of the different models to be unified, omitting model features that may be difficult to align;

- runtime mediation between implementations of the models by means of proxy applications;
- use of notation mapping tools based upon translation algorithms allows OSI/Internet objects to be mapped into CORBA object and vice versa.

Basically XoJIDM’s approach consists of two parts:

- *Specification Translation* [XoJIDM95a], which defines the static translation of GDMO/ASN.1 to IDL;
- *Interaction Translation* [XoJIDM95b], which specifies how the mapping is used at runtime.

The goal of this effort is to translate the MIB of an agent (GDMO/ASN.1) into CORBA IDL, which can subsequently be used to manage the agent using CORBA [Soukouti95]. The GDMO/ASN.1 documents describing the agent's MIB are translated into IDL and then to a server implementation. GDMO templates are mapped to IDL interfaces, and each ASN.1 type is translated to a corresponding IDL type. The generated IDL interfaces representing GDMO class templates will include IDL attributes generated from ASN.1 types. IDL is then compiled to produce the client and server stubs in the desired language binding (e.g. C++). The server stub and the implementation code generated by the GDMO/ASN.1-IDL compiler are compiled and linked to produce a CORBA implementation. The client stubs are compiled and linked with the user application. Information on available CORBA interfaces (which represent CMIP instances) is contained in the generated client stubs and therefore known to the client at compile time. At runtime, the client proxies forward any request they receive to their corresponding objects in the CORBA server. These will use the implementation code generated by the GDMO/ASN.1-IDL compiler to communicate with the managed objects in an agent using CMIP/SNMP usually using XOM [XOM] and XMP [XMP], which are the X/Open’s proposed C-interfaces for abstract syntax and management operations.

As XoJIDM’s approach relies strongly on static mappings between GDMO/ASN.1 and IDL and vice versa, it has several drawbacks:

- each modification to GDMO/ASN.1 leads to a regeneration of the IDL code, hence to the recompilation and modification of the client applications;
- including static stubs in client applications generates large applications because the translation generates a large amount of code (as seen above, this is a problem that affects network management applications in general);
- owing to the dynamic nature of ASN.1 and GDMO that some types are known only at runtime, code for all the possible cases is generated and included in the client applications.

In the “CORBA Interfaces” section on page 129 a new dynamic approach is presented, which allows most of the limitations due to the static nature of XoJIDM’s approach to be overcome.

3.3.4. Comparison of Network Management Architectures

The following table presents a comparison of the network management architectures covered in the previous sections [Rutt94]:

	OSI	Internet	OMG
Goal	Distributed Network Management.	Management of Internet networked devices.	Object-oriented distributed systems development.
Interoperability	At syntactic and semantic level. Communications interoperability.	At syntactic and semantic level. Communications interoperability.	At syntactic and semantic level. No communications interoperability (provided by CORBA v.2).
Portability	No code portability.	No code portability.	Code portability.
User Advantage	Management of heterogeneous network components.	Management of heterogeneous internetworked devices.	Transparency of applications of the underlying heterogeneous platforms.
Reusability	Library of management information.	MIB specifications.	Interface Type Library.
Object Interface Type	Communications interface agents and managers.	Communications interface between NMS and SNMP agents.	Programmatic interface (signature only) between client and server.
Transmission Protocol	CMIP.	SNMP.	Independent of the protocol (IIOP in CORBA v.2).
Open Interface[a]	ISO/IEC standards and other industrial standards [OMNIPoint93].	Many RFC, including MIB [RFC1155] and MIB-II [RFC1158].	IDL.
Protocol Model	Nonblocking message-passing with normal and exception reply types. A single request may result in multiple replies.	Nonblocking message-passing usually without guarantee of delivery. A single request results in (at most) a single reply.	Two styles: at most once (blocking with exceptions) and best effort (nonblocking, no guarantee of delivery).
Interface Concurrency[b]	Yes.	Polling driven, managers control concurrency.	Not precluded by the object model.
Object Description	MO is characterised by the attributes it makes available, its behaviour and the invocations and notifications that cross its boundary. There is support for inheritance, polymorphism, encapsulation.	MO represent individual MIB variables manipulated through SNMP. Each object has a name, a syntax, and an encoding. Inheritance is not supported.	An object is a package of data and code used to model an application entity. There is support for encapsulation and interface inheritance.
Object Operations	Operations can be confirmed or not, exceptions are signalled by means of exception replies.	Operations are confirmed and always generate a single response. Exceptions are signalled as responses.	Operations result in a single reply (the return of control to the invoker), exceptions are indicated by exception signals.
Object Events	Notifications (confirmed/ unconfirmed).	Traps (unconfirmed).	Events are sent to the event notification service, which are distributed via a push or poll model.

Table 4. Comparison of the OSI, Internet, and OMG Management Models

	OSI	Internet	OMG
Object Behaviour	Supported (described in GDMO in plain English).	Supported (described inside the MIB in plain English).	Not specified in IDL.
Object Life-Cycle Operations	Built-in on CMIP (M-CREATE and M-DELETE).	No direct support (emulated through the Set).	No built-in creation (factory objects can be defined).
Data Types	All ASN.1-defined types.	10 ASN.1 defined types.	All IDL-defined types[c].
Encapsulation	Supported.	Not supported.	Supported.
Object Reference	Object Instance.	Object Identifier.	Object Reference (Opaque datatype).
Interface Reference	Object Identifier.	Object Identifier.	Each interface has a reference Id through Module names.
Specification Tools	GDMO templates.	ASN.1 macros.	IDL.
Object Taxonomy	Containment tree.	Naming hierarchy through object identifiers.	Object types hierarchy by means of multiple inheritance.
Access Transparency[d]	Not supported.	Not supported.	Supported (ORBs may use whatever protocol is most appropriate).
Object Location Transparency[e]	Not supported.	Not supported.	Supported in CORBA.
Object Location Independence[f]	Not supported.	Not supported.	Supported in CORBA.

Table 4. Comparison of the OSI, Internet, and OMG Management Models

[a] An interface that supports a published protocol and at which conformance may be tested.

[b] The property of an interface such that it will accept protocol elements at any time.

[c] IDL is much less flexible than ASN.1, and not all the ASN.1 types can be defined in IDL.

[d] An object reference does not reveal/define the transport protocol for that object.

[e] An object reference does not reveal its location.

[f] An object reference remains valid after the object changes location

3.4. Network Management Research

Network management research can be divided into two large groups: OSI/Internet management and interdomain management. This division is necessary to distinguish between platforms and frameworks for the purpose of OSI/Internet network management and for new efforts to manage existing network resources using CORBA. In the first case the goal is to manage network resources efficiently using existing standards. In the second case the goal is to define new mappings, either similar or quite different to the one proposed by OMG, that allow CMIP and SNMP protocols to be mapped to CORBA and vice versa.

3.4.1. OSI and Internet Management

Given that SNMP is based on TCP/IP, the implementation of SNMP applications mostly concern implementations of functions able to encode/de-

code the (few) SNMP types to BER (see “ASN.1” on page 186). The situation is quite different for OSI. CMIP, the protocol used for OSI management, is much more complex than SNMP; it is based on OSI transport protocols and requires an OSI stack (see “OSI Reference Model” on page 184) in order to work. Therefore, the implementation of OSI applications presents problems for the implementation of both the CMIP protocol and of the application itself.
In addition, many problems arise from the integration of CMIP with SNMP because quite often CMIP is used for managing large networks whereas SNMP is used to manage locally the devices part of the network. Therefore, the industry and research have acknowledged the need to define new management paradigms to permit the seamless integration of CMIP with SNMP in order to facilitate the development of hybrid management applications.

The following sections present:

- ISODE

  The first (public domain) implementation of OSI protocols and SNMP. It is considered a milestone in the field of network management and has deeply influenced many other implementations and products.

- Classic Management Platforms

  Classic management platforms such as IBM NetView are the market product leaders for system and network management, and have deeply influenced some parts of this work because they represent the stereotype of how most of management applications work.

- XOM/XMP

  X/Open’s solution to seamless CMIP/SNMP integration has been adopted by most vendors.

- Java Management API

  The first Java API for management (including SNMP management). This is the first concrete effort to define management capabilities using Java.

- Web-based Management

  This section presents some solutions for web-based management, defined after the release of Liaison, and which are useful to position web-based management capabilities offered by Liaison.

- TCL/Perl-based Management

  This category includes attempts to simplify the way network management is performed by using some simple scripting languages.

3.4.1.1. ISODE

ISODE (ISO Development Environment) is a research tool developed to study the upper layers of OSI [Rose89]. At the moment, ISODE is available in two forms: an old public-domain version and an updated commercial version. ISODE supports OSI directory services, OSI mail, some integration APIs from X/Open, and SNMP management capabilities. CMIP support is

provided through an ISODE extension called *OSIMIS* (OSI Management Information Service) [OSIMIS], to which the author contributed by implementing the CMIP access control support [Knight94]. ISODE is written in C, whereas OSIMIS uses C++. ASN.1 data encoding/decoding is based on C functions generated by the ASN.1 compiler called PEPY (Presentation Element Parser Yacc-based). As OSIMIS relies on PEPY, OSIMIS-based management applications are quite large because of the large amount of code generated by PEPY. In addition, OSIMIS applications do not use metadata at all, hence all the management information has to be compiled and linked with the final applications.
In conclusion, besides the fact that both ISODE and OSIMIS have facilitated the diffusion of OSI protocols, they are based on old concepts, basically the "Vi, Unix and C" school, so applications based on these platforms are affected by technology limitations.

3.4.1.2. Classic Management Platforms

Products such as IBM NetView [IBM95a], HP OpenView [HP_DM], and Sun Solstice [Solstice] belong in this category. The common philosophy behind these products is their flexibility and their wide spectrum of subproducts which provide solutions to many problems. These products run on mid/high-end Unix workstations and usually require access to an SQL database where network information is stored. Owing to the large number of tools and components parts of this kind of platform, their price is quite high and the applications that can be based on them are closely tied to them. As these platforms have been developed over several years and sometimes merged with products coming from other vendors, the set of APIs available to programmers is often unrelated and based on different philosophies.

In conclusion, although these platforms provide the APIs and tools necessary to solve most management problems, unless the platform comes with a tool which exactly solves the user's management problem, the development of custom applications is costly for several reasons. The main reason is that a the large number of APIs is necessary to build an application that is integrated into the environment. Note that this does not guarantees that the developed application will be able to run on the same platform on different operating systems without major modifications. Moreover, the high platform costs, the need to run on mid/high-end machines, and the fact that the developed applications are tied to the platform running on a certain operating system makes them suitable only for large universities or telecommunication companies but not for small companies or institutions.

3.4.1.3. XOM/XMP

X/Open has defined a single API to support the communication of management information offered by CMIP and SNMP. *XOM* (X/Open OSI-Abstract-Data-Manipulation) [XOM] is an API to manipulate ASN.1 values which are mapped through a compiler to C datatype which is then manipulated by XOM. GDMO documents, SNMP MIBs, and ASN.1 documents are translated into XOM objects (actually they are C structs) according to an

algorithm defined by X/Open [GDMO_XOM]. *XMP* (X/Open Management Protocols) [XMP] instead, is an API to manage CMIP and SNMP instances based on XOM.
Although XOM/XMP was announced by X/Open as a big leap in the industry because it was supposed to guarantee interoperability at the source code level of management applications, it has virtually missed the target. The main reasons for this failure are:

- the fact that XOM/XMP implementations do not really guarantee interoperability at the source code level;
- the extreme complexity of XOM, which also affects applications in terms of performance;
- the fact that XOM objects are difficult to manipulate and are not defined in an object-oriented language;
- being that XOM objects are complex to manipulate, many users added a C++ layer which simplified object manipulation but jeopardised XOM/XMP because developed applications are tied to a specific implementation and also because these applications become fatter, slower and more memory-hungry due to this extra layer.

NMF and X/Open recently acknowledged the need to define a fully object-oriented framework for network management and released an early draft of TMN++ [CMIS++] based on XOM/XMP and restricted only to CMIP. TMN++ is supposed to put a C++ object-oriented layer on top of XOM/XMP and thus enable the construction of real object-oriented applications while preserving the investment in XOM/XMP.

3.4.1.4. Java Management API

The *Java Management API* (JMAPI) [JMAPI] allows any Java graphical user interface application running in a Java-enabled browser to interact with managed object instances on a central server. It also enables objects to manipulate any number of appliances (i.e. management target systems) through remote agent objects. Managed objects implement distributed management functionality. These objects map closely to the resources that are to be managed. Basically, the managed objects are a proxy to the real resources. These objects supply the interfaces that perform the real management operations, and all inherit directly or indirectly from the `managedObject` class. The `managedObject` class provides the methods that allow object developers to build on the database, security, and distribution elements of the architecture.

Among the protocols supported by JMAPI is SNMP, hence it is possible to manage network resources using the JMAPI. JMAPI provides the following for integration with existing SNMP agents:

- a set of Java classes that implements the SNMP protocol and interfaces to use the protocol;
- a set of JMAPI managed objects to more easily use the Java classes that implement the SNMP protocol;
- a JMAPI managed object that receives all SNMP traps and converts them into instances of JMAPI events.

Although JMAPI has not yet been finalised, it has been accepted by the industry and research, and hence is likely to become very important. It supports only SNMP and relies heavily on Java protocols such as RMI (Remote Method Invocation), which makes it unsuitable for anything but pure Java environments.

3.4.1.5. Web-based Management

Pushed by the inherent limitations of many management platforms (see "Classic Management Platforms" on page 63) especially in terms of central management consoles, many researchers have found that more flexible access to management information can be gained using the web over dial-up links. Besides Liaison, which was the first publicly available application that implemented web-based management capabilities, many companies and institutions have released applications and tools for integrating management capabilities within the web [Jander96] [Damocles95] [Marben95] [Knight95]. The purpose of most of these tools is to interface existing management applications with HTML pages to be able to perform limited management from the web. Nevertheless, this way of approaching the problem is somehow orthogonal to the purpose of this thesis which attempts to define an architecture able to exploit new technologies to create a new generation of management applications and not to modernise existing ones. The subtle difference is that in the first case the new technologies are (potentially) fully exploited whereas in the second case they are only partially used. This is because when there is some legacy code involved like in the second case, developers do not have much freedom. This hypothesis is proved by the fact that every web-based solution currently available is limited to only one management protocol (either SNMP or CMIP), it is often an add-on to an existing platform which provides web-visibility only, rarely it is open to extensions in terms of programming APIs but is moderately customisable concerning only the appearance of web-pages.

In other cases [Hudis96] [Vertel97] companies driven primarily by Microsoft are trying to define an enhanced version of HTTP tuned for web-based management which can easily be combined with Microsoft technologies such as DCOM and ActiveX. Although the later approach involves several companies, it is not perfectly coherent with OSI and SNMP because it is based on proprietary technologies and because it does not use standard protocols but attempts to define new proprietary ones.

3.4.1.6. TCL/Perl-based Management

The difficulties, especially in terms of complexity, encountered while developing management applications using technologies based on static compilation, C language, and automatic code generation have prompted some research groups to find new ways to develop management applications. The requirements are faster application development, platform-independent graphical appearance, ease of use, and portability. A good approach to this problem is the adoption of powerful scripting languages such as TCL [Ousterhout94] and Perl [Wall96], which have been combined with powerful toolkits for graphical user interface development such as Tk [Ousterhout94].

Many efforts undertaken in this area [Schönwälder95] [Zwemmer96] [Pavlou96], have demonstrated the feasibility of this approach although they have also shown certain limitations in terms of performance, code maintenance, inability to develop large applications, and excessive use of memory.

3.4.2. Interdomain Management

Interdomain management is a new branch of the management field which attempts to defines mappings from CMIP/SNMP to CORBA and vice versa. This is done to develop future management applications using CORBA which transparently manage CMIP/SNMP-based network devices.

This section is divided into two parts according to the way in which the mappings have been defined:

- Static Mapping

  Into this category fall the mappings defined at compile time, similar to the XoJIDM approach (see "OMG Network Management" on page 57).

- Dynamic Mapping

  Dynamic mapping is a new approach which performs the mapping at runtime by exploiting metadata information derived from the processing GDMO/SNMP documents that define the structure of the managed network devices.

3.4.2.1. Static Mapping

Static mappings are based on the idea that the CMIP/SNMP management information has to be mapped to the management domain (a CORBA-like implementation) using static translation tools. The motivation to develop this solution is that users accustomed to CORBA environments want to deal with CORBA objects, usually represented by C++ objects, which represent the management information. For instance if a managed object has an enumerated attribute which represents its state (possible values are up, down), the translation tool will generate a C++ class which looks something like this:

```
typedef enum { up, down } State;

class StateObj {
private:
   State state;

public:
   StateObj();
   ~StateObj();

   void  SetState(State value);
   State GetState();
   void  RestoreDefaultState();
};
```

Figure 8. Simple C++ Type Corresponding to an Attribute Contained in a Managed Object

The advantage of having a type in a programming language (C++ in this case) that corresponds to an attribute is in terms of generated code and flexibility:

- each attribute is mapped to one or more C++ types/classes (one type and one class in the example below);
- supposing we have a similar type which can assume the values `up`, `stand_by`, `down`, it is necessary to generate two new types instead of extending existing ones (`State` and `StateObj` in this case);
- because managed information is quite dynamic (i.e. two managed objects that are members of the same class may be quite different in terms of the number of their attributes. Therefore attributes are present/absent depending on the object state); each object has to contain the superset of all possible attributes that the class may have (for example, in the class `UnixWorkstation`, each instance must have as attributes `displayType`, `CDROM`, `FloppyDrive` and so on although a diskless workstation has none of them because it does not support those devices);
- every time the managed information changes, it is necessary to perform mapping, and to link/modify the applications even if the modifications to the managed information are not relevant to the managed application;
- even if the managed applications are not too complex, they use considerable memory due to the (shared) libraries that contain the generated code and are linked with the application.

This approach maps all the metadata information to static classes which are then compiled and linked with every application although the application may access only a small subset of them. Note that, because all the generated classes make extensive use of inheritance, there are many cross-relationships (i.e. all the datatype are closely tied) hence an application that uses, for instance, only one type has to be linked with the full library which contains all the types. Most of the methods rely on automatic translation techniques, which are performed by off-line compilers [Kong96] [Schürfeld94] [Soukouti95], whereas other approaches [Genilloud96] require manual interaction because fully automatic translation is inefficient.

3.4.2.2. Dynamic Mapping

This category is almost empty although it seems to be quite a reasonable approach to interdomain management. The only known approach belonging to this category is *GOM* (Generic Object Model) [Ban96] [Ban97], a framework for managing instances of multiple object models such as CORBA, CMIP or SNMP. GOM uses the concept of reification, modelling elements of object-oriented models as objects themselves [Maes87]. Thus, all CORBA interfaces or GDMO class templates that will ever be encountered are mapped to an instance of the generic GOM class `GenObj`, all attributes to instances of `Attribute`, and all values to instances of `Val`. As client applications do not have to include type information about available classes at compile time, they are typically very small. Moreover, when a specification is modified, clients do not have to be recompiled. Only modified specifications have to be parsed and fed into the Metadata Information Database (MID) via off line tools. The MID is used by the adapters to translate GOM into a specific model (for instance OSI) and vice versa. Contrary to the static approaches described in the previous section, there is no compiled-in knowledge of any classes in the system; adapters rely entirely on metadata about the CORBA, SNMP or CMIP classes available to perform their work. In practice the static approaches store the metadata information using C++, whereas GOM stores the information into the MID. Thus GOM produces smaller applications, that are easy to modify when the metadata information changes because basically only the MID is updated and the application remains untouched.

3.5. Do We Really Need Yet Another Architecture?

The sections above describe relevant efforts in the field of software components and network management. One may ask why there is a need to define a new architecture and a new type of software components instead of reusing existing technology. To do this, it is worthwhile to verify whether the technologies shown above satisfy the requirements stated in the previous chapter (see “Thesis Requirements” on page 23).

	OpenDoc	OLE	JavaBeans	Flexible Components	Plug-Ins
Suitable for Management Application Development	Yes (if only DSOM is used)	Unknown (probably not)	No	No	Unknown (probably not)
Available when this Work Started/Now	No/Partial	Partial/Partial (DCOM is available only on NT4)	No/Yes	No/Yes	Yes
Application Extensibility	Yes	Yes	Yes	Yes	Yes
Application Evolution at Runtime	No	No	No	Yes	Partial
Ease of Use	High	High	High	High	High

	OpenDoc	OLE	JavaBeans	Flexible Components	Plug-Ins
Ease of Development	Low (OD is huge)	Low (COM is very complex)	High	High	High
Distributed Environment Support	No	Partial (only with DCOM)	Yes	No	No
Promotes Reuse	Yes	Yes	Yes	Yes	Partial
Ability to Run with Limited Resources	Partial	Partial	Partial	No (dynamic languages need a lot of resources)	Unknown (probably not)
Portability and Genericity	Limited	Limited	Yes	No	Unknown
Based on Open Standards	Partial	No	Yes	No	Unknown (probably not)
Internet-aware	No	No	Yes	No	No
Slim and Efficient	Partial	Partial	Yes	No	Partial
Scalable	No	No	No	No	No
Performant	Partial	Partial	Partial (limited by the Java VM)	No	Unknown (probably yes)
Technology and Platform Independent	No	No	Yes	No (dependent on the availability of the language)	Unknown (probably yes)

Table 5. State-of-the-Art Software Components vs. Thesis Requirements

	ISODE OSIMIS	Classic Mgmt Platforms	XOM XMP	JMAPI	Web-based Mgmt	TCP/Perl Mgmt	Static Interdom. Mgmt	Dynamic Interdom. Mgmt
Suitable for Management Application Development	Yes	Yes	Yes	Partial (support only for SNMP)	Partial (dynamic events are handled using only Java)	Yes	Yes	Yes
Available at the Time This Work Started/ Now	Yes	Yes	Yes	No/Partial (alpha)	No/Yes	Yes	Partial/ Partial (some mappings are undefined)	No/Partial (no appl. supports both CMIP and SNMP)
Application Extensibility	No	No	Difficult	Unknown	Unknown	Yes	Difficult	Yes
Application Evolution at Runtime	No	No	Difficult	Possible	No	Possible	No	No
Ease of Use	Low	Low	Unknown	High	High	High	Unknown	Unknown

	ISODE OSIMIS	Classic Mgmt Platforms	XOM XMP	JMAPI	Web-based Mgmt	TCP/Perl Mgmt	Static Interdom. Mgmt	Dynamic Interdom. Mgmt
Ease of Development	Low	Low	Low	Average	Unknown	High	High	High
Distributed Environment Support	Partial	Partial	Unknown	Yes	Yes	Unknown (probably not)	Yes	Yes
Promote Reuse	No	No	Partial	Partial	Unknown	Unknown (probably yes)	No	Partial
Ability to Run with Limited Resources	No	No	Partial	Yes	Yes	Partial	No	Partial
Portability and Genericity	Yes	Partial	Yes	Yes	Yes	Yes	Low (ORB dependent)	Low (ORB dependent)
Based on Open Standards	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Internet-aware	No	No	No	Yes	Yes	Partial	No	No
Slim and Efficient	No	No	No	Partial (limited by the Java VM)	Yes	Partial	No	Partial (needs a MID)
Scalable	Partial	Partial	Unknown	Unknown (probably yes)	Unknown (impl. dependent)	Unknown (probably not)	Yes (thanks to CORBA)	Yes (thanks to CORBA)
Performant	Yes	Partial (they require powerful hosts)	No	Unknown (probably yes)	Yes	Yes	Yes	Yes
Technology and Platform Independent	Yes	No	Partially (potentially yes, but basically tied to UNIX)	Yes	Yes	Yes	No (ORB dependent)	No (ORB dependent)

Table 6. State-of-the-Art Management Architectures/Platforms vs. Thesis Requirements

From the two tables above, it is possible to conclude that:

- none of the component technologies nor the management architectures/ platforms satisfy all the mandatory thesis requirements;
- some technologies satisfy most of the requirements (for instance Java Beans and JMAPI) but unfortunately have significant limitations (for instance JMAPI supports only SNMP).

As far as the management is concerned, it is worth mentioning that a truly flexible and *modern management platform* should:

- provide facilities for developing CMIP and SNMP applications using conventional technologies such as pure C++;
- provide facilities for developing CORBA-based applications, because this is likely to become the primarily way to develop future management applications;
- offer Internet visibility (for instance through the web);
- allow applications to be developed in different ways (for instance using C++, Java or interpreted languages) according to the requirements.

As shown in the previous table, none of the management architectures/ platforms satisfy all these requirements.

The conclusion is that, because none of the current technologies satisfy the requirements, it is necessary to develop a new architecture that allows:

- all requirements to be satisfied,
- a truly modern management architecture to be developed, according to the definition given above.

This what prompted the author to develop Yasmin and Liaison.

4 Yasmin: the Architecture

4.1. Introduction

This chapter presents Yasmin, a component-based architecture and framework for software applications. Yasmin has been developed with the intent to simplify the implementation of open distributed applications. Yasmin defines a new style of building applications, based upon established technologies such as OOP [Booch91] [Meyer88], software components and novel concepts like cooperation [Grégoire94], delegation [Goldszmidt93] [Yemini91] [Johnson91b] and subcontracting [Hamilton93]. This is part of the effort to make computer software easy to use and develop in addition to overcoming typical problems that affect many applications, particularly in the network management field, such as being monolithic, hard to configure and extend. The experience gained by applying Yasmin to the are of network management allowed Yasmin to be refined and transformed into a more general architecture for open distributed software applications.

4.2. Conceiving Yasmin

The idea to design Yasmin [Deri97d] derived from several years of network management application development [Deri92] [Deri95b]. Despite the fact that many frameworks and architecture for building software applications are available on the market, most of them are tailored only for graphical user interface (GUI) development [Apple89] [Linton87] [Weinand88]. In the network management world, applications are usually built following the craftsman principle (i.e. everything has to be custom built for a certain task) without the adoption of application frameworks. Some companies have developed huge application systems that are composed of several applications and libraries that address every network management need. Although these systems are very powerful and rich in terms of functionality and tools, they do not address problems relating to application development. In fact in order to build network management applications based on those systems, developers need to know in detail many different libraries that have not always been designed to work together and that very seldom are based on OOP concepts. Additionally, due to the interdependency among those libraries, user applications require the installation of a large subset of the application system in order to run [Coad91]. The natural consequence is that applications are monolithic, difficult to tailor and configure and are system resources-hungry, preventing them from running on hosts of limited power. Beside this, network management applications quite often have to support different management protocols and object models other than being open to extensions and updates. Since network services should be available most of the time, it is necessary to identify mechanisms which allow applications to be selectively upgraded while they remain partially available in order to guarantee a minimal level of service. Additionally, applications must be built in such a way that it is possible to add new pieces when new hardware devices have to be supported or when users demand new services.

Yasmin attempts to address these issues and to overcome those problems mentioned before by defining an application framework characterised by the following properties:

1. light and simple kernel;
2. based upon pluggable software components;
3. built entirely on object-oriented technology;
4. extensible, easy to tailor and distribute.

The idea behind Yasmin is to build component-based applications that can be composed by the user, who can add or replace components at runtime [Marais96].

NOTE

Whereas other architectures [Smith92] allow component to be added at runtime, Yasmin has the unique capability to replace binary components at runtime. Note that dynamic languages allow the application code (hence the component code, if any) to be changed at runtime.

By enforcing the component boundaries, Yasmin prevents components from making assumptions on other components, hence reducing component inter-dependencies and making them easy to reuse on different contexts [Chen94] [Johnson88] [Johnson91a] [Meyer87]. Additionally, Yasmin loads the components on demand only when they are really needed and unloads them when no longer in use according to the policy defined by its developer. The efficient use of system resources is quite important because it enables complex applications to run on hosts of limited computation power like mobile computers.
An effective way to limit the application size is through cooperation [Helm90]. This is because components provide services available through a well-defined interface, that can be used by other components instead of re-implementing them in different flavours when needed. Often, large network management systems need to use a common set of services that sometimes require significant resources. A typical example are encoding/decoding (enc/dec) services needed to transmit information that are often replicated in different applications. As those services are often implemented using shared libraries, there is only one copy of the enc/dec logic in memory. Nevertheless this is not efficient enough because every time the enc/dec service is instantiated in the application, it needs a lot of memory for the allocation of the enc/dec tables. In this case, a single (multithreaded) enc/dec

that cooperated with all the local applications would be able to serve all of them without the need to replicate the services and hence require a larger amount of memory.

4.3. Yasmin at a Glance

Although the terms architecture and framework appear quite often in the literature, only in the last years the problem of formally describing them has been approached. The use of software architectures is pervasive in the informal diagrams and idioms that people use to describe system designs [Shaw94] [Shaw96]. The formalism used here to describe Yasmin the one proposed by Garlan and others [Abowd93] [Garlan93] [Garlan95] [Magee95] [Shaw95a] which at the moment seems to be the most adequate and complete one (for more information See "Architectures and Framework Basics" on page 13).

	Yasmin
Application Domain	Open distributed systems with emphasis on network management.
Architectural Goals	1. Genericity. 2. Enable scalability. 3. Promotion of reuse. 4. Support for runtime application evolution. 5. Enable the creation of slim ad efficient applications. 6. Technology independence. 7. Heterogeneity: allow different object models to be integrated.
Architectural Style	1. External: distributed, client-server model. 2. Internal: component-based, meshed architecture.
Genericity	Yasmin is not meant to support only a few specific protocols, but it is general enough to potentially support/integrate most of the protocols used in the context of open distributed systems.
Scalability	1. Multiple instantiations of the same Yasmin-based application. 2. Concurrency support. 3. Addition of new components at runtime.
Reuse	At component level.
Interoperability	1. Application level. 2. Component level.
Component Interaction	Message and event passing, procedure (service) call. All the interactions between components are mediated by the architecture (i.e. no direct component interaction).

Table 7. Yasmin at a Glance

The relevant architectural aspects of Yasmin are now discussed in detail.

Application Domain

Yasmin is tailored for the development of open distributed applications. Since the author is mainly involved in the management of open networked systems, Yasmin has been designed to address most of the issues present on that field (See "Thesis Requirements" on page 23). The most important ones are:

1. need to support different, heterogeneous network management standards and object models;
2. necessity to enable the integration of existing legacy code into new applications, in order to preserve the investments;
3. ability to be open to modifications and extensions, quite frequent in this field as the network technology changes;
4. scalability and exploitation of the distributed environment in order to achieve the performance demanded by the users.

Architectural Goals

The main architectural goals of Yasmin are [Johnson93]:

1. Genericity

   Yasmin has to be generic in the sense that it can be employed for building applications which belong to different areas in the context of open distributed systems. This is necessary since the investment in designing and adopting Yasmin has to be shared among different applications and projects. This also will enable reusability since different components which follow the Yasmin guidelines can be reused with very low effort [Karlsoon95].

2. Enable Scalability

   Since computer networks grow and change quite rapidly, it is mandatory to produce applications which can scale without having to rewrite them. For instance the number of network nodes in the last years has increased exponentially and the network management applications used so far, need now to handle and manage a much larger number of nodes than in the recent past.

3. Promote Reuse

   One of the key reasons for adopting one architecture is that it guarantees some basic properties. In open networked systems it is mandatory to promote reuse since the same component, slightly modified and adapted to the environment, can be successfully used in many different systems or in different environments without having to strongly modify if not rewrite it [IEEE94] [Krüger92].

4. Support for Runtime Application Evolution

   Often in distributed systems it is necessary to constantly guarantee a certain minimal level of service. When this does not happen, the consequences may not be limited to the local system. Suppose for instance that the host on which the DNS (Domain Name Server) runs is off-line for some hours. This prevents many users from accessing the network hence this situation is not acceptable. For this reason Yasmin allows applications components to be replaced and added while the application is running [Mätzel96]. This solution enables the selective update of relevant application parts without having to put the application off-line thus to totally disable the provided services.

5. Enable the Creation of Slim ad Efficient Applications

   Especially with the advent of mobile computing and downloadable applications, it is becoming necessary to produce applications which are able to run on environments of limited computing power and resources. In order to do this, Yasmin exploits mechanisms such as cooperation and delegation which prevent the replication of similar functionality and allow a simpler application structure to be designed. In fact, a application which strongly relies on cooperation is divided in small components which collaborate in order to obtain a global result (this is similar to what happens on RISC microprocessors). These components favourite the application tailoring and allow the application to load only the necessary components, hence to avoid wasting computing resources (a side effect is that the more computing resources are available for the system, the better is the global performance).

6. Technology Independence

   Yasmin does not have to be tighted to a specific OS or technology. It is very important that Yasmin-based applications can run on almost every operating system and that they do not rely on specific technologies such as database and frameworks. This is necessary in order to avoid tighting the architecture to other technologies while limiting the dependency on the OS. These dependencies may prevent running the Yasmin-based application on different machines running the same OS but configured differently. This limitation is not acceptable on computer networks where computers run different OSs and are configured differently depending on the user needs.

7. Heterogeneity

   As described in the previous chapter, relevant network management standards are quite different and heterogeneous. Since the integrated management of computer networks requires the support for (most) all of these protocols, then the architecture should promote the integration of different object models. Yasmin does this by defining a clean and sharp component interface which prevents components to expose peculiar properties which may influence the global application. For instance this happens when a component defines some peculiar datatype which then have to be included (in a C sense) by all the other components and not only by the ones which really need it (global datatype dependency).

Architectural Style

Yasmin's architectural style can be divided in two parts: external and internal style. Externally Yasmin is a distributed architecture which exploits the networked distributed environment and that it is based on the client-server model. A Yasmin-based application provides certain services accessible from the network. Applications can provide services, use services or do both.

Internally the application is divided in cooperative components which are disposed in a meshed architecture. Namely, the resulting component graph is a mesh where components cooperate by using/providing services from/to other components, either local or remote.

Genericity

In open systems the protocols which an application needs to support are not specified. Quite often it is necessary to write some proxy applications which interface different protocols or object models (see mediation functions, "OMG Network Management" on page 57). For this reason Yasmin has to potentially be able to support and use most of the protocols used on open systems. Specifically, Yasmin implements protocol supports inside its components (not at the application kernel), hence it allows future protocols to be integrated.

Scalability

Scalability is an important issue since on open systems applications are not static. This is because these applications periodically need to support new functionality as the network technology changes and to handle an increasing number of data which range from managed objects to network devices. Yasmin provides support for scalability in three ways:

1. Multiple Instantiations of the same Yasmin-based Application

   A way to increase global performance is through the exploitation of the distributed and networked environment. Since Yasmin-based applications provide collaboration and delegation support and are able to process remote requests, applications can delegate some activities to other remote applications which process requests on behalf of them. The more applications are available the better is the performance [Bernard89] [Lindenberg90]. Notice that this is an interesting way to use idle networked computers and old networked computers which are not able to deliver the performance needed by today's applications but which may be very well used for distributed computing [Sawitzki92] [Theimer89].

2. Concurrency Support

   Yasmin's components are reentrant making them suitable to process concurrent requests. Since Yasmin provides facilities for concurrency support, a Yasmin-based application can be concurrently used by different users or by the same user which issues concurrent requests [Löhr93]. Additionally with the advent of affordable multiprocessor computers, the concurrency support is becoming necessary to exploit these architectures.

3. Dynamic Application Extension by Adding New Components at Runtime

   Yasmin is based on software components which can be even added at runtime allowing application behaviour to be extended [Micallef88]. Given a problem, it is possible to implement support just for the core functionality adding further functionality later on. Notice that in computer networks is often quite difficult to estimate a problem size due to the rapid changing requirement.

Reuse

Yasmin facilitates reuse at component level, i.e. a component can be reused on different context [Shaw95b]. Since components can provide multiple services (see next section) it is possible to create a component which implements just one service hence to have a very fine granularity. Additionally, since services can be accessed from remote, in order to reuse Yasmin's compliant services it is not necessary to have the component which implement these services locally but it is possible to issue remote service calls. This way to reuse services and components it is preferable when the requester and the server hosts are running different operating systems, because it allows services to be reused without having to port the code which implements that service on the local operating system. Finally, notice that the porting process sometimes is not possible because, especially on network management, there are a lot of proprietary applications which provide services which are able to run only on a specific host/operating systems hence that cannot be ported to another platform.

Interoperability

Interoperability of Yasmin-based applications is at two levels:

1. Application Level

   At this level two applications can interoperate if the format of exchanged messages is the same. Since Yasmin does not imposes any restriction of the transport protocol and of the message format/content, this aspect of interoperability is totally delegated to developers. For instance Liaison (see next chapter), a Yasmin-based application, uses HTTP as transport protocol and also specifies the message format and its encoding. This guarantees that different instantiations of Liaison can communicate but it does not guarantee that they can interoperate because developers are responsible for the message contents.

2. Component Level

   Yasmin is responsible for component communications, namely it guarantees that a message/event sent to a component and its corresponding reply (if any) are delivered correctly. Developers are then responsible for the content of these messages/events.

Yasmin do not imposes any restrictions on the message format because this would impose some constraints (i.e. limited genericity). Nevertheless this approach leaves completely to developers the responsibility to format the messages properly since Yasmin just guarantees that the requests/ responses are delivered correctly.

Component Interaction

Components interact by message and event passing and by service call. As seen before, Yasmin does not define the semantic format of the communications. Instead, Yasmin mediates all the component interactions. This is necessary because it:

1. avoids direct component dependencies, necessary to replace components at runtime;
2. simplifies the component code since components are not responsible for locating other components or services.

Since Yasmin is a component-based architecture, it is reasonable to describe the architecture beginning with the description of Yasmin's components, named droplets.

4.4. Droplets

In the case of Yasmin, components can be regarded as the atom of the application. Yasmin components are typically faceless, i.e. without a visual appearance, and provide services to users and other components accessible both locally and remotely, under certain constraints which will be shown later. It is up to the developer to identify the granularity of the components. The smaller the granularity, the easier is the component reuse because the component provides a generic service. This approach has certain side effects however, because very generic services have to be composed in order to provide the service required, and a large number of active components may have a negative impact on the performance. Often, when an object-oriented language is used, a component can be defined in terms of a class. The main difference between a class and a component is felt when new classes are defined. A new class usually inherits from other classes and therefore specializes the parent class.

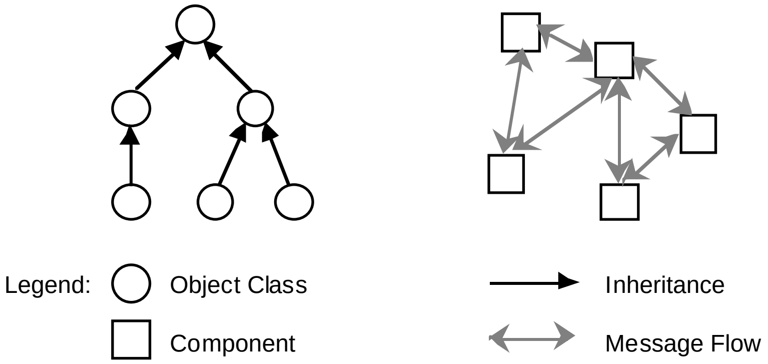


Figure 9. Object Classes vs. Components

A component, on the other hand, adds further services and functionality to the system, often exploiting existing services provided by other components. This is done by exchanging messages containing operation requests with other components. The core idea is that a component provides new services by collaborating with peers, whereas classes do this using inheritance. Such a statement does not mean, like in some object models (for instance COM), that inheritance is not supported. It means that, internally, a component can use inheritance to implement its logic but, whenever it has to interact with other entities, it does so by using the component interface and not by defining additional methods or class names. In other words the interface inheritance is supported but not the implementation inheritance. Therefore a new component provides new services by giving other components the opportunity to use its services through the component interface, whereas a new class provides new methods and a new starting point for further specialisation.

A *droplet* [Deri95c] is a software component having the following specific properties:

1. it is not statically linked to the application but it is loaded at runtime;
2. it has the ability to be replaced (i.e. a new version of the droplet can replace a previous one) at runtime while the application is running;

   NOTE

   Since droplets can be replaced at runtime, droplets are responsible for saving and restoring the persistent information they need to maintain. Typically, other droplets provide persistency facilities.

3. it has a well-defined interface, *droplet interface*, that makes it possible to communicate with other droplets independently of the type of the services provided;
4. it is reentrant hence the droplet can process concurrent requests.

NOTE

It is the droplet developer's responsibility to make sure that the droplet code is reentrant.

The term droplet, which does not have to be confused with MacPerl™ droplets, derives from the fact that in order to activate a droplet it is necessary to drop it into a certain directory monitored by the running application. The droplet interface (the component plugs) defines the provided services, the format of the messages that will be exchanged with the outside world, and additional information needed to load the droplet. Notice that whereas a class is an integral part of the application, a droplet requires an application in order to live even if the application can exist without the droplet. This is because the droplet adds functionality and services to the application but the application can exist independently of the number and the type of droplets.

4.4.1. Droplet Interface

A Yasmin-based application does not know at compilation time what services it will have to provide since the services are stored inside the droplets and not in the application. Due to this, such application has to be split into two parts: the core application, which provides the generic and the basic services, and the droplets which can provide additional services. For instance considering the case of a Web browser, the core services are: the user interface and the functions that handle the basic communication (e.g. TCP/IP socket creation, DNS lookup). Other services, which are part of each Web browser (for instance HTML display and FTP) could be implemented as droplets. The heterogeneous nature of droplet-provided services exactly identifies the boundary between the core application and the droplets. The core application is the most important part of the whole application because it is the part that never changes. Nevertheless it has to be general enough to accommodate current and future droplets.

Every routine that can be considered of public interest has to be added to the core application. Every access to data and services must be mediated by Yasmin, hence they can be accessed indirectly through the core application. A droplet should not be allowed to invoke functions directly or issue service requests provided by other droplets because this would introduce some droplet interdependency which are explicitly forbidden by the architecture. Every request has to be sent from the droplet to the core application that invokes the requested service on behalf of the droplet. This mechanism works based on the assumption that every communication has to pass through the core application because it is the only entity that knows how to locate and request services.

IMPORTANT

Suppose a droplet A want to invoke a service X provided by a droplet B. Event if the A would like to call X directly, it could not do it because the (linker) symbol for X has not been exported outside B due to the way droplets are built. Due to this X is not accessible directly by A, unless B access is on behalf of A.

Thanks to this mechanism, a droplet can use a service without knowing where it is really provided and it does not contain direct dependencies on other droplets. Either the core application or another droplet may even act as a proxy for another remote entity that provides the real service [Coplien95].

Method and Function Resolution

Before discussing how the droplet interface has to be implemented, it is necessary to understand how droplets can invoke external methods and functions not known at compilation time. This mechanism is called method/ function resolution and entails the step of determining which procedure have to be executed in response to a method invocation/function call. There are a few techniques for performing the resolution:

- offset resolution,
- name-lookup resolution,
- dispatch-function resolution.

The offset resolution is the fastest technique but also the most constrained because it requires that the name of the method/function to be invoked and, in the case of object-oriented programming, the name of the class that introduces the method be known at compile time.
The name-lookup resolution, similar to resolution techniques used by Objective C [Hook88] and Smalltalk [Goldberg83], is more flexible than the offset resolution and can be used when one of these conditions is verified:

- the method/function name is not known until runtime;
- the method is added to the class interface at runtime;
- the name of the class introducing the method is not known until runtime.

The dispatch-function resolution is the slowest and most flexible technique. It allows method resolution to be based on arbitrary rules associated with the class of which the receiving object is an instance.

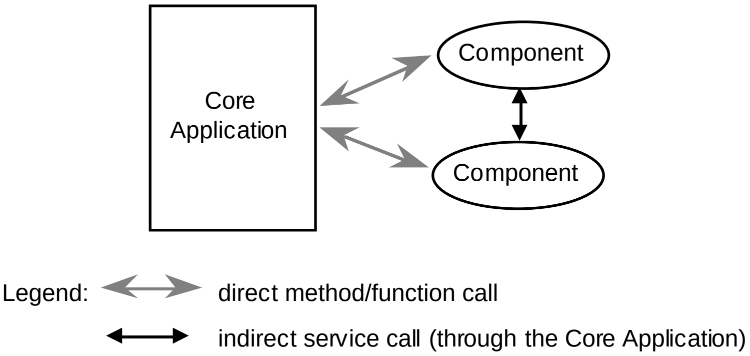

Core
Application
Component
Component
Legend:
direct method/function call
indirect service call (through the Core Application)

Figure 10. Direct and Indirect Calls

The droplet issues direct (offset resolution based) and indirect (name-lookup resolution based) calls. The direct call is used when the droplet invokes functions provided by the core application. A droplet is a sort of small application linked at runtime with the core application.

NOTE

A droplet, usually implemented using shared libraries, is actually loaded and not linked at runtime. Nevertheless, since the core application provides some basic services which the droplet may use, when the droplet is linked it has to be linked with a shared library which contains the symbols of the services provided by the core application. The linked droplet, will not include the code of the services contained in the core application but just the reference to those symbols. These references are then resolved dynamically by the loader when the droplet is loaded. Due to this, a droplet is then both loaded and linked at runtime.

Droplets can use functions/methods, access objects contained in the core application and issue service calls. When a droplet calls a core application provided function/method, the offset resolution technique is used because the name of this function/method is known at compile time, and the linker can resolve these references based upon such information. If a droplet requests a service that is not provided by the core application, the indirect call is used. In this case the droplet invokes a core application function/method that takes the service name and the parameters as input (See "Service Manager" on page 96). Then the core application function/method performs the name-lookup and, if the requested service has been found, it calls the requested service and returns the results of the call to the droplet that issued the request. In practice, a service can be used by a droplet not only if the service exists but also if the droplet has access to the service. Finally note that it does not make too much sense to allow droplets to use all the available services without enforcing some basic security rules: the droplet must have the necessary credentials and access rights in order to access critical or privileged services.

IMPORTANT

All the code fragments are written using the C++ language [Ellis90] [Stroustrup91]. Because droplets do not rely on specific C++ features, they can be written using whathever programming language ([Deri95e] covers droplet implementation using Java).

The following example shows how a droplet calls an external service, i.e. a service provided by another local droplet, named ABC:

```
int returnCode;
ABC_InterfaceType ctrBlk;

ctrBlk.input  = <input params>;
ctrBlk.output = <output params>;
ctrBlk.rc     = &returnCode;

switch(CallLocalService("ABC", (void*)&ctrBlk, NULL)) {
  case SERVICE_NOT_AVAILABLE: // Service not available
    cerr << "The service ABC is not available" << endl;
    break;
  case SERVICE_ERROR: // Service call returned an error (check returnCode)
    cerr << "The service ABC returned an error" << endl;
    break;
  case NO_ERROR: // Service executed successfully
    cerr << "Done." << endl;
    break;
}
```

Figure 11. External Service Call

The name-lookup resolution technique is used in the droplet-based application because method dispatching has to be done at runtime for two reasons:

- it is not possible to statically link the droplet to the core application because the number and the type of the droplets are not known at compile time;
- the indirect call technique decouples the droplets and makes them independent of the existence of other droplet-provided services.

Thus the indirect dispatch is of primary importance because it makes the application independent of the location and the existence of services.
Implementing the droplets using shared libraries does not have much effect on the application speed of a monolithic application. In fact in the latter case the compiler and the linker resolve the references. In a Yasmin-based application the resolution is performed at runtime by the indirect dispatch mechanism. Every time a droplet or a service is invoked, the core application has to find the corresponding function (this search may fail if the function does not exist). This lookup has an impact on the performance but the slow-down effect can be minimized if caching or an efficient search mechanism such as hash tables is used.

NOTE

The author has not investigated in detail how to efficiently solve the problem besides using basic cache and has tables. The Self language uses some very interesting techniques [Ungar87] that can be employed to significantly decrease the lookup time.

Notice that when a virtual class or method is used in an object-oriented programming language a similar lookup is performed. This analogy illustrates how the droplet technology is similar to the object-oriented technology and how in some ways both techniques share similar problems.

Understanding the Droplet Interface

The *object interface* is the set of methods to which the object responds. The method name and its signature are defined at compilation time and cannot be changed at runtime. Hence the only way to add new methods or to modify existing ones is to shut down the application, modify it, recompile it and start it again.

The *droplet interface* defines the boundary between the core application and the droplets. Unlike the object interface, the droplet interface is well defined and it is the same for every droplet independently of the provided services. It also has to be general enough in order to accommodate different and heterogeneous services. The difference between an object interface and a droplet interface can be emphasized by means of an example. Suppose we have a droplet with this interface:

```
class Class {
public:
 int PerformAction(char *serviceName, char **returnValue);
}
```

Figure 12. Simple Droplet Interface

Imagine that the droplet recognises only the “Apple” and “Pear” services, and that we have to extend it by adding a service called “Orange”. The way to do this, is to add the functions/methods needed to handle it. When the `serviceName` parameter has the value “Orange”, this new logic is used. Thus a droplet does not redefine its interface but releases some constraints on the parameters by accepting a wider set of values. Since the interface is untouched, the application which uses the droplet does not have to be recompiled but just the droplet whose new version can be reloaded by the running application [Ban95]. The example seen before can be modified as follows in the case of a class:

```
class ParentClass {                  class Class: public ParentClass {
public:                              public:
  int Apple();                         int Orange();
  int Pear();                        };
};
```

Figure 13. Class Definition

Subclassing is necessary because the `ParentClass` class may have been shipped as a library and thus it cannot be modified but only subclassed. This prevents the new class and the new methods from being used by a class that is not derived from `ParentClass`. Also in order to make the new service available, the application has to be recompiled and restarted, whereas the droplet-based one has simply to reload the droplet.

NOTE

Implementation issues concerning droplet reload are not relevant for this work since they vary from platform to platform. Since droplets have no cross dependencies, they can be loaded/unloaded at runtime without affecting other droplets. Due to this, droplet load/reload is a pure programming exercise and it does not present any problem. What is not possible to do, is to change the implementation of a shared library used by some droplets and pretend that the system automatically reloads the involved droplets. This limitation is due to the fact that the system knows only about droplets and not about the libraries loaded by such droplets. In any case this is a secondary limitation. More information about this topic can be found in [Crelier94].

It is worth mentioning that a droplet can be written either in a object-oriented language or not. Thus the difference between droplets and classes is that the droplet adds new services by releasing limitations on the values of the interface parameters (the interface being unchanged), whereas a class provides new functionality by subclassing and defining new methods. This difference is shown in the following example:

```
int Class::dropletInterface(char *serviceName, char &returnValue)
  {
    if(strcmp(serviceName, "Child") == 0)
       strcpy(&returnValue, GetChildName());
    else if(strcmp(serviceName, "Father") == 0)
       strcpy(&returnValue, GetFatherName());
    else
       return SERVICE_NOT_HANDLED; // Service not handled by this droplet

    return NO_ERROR; // No error
}
```

Figure 14. Droplet: Service Handling

In any case, the droplet paradigm does not have to be considered a class competitor. It has been shown that both paradigms can fit together and that, thanks to this, the droplet paradigm can be introduced into existing applications written in an object-oriented language like a transition path towards a pure compound application.

Droplet Interface Definition

A droplet is not statically linked to the core application but is loaded at runtime. Therefore a mechanism to load a droplet dynamically and to bind it to the core application has to be identified. Many operating systems provide facilities for the creation and use of dynamically bound shared libraries. Dynamic binding allows external symbols referenced in user code and defined in a shared library to be resolved by the loader at runtime. The shared library code is not present in the executable image on disk. Shared code is loaded into memory once, in the shared library segment and shared by all processes that reference it. Considering the facilities offered by a shared library and its great versatility, it makes sense to use it for droplet implementation.

A droplet is seen by the core application as a shared library. The core application can load droplets on demand or at start-up time. The entry point of a shared library can either be a function/method or a variable. Given the entry point, the core application has to be able to access all the services provided by the droplet. It is therefore necessary to define a new type or class that contains all the information about the droplet and to define a static variable in the droplet code containing this information. Such a variable will be the entry point of the library. This is the basic information that can be specified by a droplet:

```
typedef void(*DropletInitTermFunct)(<function parameters>);
typedef void(*DropletInterface)(<interface parameters>);

typedef struct {
  short version, loadOnDemand;
  unsigned long  lifetime; /* in seconds */
  char *dropletName, *dropletInfo;
  void* additionalInfo;
  DropletInitTermFunct startFunct, endFunct;
  DropletInterface toolFunct;
  LocalService *localServices;
```

```
   RemoteService *remServices;
 } DropletInfo;
```

Figure 15. Droplet Information

The `DropletInterface` is defined as a function but it can be also defined as a static method. The parameters are not specified because they can vary from application to application, however each application defines a droplet signature that will be used for all the droplets. The `version` field (major and minor number) is useful to check the version in order to prevent inconsistencies between droplet versions. The `loadOnDemand` field (boolean) specifies whether the droplet can be loaded on demand or if it must be loaded at start-up time, this mostly for performance reasons since some droplets may take some time to initialise. The `lifetime` field specifies for how long the droplet can stay in memory since the last time it has been used (See "Droplet Manager" on page 93). The `dropletName` and `dropletInfo` fields are used to pass the droplet-related information to the core application. The `dropletName` is necessary to find droplets given their name, and the `dropletInfo` to characterise the droplet by specifying what the droplet does and how it expects to be invoked (the technique of putting comment information into the item itself has already been applied in languages such as SmallTalk). The `additionalInfo` field contains some (optional) private droplet information in an unspecified form. The core application is not responsible for this information but it is the droplet itself that has to manage it. The `startFunct`, `toolFunct` and `endFunct` are pointers to droplet functions. The `startFunct` and `endFunct` are optional (they can be set to `NULL`) and when present they are called when the droplet is loaded/unloaded. The `toolFunct` identifies the core droplet function. It is invoked by the core application when an operation concerning the droplet has been requested. The ability of the core application to load the droplet at runtime permits droplets to be loaded on demand. This technique can be used to save system resources (for instance memory) and to make distributed applications more flexible. For example a remote application can provide a service, implemented by a droplet, and used frequently by a local application. In order to reduce the network traffic, it makes sense to copy the droplet from the remote host to the local host and to attach it to the local application. Due to this, each time this service is requested, a local request is issued instead of a remote one. Similar techniques can be used in many other cases where the performance, the network traffic or the resources in use have to be optimised.

Heterogeneous Code Integration

Besides advantages relative to application extensibility and runtime behaviour modification, the use of droplets has another important advantage.

The integration of legacy code into a new application creates a lot of problems. Namely:

- integration of legacy code coming from different sources usually imposes different programming styles or the use of different programming languages;
- since legacy code is often available only in binary form, the developer does not have control over it hence this code may clash with other parts of the application being developed.

The first problem is more a programming problem. Mixing different programming styles into an application is not a good programming technique since this makes the code more difficult to understand and also creates several problems if the code to be integrated is not written using a single programming language.
The second problem is more subtle and difficult to track and solve. This can be clarified with an example. Suppose having a library which contains the following code fragment:

```
// The following class is used internally by GetMainLibraryService
class SimpleClass {
public:
  SimpleClass();
  ~SimpleClass();
};

int numItems;
char* GetMainLibraryService() { ... };

SimpleClass()::SimpleClass()  { ... };
SimpleClass()::~SimpleClass() { ... };
```

Figure 16. Example of Legacy Code

Suppose also that the application needs to use the `GetMainLibraryService` function. This requires the application to be linked with the library. In case the main application contains a class named `SimpleClass` there will be a link problem because the linker symbols for `SimpleClass` are duplicated since the same symbols are present in both the application and the library. One way to solve this problem is to modify the application and to rename the `SimpleClass`, although this might be a problem in case the application is large or when the class name cannot be changed for other reasons.
A more subtle problem can be caused by the `numItems` variable. If the application defines a variable with the same name (the type does not have to be necessarily the same since the compiler will always generate a symbol named `numItems` independently from its type) at link time there will be a symbol clash hence one of the two symbols will be overwritten (notice that smart linkers may highlight the problem) with the result that the application will interfere with the library and vice-versa.

This simple example demonstrates that the integration of legacy code may cause several unexpected problems. The use of droplets prevents all these problems with no effort. In the example above, the developer will implement a droplet which has as `toolFunct` (see Figure 15, “Droplet Information,” on page 87) the `GetMainLibraryService` function and then will link only such droplet with library. In case the application or another droplet define some symbols as the ones defined in the library, there are no side effects since the application or the droplet will not be linked with the library because they do not use the `GetMainLibraryService` function.

The droplet ability to prevent the problem seen above, it is very useful because the integration of legacy code into a newly developed application is quite difficult to solve when the application is monolithic since there is a single final link step instead of several link steps, one for each droplet. A side advantage of having a link step for each droplet is related to the link time. Since a droplet is much smaller that the whole application, the time necessary to link a droplet vs. the time necessary to link the monolithic application is much smaller. Additionally a modification in a droplet does not require the whole application to be relinked but just the modified droplet.

4.4.2. The Service Interface

Droplets should not be regarded as mere parts of an application. They are separate entities that provide and use services. In order to specify the services provided by a droplet, the droplet interface includes the service interface (`localServices` and `remServices`), which specifies respectively the services accessible only locally and remotely. This interface, like the droplet interface, masks how the service is implemented and other details that need not be of public interest, just like a class of an object-oriented language does. The interface is specified as follows:

```
typedef int(*LocalServiceFunction)(void* in_data, void** out_data);
typedef void(*RemoteServiceFunction)(char* in_data, char** out_data);

typedef struct {
  LocalServiceFunction localServPtr;
  char *servName,      /* Name of the service         */
       *servInfo,      /* Info about the service      */
       *servInParam,   /* Info about input parameter  */
       *servOutParam,  /* Info about output parameter */
       *servRetValue;  /* Info about return value     */
} LocalService;

typedef struct {
  RemoteServiceFunction remoteServPtr;
  char *servName,      /* Name of the service         */
       *servInfo,      /* Info about the service      */
       *servInParam,   /* Input parameter format      */
       *servOutParam;  /* Output parameter format     */
} RemoteService;
```

Figure 17. Service Interface

The string `localServPtr/remoteServPtr` are the pointers to the function that provide the local/remote service. It is very important that this function have a well-defined and general interface like the one shown above. There are other possibilities to define `LocalServiceFunction` such as imposing no constraints on the number and type of parameters instead of using a `void*` which can be used to pass every datatype. This requires use of a variable argument list that may reduce robustness due to the limited checks that can be performed on the parameters. The `servName` identifies the name of the service. To improve the lookup speed, however, a numerical index can be added. The other fields are used to provide additional information about the service: what it does, which input/output parameters it expects, and how to interpret the return value.

IMPORTANT

In the case of `LocalServiceFunction` the type of the input/output parameters is undefined (i.e. every type can be specified), whereas in the case of `RemoteServiceFunction` their type is `char*` and it is developer's responsibility to marshall/unmarshall the information, i.e. to pack/unpack the information in/from a stream of bytes.

These fields are very important whenever a droplet has to call services provided by other droplets and whenever it has to find out how it is supposed to issue requests. They do not have to be identified as information of secondary importance but they have to be considered part of the dynamic service call interface. For example suppose that an application displays a pop-up menu containing the names of all the external tools such as circle, line, or rectangle. The application can exploit `servName` to add the tool names to the menu and `servInfo` to draw an icon for each entry. Note that certain services may be available only when specific conditions are verified (e.g. only when some resources are available). For this reason the core application should offer a way to register/deregister services dynamically other than via `DropletInfo`. Services are identified with a precise name just like for methods. Nevertheless it is important to include a description of the service. It has to be remembered that droplets may have been written by different persons for very different purposes and therefore may not have a description, so these droplets are useless for anybody but the droplet-author.

4.4.3. Comparison with Other Software Components

In the previous chapter (see “Component-based Architectures” on page 33 and “Plug-in Components” on page 49) state-of-the-art component-based architectures and software components have been discussed and compared. It is interesting to see how droplets compare with them. The following table lists some of the major differences between the various component types according to certain key criteria [IBM94b]:

	OpenDoc	OLE	Java Beans	QTCM[a]	Droplets
Application Domain	Generic and document-centric applications.	Generic applications.	Generic applications (usually with a GUI).	Plug-in architecture for extensible applications.	Generic, modular, dynamically-extensible applications.
Programming methodology[b]	Object-Oriented.	Object-based.	Object-Oriented.	Object-based.	Object-based.
Relationship to Languages	Language-neutral.	Language-neutral, function pointer table based user-built.	Written on Java.	C/Pascal.	Language-neutral, function pointer table based user-built.
Remote Component Interaction	No.	Yes[c].	No.	No.	Yes.
Method Resolution Mechanisms	Automatic, three forms are supported: 1. offset, 2. name, 3. dispatch.	User responsibility: 1. Use QueryInterface to get the Interface. 2. Indirect function call via interface vector table.	Automatic, provided by Java.	Indirect function call via interface vector table.	Automatic: indirect function call via droplet interface vector table. Note that being the droplet interface object-based, components can only invoke functions and not methods.

	OpenDoc	OLE	Java Beans	QTCM[a]	Droplets
Polymorphism Support	Yes. Granularity: 1. interface level; 2. method level; 3. user-defined.	No. It must be emulated by user-written code.	Yes, provided by Java.	No.	No. It must be emulated by user-written code. If droplets are specified using an object-oriented language, all the mechanisms offered by the language can be used.
Inheritance Support	Yes: 1. single, 2. multiple, 3. abstract (interface) inheritance.	No. Instead, containment and aggregation (manually) are offered.	Yes, provided by Java.	No.	No. Instead, cooperation and aggregation (manually) are offered.
Component Interface Definition Language	SOM IDL based on OMG IDL.	None.	None.	None.	None.
Component Interface	Statically typed.	Statically typed. Developers can choose at runtime the interface they intend to use.	Statically typed.	Statically typed.	Statically typed (Droplet Interface).
Component Interaction	Events, method call.	Events, procedure call.	Events, method call.	Indirect procedure call through the component manager.	Events (single and multicast), service call (local and remote).
Encapsulation Boundary	Object Level.	Interface Level.	Interface Level.	Interface Level.	Interface Level.
Active Components	No.	No.	Yes.	No.	Yes.
Concurrent Objects	Not specified (usually it has to be emulated by user-written code).	Not specified (usually it has to be emulated by user-written code).	Yes.	No.	Yes.

a. This column covers the QuickTime Component Manager and the other type of components, similar to QTCM, covered in the section “Plug-in Software Components” on page 48.

b. According to [Wegner87].

c. Exclusively with DCOM, available only on WindowsNT release 4.

From the previous table it is possible to see some similarities between droplets and other component types. Namely, droplet are similar to:

- OLE objects at the interface level in the way components interact, both in a local or remote fashion;
- Java Beans because both component types allow components to be active entities and support concurrency;

- QTCM because both component types allow applications to be extended at runtime simply by placing components in a well-specified directory.

A unique droplet characteristic is its ability to be added and replaced at runtime while the application is running.

At a glance, droplets may seem to be quite primitive because they are object-based. From another perspective, this feature allows them to be relatively simple yet quite powerful. The cost of implementing components in a full object-oriented fashion like in OpenDoc is certainly convenient for component writers but it fully ties components to OpenDoc. This means that such components can only run on those platforms where OpenDoc is available. Moreover it prevents the reuse of existing components in non-OpenDoc applications. This is a major drawback especially on computer networks where computers are very heterogeneous. Even if two computers run the same OS there is no guarantee that both have the same version and the same components. One of the design goals of the droplets has been to make them as OS/technology-independent as possible in order to achieve portability while including only the basic features each network application needs in order to keep them simple and light.

4.5. Yasmin Components

The following picture depicts Yasmin's components.

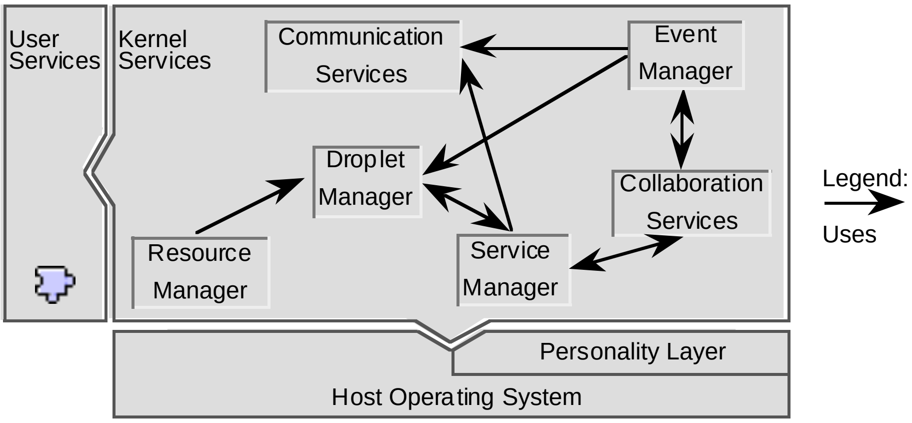


Figure 18. Yasmin's Components

User services are implemented using droplets, which change from application to application. Kernel services instead, are part of each Yasmin-based application and provide services, and functionality on which droplets rely. The following sections cover each kernel component in detail.

4.5.1. Personality Abstraction Layer

The kernel services are designed to run on different operating systems. For this reason it is necessary to abstract the operating system through a thin layer called a personality.

NOTE
The term personality has been borrowed from Mach kernel [Accetta86] terminology. In that case, the kernel provides some basic services on top of which different operating systems or different instances of the same operating system, called personalities, can run.

Personalities provide abstractions for low-level services performed by the host operating system. Those services include, but are not limited to:

- concurrent execution (threads) [Atkinson91];
- synchronisation primitives (semaphores);
- loading/unloading of libraries and metadata information;
- interprocess communication.

If a potential operating system does not support all of these capabilities, it is often possible to run Yasmin-based applications too. For instance an application can run in single-threaded mode if threads and semaphores are not supported. The use of personalities allows a single source code tree for different operating systems to be maintained, whereas the personality contains operating-system-dependent code. This facilitates porting code on different operating systems and to maintain the different code versions. In addition it prevents having to change the entire application but just this component when different releases of the underlying operating system offer new or more performant functionality.

4.5.2. Droplet Manager

The droplet manager (DM) is responsible for handling droplets. Namely it:

- loads droplets on demand;
- maintains a droplet reference counter that allows droplets to be purged;
- is responsible for detecting new droplet versions or further droplets added at runtime;
- collaborates with the service manager, informing it of newly available services.

A Yasmin-based application stores droplets in a well-defined directory (usually named `Droplets/`). The user puts (or drops, if drag and drop is used) droplets in such a directory and is free to replace them during program execution, hence the term droplet. The DM at start-up time searches for a file called `index`, which contains the name of the droplets and the services they implement. If such a file is found, the DM verifies that it is newer than all the other droplets (this is done by checking the file modification time) to ensure that it is consistent with the current droplet set. If not, or if there is a new droplet, the index file is rebuilt automatically without any external intervention. This operation, similar to the registration of OpenDoc part editors, is done by loading and unloading in sequence each droplet in order to build the list of droplets available and of the services provided by the droplets that are not visible at the file system level because they are usually stored inside the droplet itself.

Inside each droplet the droplet lifetime is specified. The lifetime, which ranges from one second to infinity, specifies how long the droplet has to be kept in memory since the last time it was used. This facility is used to avoid keeping in memory droplets that are no longer needed. It indicates to the system when to purge some resources in case of low memory conditions. If the droplet's lifetime has expired, the DM unloads it and releases all the memory and resources allocated by the droplet by calling the droplet termination function.

The DM is also responsible for detecting new droplets and new versions of them. In this case the `index` file is updated and the service manager (SM - See "Service Manager" on page 96), responsible for handling the services, is notified of the new services available and the ones no longer available (this happens when a new version of a certain droplet does not implement all the services implemented by the former version). Moreover, droplet versioning prevents the system from loading and using droplets that have been developed for a different application version, which may introduce problems or spurious errors. Similar to the versioning system used by IBM DSOM, droplets have a minor and a major version number. The major version number specifies which application version can use a given droplet version, whereas the minor number is used to implement the droplet versioning.

It is worth noting that droplets cannot be unloaded when in use, whereas it is possible to have one or more different versions of the same droplet active at the same time. This technique works because the DM is the *only* entity that maintains direct references (i.e. pointers) to droplets, whereas other entities such as the service manager simply access services through the DM. For this reason, whenever there is a new droplet version, the DM loads it without checking whether someone is still using the old version. If the old droplet version was no longer in use, then that version is unloaded and the new version is loaded, otherwise the DM flips the pointer to the droplet vector table. This operation:

- allows the new droplet to be used whenever a new request for such a droplet has to be processed;
- prevents new requests from being processed with the old droplet while the operations in progress (that make use of the old droplet) can continue;
- allows the old droplet to be unloaded whenever all the operations involving the old droplet have been completed.

NOTE

Although replacing components on demand seems to be an easy job, this is possible if and only if the access to the components is mediated by a single manager (the DM in the case of droplets) which intercepts all the accesses to the component. The lack of this manager is the reason why other component-based architectures such as OpenDoc do not allow components to be replaced at runtime.

The DM collaborates with the SM in order to guarantee this behaviour by keeping track of the requests currently in progress for each droplet. A request is considered "in progress" from the time the SM issues a new service request to the DM until the SM notifies the DM that the request has been completed. This mechanism works because services and droplets are accessed only through managers which shield them from the rest of the system. Access to resources and services exclusively through managers:

- contributes to the global system robustness;
- prevents droplets from being directly dependent on each other, i.e. by means of direct function or method calls;
- allows droplets to be selectively plugged and unplugged at any time because they have no cross dependencies of any type.

4.5.3. Event Manager

Events are by nature asynchronous and usually indicate that something has occurred. Typical events are mouseUp/mouseDown or network events. Normally, events are processed when they occur and their type is well defined so the program can handle them. In Yasmin, these limitations are relaxed, hence:

- events can specify when they have to be processed;
- droplets can define new event types.

Yasmin represents events as information records containing the type of the event (`event type`), when it has occurred (`event time`), and additional information relative to the event itself (`event info`). Yasmin adds a new field to this record which specifies when the event has to be processed; it may contain an absolute time or a displacement with respect to the time the event occurred. If this field is set, the event is called a *delayed event*. A delayed event is used to implement periodic tasks and activities that have to be performed at a certain time. Typical examples are chime clock events that have to be executed every hour or system backups that have to be performed every Sunday at 1am when virtually nobody is expected to be using the system.

The Event Manager (EM) is responsible for:

- delivering events to the various components,
- handling delayed events,
- allowing different droplets and services to cooperate and interact by means of the events they exchange.

Yasmin defines a set of basic event types and allows droplets to define their own custom event types, specified inside the droplet itself. Hence, whenever a new droplet is loaded/unloaded, the DM notifies the EM about the event types that can be handled. As different droplets can handle the same event type, a string called `eventDestination` specified inside the event record is used to identify the type of the received event. Such a string can have three values: a droplet name, a star (“*”), or a null value. In the first case the event is delivered to the specified droplet, in the second to all the droplets that handle such an event, in the last to a droplet that handles the event, if any. If an event cannot be handled, it is discarded and the memory used by the event is freed.

Yasmin’s event flexibility allows droplet intercommunication to be implemented easily and clean-way. Delayed events facilitate the implementation of periodical tasks whereas custom events allow different droplets to interoperate in an appropriate way to send a specific event for a certain situation instead of using generic ones that must be jeopardised in order to express peculiar situations. Additionally, the event destination enables droplets to implement a multi/broadcast facility, which is useful when multiple droplets have to be informed of a certain event that is important for all of them (for instance a `resourcePurge` event, which is broadcast by the system in low memory situations).

4.5.4. Service Manager

The Service Manager (SM) interacts with the DM to handle the services provided by the droplets. When a droplet is loaded, the DM notifies the SM of the services provided by the droplet, which are made available to the entire system. When a droplet is unloaded, the DM informs the SM of the services that are no longer available. Whenever the SM receives a request for a certain service, it request the DM to return a reference to the requested service. The DM, according to the information contained in the `index` file, identifies the droplet which implements the service, loads it on demand if necessary, and returns a droplet reference to the SM. If there is no droplet which implements the requested service, the request is rejected by the SM.

Services, identified by a unique string, can be of two types: local or remote. A local service can be used only locally whereas a remote service can be used both locally and remotely in an RPC-like way [Birrell84] by exploiting the Communication Services (CS - See “Communication Services” on page 98). The main difference between local and remote services is that for remote services, the input/output parameters are both strings (hence the service is responsible for marshalling/unmarshalling data) whereas local services can use any type they want. Local services are requested using `CallLocalService`, whereas remote ones using `CallRemoteService`. Each service is specified by an entry inside the droplet information record. Such entries contain the name of the service, information about the service and about the input and output parameters specified as strings using the C language convention. Remote services contain `char*` in both input and output parameters whereas local services parameter contain the real type. For instance a local service that takes as input a record containing the name and the age of a person has a service input parameter that looks like `char*, unsigned short`. Service parameters are mandatory and are useful for developers whenever they want to access services provided by a droplet written by third parties.

NOTE

In a future Yasmin revision, service parameters (see Figure 17, “Service Interface,” on page 89) will be used to do transparent marshalling/unmarshalling, allowing just one type of service accessible both locally and remotely.

When a droplet has to invoke a service request, it cannot call the service directly, thus the SM does this on its behalf. The droplet provides the service name and the input parameters to the SM, and it then receives from the SM the result of the service invocation or an error if the service cannot be found. This design choice derives once more from the plug ‘n play principle, which specifies that information access cannot be gained directly (e.g. through a pointer) but has to be mediated by the entity responsible for managing such information.

In Yasmin, service requests must be processed within a limited amount of time in order to leave processing time to other requests. This means that a service request has to terminate and it cannot last for an infinite amount of time (e.g. an endless loop). Services that may take a long time to process requests (for instance they may need some resources not yet available) should be divided into subservices that are activated sequentially by means of events. The requirement to complete service requests in a finite amount of time is very important in single-threaded systems, because the entire system is blocked until the service has been completed. In a multi-threaded system, although the system can continue to work, long lasting services occupy resources (for instance threads) and hence reduce the global application performance. The SM has no way to guarantee that services do not take too long because programmers are responsible for this. The only way for the SM to prevent application deadlock or wild resource usage is to run a sort of garbage collector that kills the threads that are apparently in an infinite loop or that have been running too long. Although this solution is not very clean because resources in use by threads may not be freed when the thread is killed, the SM has no other choice to guarantee a minimal quality of service and to prevent application deadlock.

4.5.5. Resource Manager

The Resource Manager (RM) cooperates with other managers to use system resources efficiently. Such resources include but are not limited to memory, communication sockets, and droplets. The RM makes sure that system resources are not wasted and that like a kind of garbage collector is activated periodically to purge resources no longer needed. The RM:

- informs the DM when a droplet’s lifetime has expired so it can be unloaded;
- makes sure that threads are used efficiently by not starting too many threads, which would decrease the overall application performance;
- is responsible for purging memory and other system resources (including droplets) periodically or when it is required to perform a certain task and the available resources like MacApp™ [Apple89] does.

Although the RM is a hidden component, it is very important because it allows the system to run with very limited resources and prevents wasting them. For instance, Liaison (cfr. next chapter), a Yasmin-based network management application, can perform complex network management tasks using a very small amount of memory because the RM contributes to scale-down network management applications from large hosts to standard machines.

4.5.6. Collaboration Services

CollaBoration Services (CBS) enable droplets and services to communicate and cooperate in order to perform a certain task [Nierstrasz90]. CBS, exploiting SM and EM, allow a task to be divided into many small cooperative subtasks. This solution enhances performance because these subtasks can be performed concurrently. This helps keep complexity low because each subtask is simpler than the original task. CBS provide facilities for:

- sending requests in multicast/broadcast mode and collecting results;
- synchronising tasks by means of events.

It is worth noting that Yasmin implements collaboration not only by means of CBS but also through the SM. In fact droplets collaborate with the rest of the system by providing services that can be of general interest. This avoids services being replicated, which saves development time and keeps the system slim and efficient.

4.5.7. Communication Services

Communication Services (CS) allow Yasmin-based applications to communicate with remote entities (local communications are performed by means of events). As Yasmin has been designed with portability in mind, external communications should be based on well widespread protocols such as TCP/IP or HTTP. Developers use the CS in order to:

- register/deregister communication sockets,
- be notified when data is available,
- issue requests and retrieve results.

NOTE

A communication socket is used to identify a communication channel which allows the application to communicate with external peers. The term socket does not necessarily mean that Unix BSD sockets [Stevens90] must be used.

CS are also used internally by other architecture components such as the SM, which uses it to transparently send remote service requests and to receive responses. In fact an important characteristic of CS is that they allow one to send data in a reliable way and to transparently handle socket and protocol errors, shielding droplets from differences among socket implementations available on various platforms.

4.6. Yasmin’s Design Choices

The Yasmin’s design has been an iterative work. Some architecture components have been added and others have been significantly redesigned while developing the architecture. This section highlights some design choices and it discusses possible alternatives.

Droplets

The droplet design has influenced the whole architecture. It was clear since the beginning of this work that Yasmin needed to be a droplet-centred architecture. The reason for this is very simple: the use of programming techniques such as inheritance or polymorphism used to glue different pieces of an application, has often produced very monolithic applications. Then I decided to implement the application using software components able to be replaced and added to a running application. The various alternatives could

have been CORBA, OLE, or the use of languages such as Objective C or SmallTalk. I decided to drop CORBA and OLE because their components are not replaceable at runtime, and also because the disadvantages are predominant with respect to the advantages (complexity, cost, platform-dependence). I have also dropped Objective C because is not available on all platforms I was interested in. I decided not to use SmallTalk because all the SmallTalk implementations I have been able to use were not able to deliver the performance needed in network management, not mentioning other important issues such as memory use and portability.
I decided to design the droplet interface in a ‘C’ fashion after having thought for a while to implement it in an object-oriented language such as C++ in order to manipulate droplets as pure objects. I decided not to use C++ because:

1. Due to they way C++ compilers generate symbols, the integration of droplets developed using two different compilers presented some problems because the dynamic linker was not always able to resolve all the references.
2. The number and the name of services provided by a droplet is not fixed hence each droplet should have a different interface. For instance:

```
class InterfaceDroplet_A {
public:
  int Service_A1();
  int Service_A2();
};
```

```
class InterfaceDroplet_B {
public:
  int Service_B1();
  int Service_B2();
  int Service_B3();
 };
```

Table 8. Some C++ Droplet Interfaces

This solution has been discarded because C++ does not offer a standard way of getting information about classes (reflection), hence about the name and the number of services.

3. Because droplets are independent entities, a droplet must not have direct references to other droplets. For instance if a droplet A needs to call the service `Service_B1()` as defined in the previous table, it should contain the call `Droplet_B_InterfacePtr->Service_B1()`. Therefore droplet A should make assumptions on the droplet interface of B. This prevents for instance droplet B from being modified without having also to modify droplet A.

NOTE

The adapter pattern [Gamma94] cannot be applied to the problem just described, because such patter needs to know the interfaces at compile time.

In conclusion, the definition of the droplet interface using an object-oriented language presents many drawbacks. Therefore I decided to model droplets as objects, so they look exactly as objects with the difference that internally they have to deal with a non object-oriented interfaces hence with functions instead of methods. This solution prevents mechanisms such as inheritance and polymorphism among droplets to be used. Nevertheless this does not have to be considered a limitation because those mechanisms are incompatible with the dynamic nature of the droplets.

Slim Application Kernel

Since quite some time there is a debate concerning what has to be included in an operating system kernel, and what has to be implemented in the user space. Some people say that a microkernel has advantages over a fat kernel because once the kernel has been ported to an hardware platform then it is quite simple to port the rest of the OS. Additionally a microkernel can be optimised more easily that a fat kernel hence the performance degradation due to the several calls to kernel functions can be balanced by a better kernel performance.

Similar considerations can be applied to Yasmin. Yasmin has been designed to produce applications both portable and performant. The problem about what to include and what not to include in the application kernel came into the arena due to the unique structure of Yasmin-based applications that are split into kernel and user services. Since I have not been able to find a definitive answer on the literature about kernel granularity, I have decided to follow the following approach. At first all the necessary services have to be included in the kernel, and then services will be selectively moved from kernel to user space (droplets) until a good balance of performance and flexibility is achieved. The outcome of this iterative process produced a slim kernel that is not exactly minimal but that contains only the services used by all the droplets. The decision not to have a very minimal kernel has been made after having developed a few prototypes. If a minimal kernel is used, the application performance is affected proportionally to the number of times that non-kernel entities (droplets) have to use kernel services. This performance degradation is acceptable if there is a real need to have a very minimal kernel, namely if some of the services included in the kernel need to be modified at runtime. Because this is not the case, I have decided to design a slim kernel which is more performant with respect to a minimal kernel but which is still flexible because the services that the kernel provides are not expected to be modified at runtime.

Event Manager

In general events are issued when a certain situation occurs. Common events are mouse click and disk insert. In the network management field it is often necessary to perform periodic tasks. This happens for instance when an important file system has to be monitored in order to avoid users to fill it up. With the advent of threads, many developers implement periodic tasks using a thread. Each thread contains an endless loop in which the thread performs the task and then sleeps for a specified amount of time. Unfortunately the number of threads an application can spawn is limited, whereas the number of these tasks can be potentially high, and it is not possible to know in advance what is the maximum number of periodic tasks an application has to perform. In addition, as far as the author knows there are no patterns which can be applied to the problem. Due to this, delayed events have been defined (See "Event Manager" on page 95) with the purpose to be able to handle a potentially unlimited number of periodic tasks without the need to spawn a thread for each of them.

NOTE

Delayed events have the advantage that can be implemented with or without threads, hence they do not necessarily need thread support from the operating system.

Conclusion

Although considerations similar to the ones above can be applied to every Yasmin component, this section demonstrates that each component has been designed in an iterative way:

- collection of the requirements;
- identification of the mandatory features the component must have;
- survey of the patterns and solutions already available that could be used to design the component;
- decision to reuse (and modify) an existing solution or to design a new one in the case no satisfactory solutions are available;
- implementation of a small prototype used to select the adopted solutions in case there are a few alternatives to the component implementation.

It is worth noting that very seldom there is a unique solution to a given problem. In fact quite often a certain solution is preferred with respect to others because it is the best compromise between advantages and drawbacks.

4.7. Comparison with Other Architectures

In order to better understand the strengths and limitations of Yasmin, some criteria for comparing it with the architectures presented in the previous chapter have been identified. The result of the comparison is contained in the following table.

	OpenDoc	OLE	Java Beans	Yasmin
Application Domain	Generic and document-centric applications.	Generic applications.	Generic applications.	Open distributed systems with emphasis on network management.
Architectural Style [Garlan93]	Data abstraction and object-oriented organisation, layered architecture.	Event-based, implicit invocation architecture, layered architecture.	Data abstraction and object-oriented organisation.	Distributed (client-server), event-based, implicit invocation, meshed architecture.
Programming methodology[a]	Object-Oriented.	Object-based.	Object-Oriented.	Object-based.
Static Typing	Objects statically typed via the class hierarchy.	Individual (not objects) interfaces are statically typed.	Statically typed classes and interfaces.	Individual (not objects) interfaces are statically typed.
Distributed Object Support	Provided by the DSOM Framework.	Provided by Networked OLE[b] (DCOM).	Provided through the Java RMI API.	Yes, provided by the Communication Services.

	OpenDoc	OLE	Java Beans	Yasmin
Distributed Communication Protocol	Corba (IBM DSOM).	Lightweight-RPC (MS-proprietary RPC).	HTTP, Java RMI, Java IDL and JDBC.	Not specified by Yasmin. Liaison (cf. next chapter) uses the HTTP protocol.
Component Migration[c]	Not applicable.	Possible but entirely user-written (applicable only in the case of DCOM).	Yes.	Yes (not between different OS types)
Binding Technology	1. Compile time (using containment, inheritance); 2. Runtime (via OSA).	1. Compile time (using containment, aggregation). 2. Runtime (via OLE Automation).	1. Compile time (using containment, inheritance, property editors and composition wizards). 2. Runtime (scripting languages).	1. Compile time (using cooperation and aggregation); 2. Runtime (via external requests like the external bindings described in the next chapter).
Runtime Evolution Support[d]	No.	No.	No.	Yes.
Interoperability[e] **Support**	Yes (via DSOM).	Yes (at binary level).	Yes (the Bean interfaces to be supported guarantee basic interoperability).	Yes (droplet interface and events guarantee basic interoperability).
Interoperability Type	Object-oriented interoperability.	Procedure-oriented interoperability.	Object-oriented interoperability.	Procedure-oriented/event interoperability.
Versioning	Automatic (major & minor numbers).	User-written (major & minor numbers).	None.	Automatic through the DM (user-written version number).
Object Lifecycle Support	Automatic, including object creation, object destruction, object factories.	User-written, implemented using reference counting.	Automatic, provided by Java.	Automatic, including object creation, object destruction, object factories.
Concurrency Support	Not specified.	Not specified.	Yes.	Yes.
Platform Coverage	AIX, MacOS, OS/2, Win95/NT.	MacOS, Win95/NT.	All platforms which offer Java support.	Platform neutral. Currently running on AIX, OS/2, MacOS, Linux, Win95/NT.

a. According to [Wegner87].

b. Available only on WindowsNT release 4.

c. Migration concerns, in a client/server environment, the replication/move of a component from the server to the client host in order for the previously remote component to be available in the local environment.

d. Evolution concerns the ability to change/extend the application behaviour.

e. Interoperability is the ability of two or more entities to communicate and cooperate despite differences in the implementation language, the execution environment, or the model abstraction.

4.8. Final Remarks

This chapter presented a new component-based architecture for software applications called Yasmin. Its main characteristics are that it is:

- highly portable, configurable, and extensible;
- built upon dynamic software components called droplets;
- founded on cooperation and delegation, used to glue components together;
- a slim and efficient kernel that relies on a fast event handler;
- an efficient use of system resources, which enables Yasmin-based applications to run on environments of limited computing power.

Despite Yasmin's native ability to work in a networked environment, this architecture is general enough to be used not only on this application field. Its ability to modify and extend applications at runtime makes it attractive in dynamic environments where new services and resources must be supported while the original application must remain active and ready to serve incoming requests. The following chapter covers the design and the implementation of Liaison, a Yasmin-based application, used to manage telecommunication networks.

5 Liaison: Yasmin at Work

This chapter covers the design and implementation of Liaison, an application based on Yasmin developed for the purpose of network management. As Liaison represents le “trait d’union” between Yasmin and the network management world, familiarity with terms and concepts covered in the previous sections is necessary.

5.1. Introduction

Today’s networks are composed of many interconnected heterogeneous resources. It is essential to use tools able to master this great amount of complexity and diversity. Network management standards provide the basis for hiding differences among resources thus allowing them to be managed in a consistent way. Although this is the first step towards integrated network management, there is a clear need to provide the end-user tools able to manage network resources in a single and consistent way. There are many solutions on the market that address some aspects of this problem. Network resources often come with proprietary software applications that run on a specific platform and are capable of managing such resources. When a network is composed of different heterogeneous resources, the system administrator often has to use various tools to manage the different network parts. A partial solution to this problem is to adopt very powerful and general-purpose network management tools. The disadvantage of this solution is that such tools are costly and require powerful hardware to run them. They are also complex in terms of usage and configuration. Therefore this solution is suitable only for customers with a reasonably large budget, but certainly not for small and mid-sized companies or many public institutions or universities.

Other events in the computer industry contributed to make this scenario even more complex as depicted in the following figure.

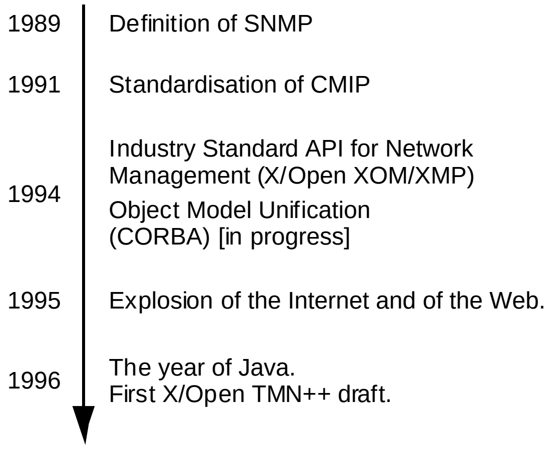

1989
Definition of SNMP
1991
Standardisation of CMIP
1994
Industry Standard API for Network Management (X/Open XOM/XMP)
Object Model Unification (CORBA) [in progress]
1995
Explosion of the Internet and of the Web.
1996
The year of Java.
First X/Open TMN++ draft.

Figure 19. Recent Relevant Events in the Network Management/Internet World

In the current 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 [CMIP] and SNMP [SNMP]. In this view, X/Open has defined an industry standard C-based API called XOM/XMP (see "XOM/XMP" on page 63), 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 a new effort by X/Open called TMN++ [TMN++96] pretends to add an object-oriented interface on top of XOM/XMP in order to have a simpler object model for programming management applications. Although TMN++ is at the moment far from being completed, this effort appears to be yet another dinosaur in terms of complexity and final application size.

Parallel to these activities, the increasing popularity of the CORBA [OMG92] industry-standard pushed many people to write mappings between CMIP/SNMP and CORBA (see "Interdomain Management" on page 66) 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, there are many different mappings available today that usually do not fully support CMIP/SNMP (see "Interdomain Management" on page 66). Another drawback is related to the significant amount of code that must be generated to implement these mappings and that has to be linked with the final application. Moreover, a network management expert who intends to write a management application must learn CORBA, IDL, how the mappings have been defined, and must have an ORB installed somewhere. It is clear, for instance, that the initial vision of SNMP to be simple and light has been jeopardized.
Whereas in the management community most efforts are spent defining a single API or object model for coding management applications, in the Internet world a new tool, the World Wide Web, has come into the play. It was pioneered by an English engineer, who envisioned a system able to handle various Internet protocols as well as different data formats using a single consistent user interface. In addition, this system handles a new protocol called HTTP (HyperText Transfer Protocol) [HTTP] and a new data format called HTML (HyperText Markup Language) [HTML]. HTML is a

simple markup language used to create hypertext documents that are portable from one platform to another. Such documents are identified by URLs (Uniform Resource Locators) [URL]. The growing popularity of the web offers a new way to provide wide access to complex software applications. Almost every platform supports web browsers, and many people are using the web as a single tool to access many different services. Hence the web is quickly becoming a sort of general-purpose container for every kind of information, whether distributed on the network or contained in a single large database. Another important advantage derived from using the web is related to software development and distribution. Many problems arise for various reasons: the high cost of porting software to different platforms together with the difficulty of training people, maintaining and updating software running on different operating systems are just a few. These problems can be addressed using the web because:

- it allows a single and established user interface, thus eliminating the amount of training necessary;
- it does not require software porting because web browsers run on almost any platform;
- it makes software maintenance easy because the web-enabled applications are installed in a few places by the system administrators, who can maintain them easily without the need to update the client machines that run conventional web browsers.

From the beginning, the web showed great potential, which did not fit in with the limitations of the initial vision, according to which the web was simply a tool to retrieve and visualise multimedia documents such as text, graphics, and sound. It was clear that the web required some extensions to make it more interactive. The first solution proposed by NCSA was CGI (see "CGI Applications" on page 48), which consisted essentially of a simple way to start applications on the server side, which returned data based upon the requested URL. This solution still suffered from the client-server nature of the web, meaning that the web browser requested documents from the server, thus making the client dependent on the server. People demanded more intelligence on the client side to interactively run applications. The Java language [Sun96a] solved this problem. *Java*, a concurrent object-oriented programming language, is compiled to a platform-independent bytecode and then interpreted by a virtual machine (VM). The integration of the Java VM into web browsers allowed people to run small applications, called applets, inside the browser, hence to put intelligence into the browser. Java native ability to handle network communications and concurrent processing (multithreading) facilitated the creation of Internet-aware applications (i.e. applications that support Internet protocols such as HTTP or Gopher) and contributed to the explosion of Internet.

5.2. Motivation

Although many network management tools are on the market, some of

which have been produced by the author [XOBJ] [OSIMIS] [IBM95a], they frequently do not work together in a consistent manner mostly due to the way these tools implement standards. Moreover they are targeted at big customers with huge networks to manage.
The idea to integrate the world of network management into the web emerged from users' frustration with conventional network management tools and from the need to have simple but powerful tools accessible from almost every platform. The time necessary to install and configure such tools, together with the difficulty of using them, was too much for an average user. Although computers are becoming faster and more powerful, software tools have not changed very much. A few years ago it was acceptable to have simple, user-friendly tools for activities such as word processing and complicated but powerful tools for more complex tasks. Today, users demand simple user interfaces for both simple and complex tasks. The great popularity of the web combined with its power and acceptance has rendered it a reasonable candidate for building a new class of network management tools. These tools will be accessible from every platform capable of running a web browser and they will have one user interface that is simple enough to be used by the average user. Often, for example, graphical user interfaces are not very easy to use, although they may look quite appealing.

Although web-based network management seems to be the solution to every management problem, it is clear that the limitations of the web in terms of interactivity and master/slave architecture (the server cannot push data to the client) prevent this. In this context *Liaison* was born. Liaison is a proxy application [Shapiro86] [Gamma94] based on Yasmin which interfaces the HTTP-based world with network management.

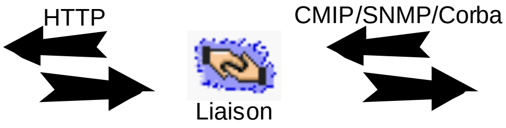


Figure 20. Liaison at a Glance

The idea is to use Liaison to shield management applications from the complexity of the management world instead of moving this complexity up to the applications, as usually happens. The HTTP protocol has been selected because it is standard, open, and used by the web to transport multimedia information. In this context, web-based management is a special case of HTTP-based management, where the type of data transported by HTTP is HTML. Liaison allows client/server applications based on HTTP to be created. Basically Liaison acts as a server application that provides the management functionality accessed by client applications via HTTP. The simplicity of HTTP does not have to be seen necessarily as a limitation. In fact HTTP makes it extremely difficult to move complex data, so it pushes developers to simplify the management data being exchanged between Liaison and client applications.

The following sections cover Liaison and its implementation in detail. Groups at other companies [Jander96] [Damocles95] [Marben95] and universities [Knight95] [Schönwälder95] have worked on integrating web technologies with network management. Nevertheless the present work is different in several aspects:

- it is not a simple web-ification of existing tools but provides a general framework to accommodate different protocols (SNMP, CMIP, CORBA and others) (see “Web-based Management” on page 111 and “HTTP-based Management” on page 121) [Taligent93] [Taligent95b];
- it allows efficient client/server applications for the purpose of management and based on HTTP to be built [Meyer95];
- it has been able to seamlessly integrate different management protocols and object models for interdomain management in a open and extensive framework, delivering today what other organisations such as X/Open and OMG have not yet released;
- it has proposed an open way to perform HTTP-based management which has been published to the community as an IETF Internet Draft [Deri96c].

It is worth remembering that Liaison is a recognised pioneer in this field. It is the very first implementation of integrated web-based management for CMIP and SNMP (and so far the only one), and the first application to propose and implement HTTP-based management. Its public availability on the Internet at no cost and for several platforms has turned it into a well-accepted platform for network management.

5.3. Welcome to Liaison

In an effort to integrate the network management world with the web, the author developed an application called Liaison. Liaison is based on Yasmin and fully follows Yasmin guidelines because:

- it is based on the droplet paradigm (see “Droplets” on page 80) as demonstated in “From Theory to Practice: Implementing Droplets” on page 207;
- it implements a light kernel containing the components specified in “Yasmin Components” on page 92.

The kernel part of Liaison, usually called the *Proxy* server, implements Yasmin’s kernel services. It is a pure application in the sense that it does not implement management functionality which instead are implemented inside droplets. As seen above, droplets have the ability to be replaced and added at runtime, allowing the behaviour of the application that contains them to be dynamically modified and extended. Each droplet, built upon shared libraries, implements one or more services. These components cooperate through Liaison, which handles communication with the outside world. Liaison implements the HTTP protocol, so remote web users can access it directly without the need to have, for instance, CGI applications that interface the HTTP server with Liaison itself. This solution presents several advantages in terms of performance and configuration. Liaison comes with droplets that implement:

1. web-based management with full CMIP and SNMP support;
2. a basic directory service for locating management resources and other Liaison instances running on local or remote hosts;
3. a metadata repository only for SNMP because the metadata information relative to CMIP is retrieved by Liaison directly from the OSI stack;
4. external bindings (see "HTTP-based Management" on page 121), which enabled the creation of client/server management applications;
5. CORBA interfaces for interdomain management (see "CORBA Interfaces" on page 129).

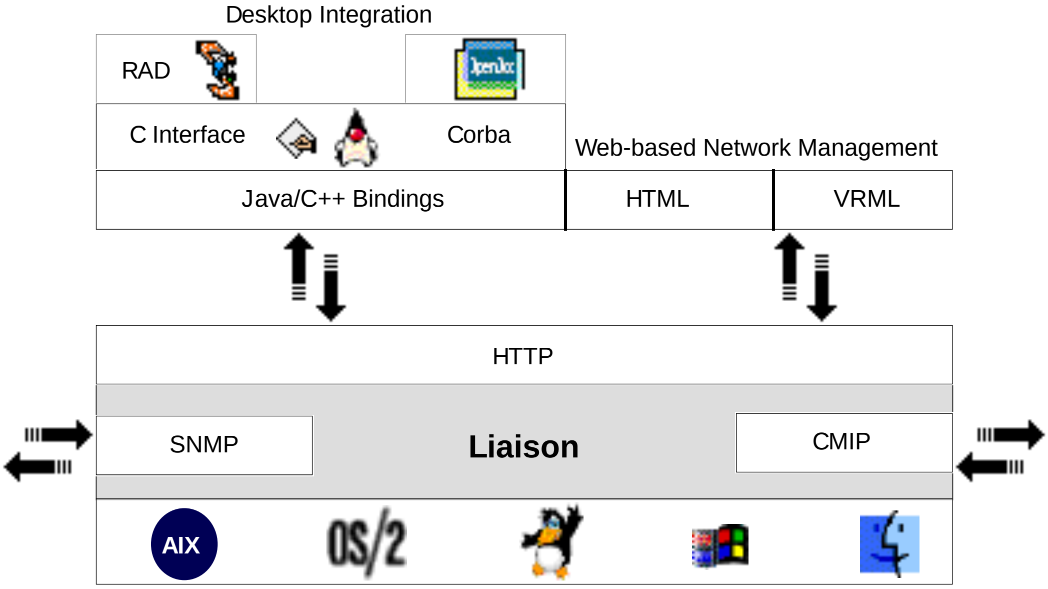


Figure 21. Liaison's Components

Thanks to droplet flexibility, it has been easy to support different management protocols and to integrate them with Liaison. The idea is to implement a droplet for each management operation and then cooperate with the existing droplets in order to reuse the services they provide especially with respect to metadata access. This demonstrates how powerful software components are and how they allow existing services to be reused and then applications to be built incrementally, instead of starting from scratch every time.

Each droplet communicates with remote requesters through Liaison, which acts as a tunnel application. As the droplet code is reentrant, concurrent requests can be served. Liaison communicates with remote clients over HTTP and exchanges local messages/events with the droplets. Use of the HTTP protocol has been preferred over other protocols because it is simple, well established and can flow through firewalls, allowing hosts to be managed everywhere on the network. Because each droplet exploits existing services, the average size of a droplet that implements a management 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 Liaison structure is stateless and it is up to Liaison clients to maintain state information. This contributes to keeping the code simple and it allows droplets to be replaced at runtime without having to save a large amount of data concerning the state, similar to what threads do.

So far, Liaison strictly follows Yasmin. The droplets make it unique and allow it to be tailored for network management. In the following sections the droplets part of the basic Liaison distribution is analysed. These droplets have been divided according to the services they provide for the purpose of web-based and HTTP-based management.

5.4. Web-based Management

The current trend is to replace centralized and powerful computers with a network of cheap and mid-power computers. Network management has been affected by downsizing, and the current trend is to use high/mid-sized hosts for running agent applications and to reserve limited-power machines for manager applications. These manager applications feature:

- ease of use: users of mid-sized machines are accustomed to the WIMP (Windows Input Mouse Pointers) metaphor and do not like applications with a poor user interface;
- plug 'n play: average users want to install an application and run it right away without having to be a configuration expert;
- low resource usage: manager applications should not use all the available resources, but are expected to leave room for other running applications.

Mobility is another important factor. People tend to move more frequently than in the past, and a whole set of new technologies has to be introduced in order to access network resources transparently and independently from both the location and the computer type [Reilly97]. This pushes software developers to create applications that run efficiently on different platforms independently of the version and type of the operating system. One solution is to use frameworks that hide the differences among operating systems, using proxy agents that can be accessed by a well-established protocol like TCP/IP or by interpreted languages. In this complex scenario, the web plays a key role due to its growing popularity and its ability to access every kind of information using a single established interface.

In this context, web-based management was born. *Web-based management* is the system and network management using web technologies [Kasteleijn97] [Barillau97] [SimpleTimes]. Web-based network management applications have the following properties:

- different tools share the same 'look and feel';
- limited configuration: the system locates network resources and limits the number of choices the user has to make by providing default values;

- on-line help: documentation and other facilities are directly available on-line and exploit the web hypertext facilities;
- active support: in case of error, the system identifies the cause of the problem and indicates possible solutions;
- limited alternatives: the system shows only valid operations, thus preventing the user from performing invalid ones.

The web has been preferred over other tools [Rice95] or platform-independent languages such as TCL/Perl [Pavlou96] [Schönwälder95] (see "TCL/Perl-based Management" on page 66) because it:

- is an accepted and well-known user interface;
- is available for virtually every platform;
- is general purpose: it is able to accommodate today's services and is open to future extensions;
- has an integrated hypertext capability ideal for navigating different kinds of information;
- is more secure: security facilities provided by network management protocols can be combined with emerging web security protocols;
- is a web-enabled program that can be used by different platforms without the need for it to run on those platforms.

Contrary to other approaches [Marben95], Liaison [Deri96a] has used the web not only to provide a single and unified user interface but pretends to offer users the same ease of use that they experience navigating HTML documents. Liaison takes an approach similar to the desktop metaphor on MacOS™ [Apple96]. Users do not have to configure anything or know where resources are located. The system does this itself and it shows the network resources using well-established macebearers such as files and folders. This automatic configuration and resource discovery facility combined with the ease of use of the web allowed a light and powerful application to be built to access network resources from virtually any platform.

Liaison is based on the idea that the complexity of protocols such as CMIP or SNMP has to be hidden by the system. It then contains facilities for:

- automatic discovery of network resources;
- access to metadata information and error handling;
- ASN.1 representation using a string API instead of complex data structures [IBM95a];
- enabling only the execution of valid operations.

These facilities greatly simplify interactions with the user. Liaison requires user input only when it is unavoidable such as when a new value for an attribute must be specified. In all other cases Liaison presents the user with only the set of operations that are valid at that point of the execution by exploiting GDMO and ASN.1, if available. This prevents invalid commands from being issued and simplifies the application by removing many checks on the user data. For instance Liaison suppresses the display of the CMIP M-ACTION icon when a certain instance, according to the GDMO, cannot execute actions. In other situations, like when the value for the scope or synchronization parameter has to be provided, Liaison prompts the user to choose only valid values. Another advantage of this approach is the limited amount of information the user has to keep in mind. For instance Liaison shows the ASN.1 syntax of every instance attribute by analysing the encoding/decoding data currently used by the OSI stack.

As shown in figure 20, Liaison implements the HTTP protocol. Hence it is seen from web browsers as a web server. For every request issued by the web browser, Liaison processes it and returns the HTML response document to the browser. If Liaison is not able to process the request, it acts like a tunnel application and forwards the request to the local HTTP server, if any. This solution has several benefits with respect to conventional scenarios in which the HTTP server invokes external applications using the CGI interface:

- no need for CGI scripts/applications to interface Liaison with the HTTP server,
- no latency due to the CGI interface because Liaison is directly connected to the web browser,
- very low load on the HTTP server: modern browsers have a local cache and therefore Liaison contacts the HTTP server very seldom only when a new icon not yet cached is requested by the web browser.

Thanks to this solution and to the multithreaded architecture of Liaison, it is possible to serve multiple concurrent requests efficiently even on machines of limited computing power.

Mapping Management Requests to URLs

Management requests issued by the web browser are mapped to URLs. This mapping is very important because the entire Liaison architecture relies on it. It allows a URL to be mapped uniquely to a management operation and vice versa. This mapping has to work with conventional HTTP servers and web browsers in order to access Liaison from every host without the need to install special software. As specified in [Deri95d] [Deri96c], URLs are composed of five elements:
`http://<host>/<protocol>/<operation>/<context>?<parameters>`, where:

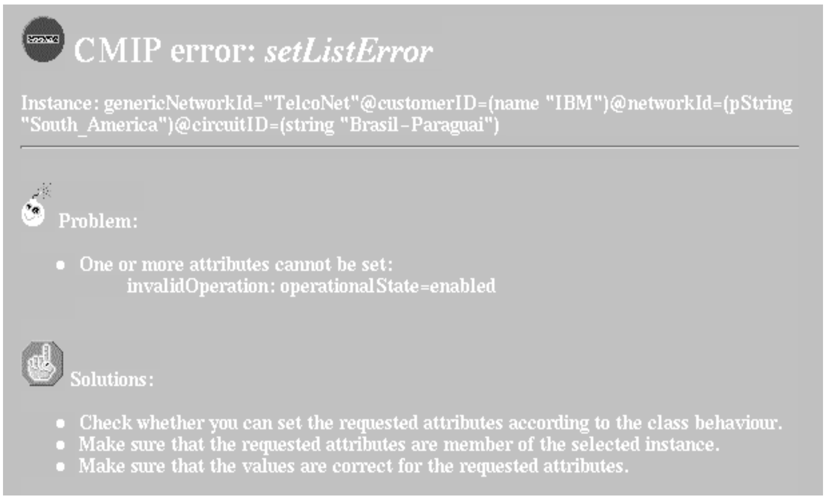
1. `<host>` identifies the host where the HTTP server runs;
2. `<protocol>` specifies the protocol used (either CMIP or SNMP);
3. `<operation>` specifies the management operation (CREATE, GET, ...);
4. `<context>` specifies the context used, if any;
5. `<parameters>` contains the operation parameters.

For instance, if `<protocol>` is set to CMIP, `<context>` contains the agent title and the managed object instance, whereas for SNMP `<context>` specifies the object identifier of the attribute. `<parameters>` contains operation-specific parameters (e.g. for CMIP M-SET, `<parameters>` contains the attribute(s) to set and their new values) and other values such as timeout or the name of the host on which the agent is running. All the ASN.1 values (such as the ones contained in the `<parameters>` field) have to be expressed in string format and with binary values. This is because this mapping is supposed to be used not only by software applications but also by human operators from within their web browsers. ASN.1 values for SNMP/CMIP are fairly simple and follow public guidelines [Geiger94]. The following examples show how the mapping works, assuming an HTTP server running on the host `kis.zurich.ibm.com`:

- CMIP SET the attribute `administrativeState` of the managed object instance `systemId=(name IBM)` contained on the agent whose AE-title is `abc` to enabled: `http://kis.zurich.ibm.com/CMIP/SET/abc/systemId=(name+IBM)?administrativeState=enabled&timeout=30`
- SNMP GET for the attribute `sysDescr` contained on the SNMP agent running on the host `bal.zurich.ibm.com`: `http://kis.zurich.ibm.com/SNMP/GET/sysDescr.0?Host=bal.zurich.ibm.com&Community=public`

Although this mapping is almost straightforward, Liaison hides it from the user. In fact, Liaison shows the user a starting point and then the user does not have to worry about the syntax because URLs are dynamically generated by the system. This usually happens with most web sites where the navigation starts with a simple entry point and then the system returns more complex URLs embedded on the returned HTML documents.

Each management primitive such as CMIP M-ACTION or SNMP GET is implemented in a droplet. The `dropletName` (see figure 15, "Droplet Information," on page 87) follows the URL format `/<protocol>/<operation>` (for instance for CMIP M-ACTION the name is `/CMIP/ACTION`). The web sends the requested URL to Liaison, which compares the initial part of the URL with `dropletName`. If they match, Liaison forwards the request to that droplet, otherwise Liaison checks whether the request corresponds to a local file. If not, the request is forwarded to the local HTTP server, if any, because the request cannot be handled by Liaison. Each droplet is responsible for executing the request and building a response HTML document, generated on the fly, containing either the output of the request or an error message in a user-friendly way, simple enough to be understood by almost any user.



CMIP error: *setListError*

Instance: genericNetworkId="TelcoNet"@customerID=(name "IBM")@networkId=(pString "South_America")@circuitID=(string "Brasil-Paraguai")

Problem:

- One or more attributes cannot be set:
  invalidOperation: operationalState=enabled

Solutions:

- Check whether you can set the requested attributes according to the class behaviour.
- Make sure that the requested attributes are member of the selected instance.
- Make sure that the values are correct for the requested attributes.

Figure 22. Simple Error Message

Liaison's architecture - Yasmin - greatly simplified the development process due to the clear separation, at the droplet interface, between the droplet itself and the rest of the application. This separation also allows new services to be accommodated easily and the application size to be kept small because common routines are contained in the core part of Liaison and not replicated in every droplet. Currently, Liaison comes with droplets for every CMIP and SNMP operation and with additional droplets that implement directory services and Liaison-to-Liaison communication facilities.

The droplet paradigm allows one to combine services easily. For instance Liaison comes with a droplet that implements a directory service. This service is used by the discovery droplet, which, is responsible for locating network resources. Composition of services has several advantages. It keeps the application complexity low and allows service implementations to be replaced with new and more efficient ones without having to affect the users of those services.

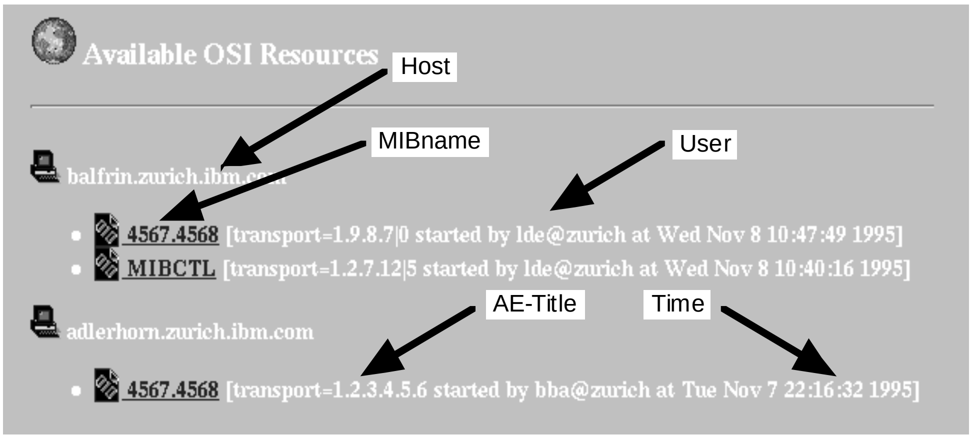
Liaison has the ability to transparently locate and exchange information with other Liaison's running on different hosts. This enables one to use the Liaison that is closest to the managed resource. In fact, thanks to the TCP/IP-based Liaison-to-Liaison communication services, a Liaison that has to deal with management resources running on a remote host delegates whenever possible the request to another Liaison that is running locally with respect to the managed resources. This solution allows bandwidth to be saved because it ensures:

- local computation: all communications Liaison-managed resource are local;
- Liaison-to-Liaison communication uses a simple protocol that moves less data than an equivalent CMIP/SNMP request/response;
- Liaison-to-Liaison communication is always 1:1 (one request/one response), whereas CMIP is 1:n in the case of scoped operations.

Liaison usually is installed by the system manager on the same host where the OSI stack runs. Hence users can access network resources from both conventional hosts and portable machines connected via SLIP or PPP to such a host. In this way, it is possible to have a single Liaison application and multiple web browsers running on light and cheap machines accessing the agent.

Using Liaison

Liaison has been designed to run in a dynamic environment, where network resources can change state very often and where users access these resources using portable machines that are frequently attached and detached from the network. Interaction between the user and Liaison has been made very simple. The user starts a web browser and connects it to Liaison. Liaison then returns an HTML document containing the network resources currently available. This document is used as a starting point for navigation. The following picture shows the initial page for OSI resources.



Available OSI Resources

balfrin.zurich.ibm.com

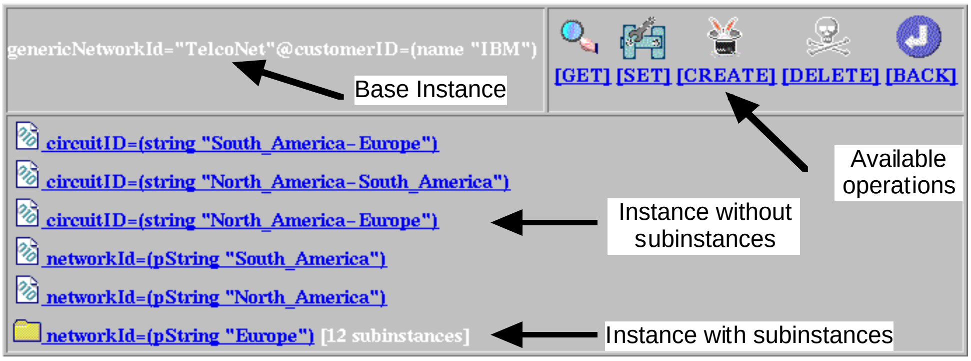
- 4567.4568 [transport=1.9.8.7|0 started by lde@zurich at Wed Nov 8 10:47:49 1995]
- MIBCTL [transport=1.2.7.12|5 started by lde@zurich at Wed Nov 8 10:40:16 1995]

adlerhorn.zurich.ibm.com

- 4567.4568 [transport=1.2.3.4.5.6 started by bba@zurich at Tue Nov 7 22:16:32 1995]

Figure 23. OSI Resource Discovery

The resource discovery droplet is responsible for building the HTML document above by communicating with the yellow page droplet responsible for collecting information related to the network resources. OSI resources are grouped by host name. Each resource has a MIBname (e.g. MIBCTL) and shows the AE-title, the user who started it and the time. URLs are dynamically generated by Liaison. Thus the user does not have to type or remember them because they are embedded inside HTML anchors.
Once an OSI resource has been selected, all the following requests will be relative to it. Liaison shows instances contained in an OSI resource using the folder/file metaphor used by modern operating systems.



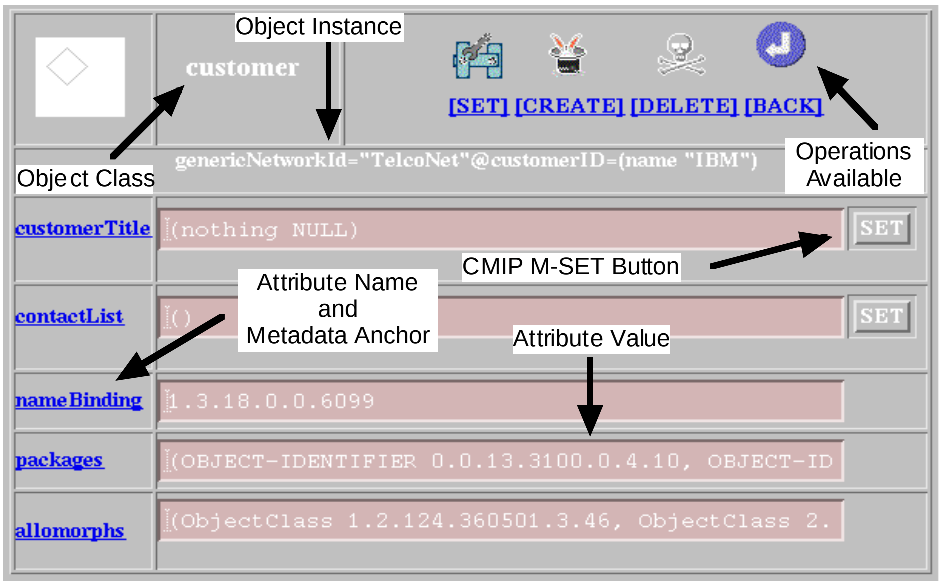
genericNetworkId="TelcoNet"@customerID=(name "IBM")

[GET] [SET] [CREATE] [DELETE] [BACK]

circuitID=(string "South_America-Europe")
circuitID=(string "North_America-South_America")
circuitID=(string "North_America-Europe")
networkId=(pString "South_America")
networkId=(pString "North_America")
networkId=(pString "Europe") [12 subinstances]

Figure 24. OSI Resource Exploration

A file represents an instance without subinstances, whereas a folder represents an instance containing subinstances. The instance name, displayed at the top of the window, is relative to the base instance. Icons at the top-right of the window represent the operations that can be performed on the base instance. In this case of the figure above, the M-ACTION operation cannot be performed because the base instance, which is a member of the class customer, does not allow actions according to its GDMO definition. This example shows how Liaison can access metadata information used by the OSI agent in order to check whether certain operations can be performed. In the current version, Liaison provides facilities that allow this information to be retrieved directly from the OSI stack. There is also a droplet that looks at the encoding/decoding information and then builds the ASN.1 type corresponding to a certain syntax. This is depicted in the following figure.



customer

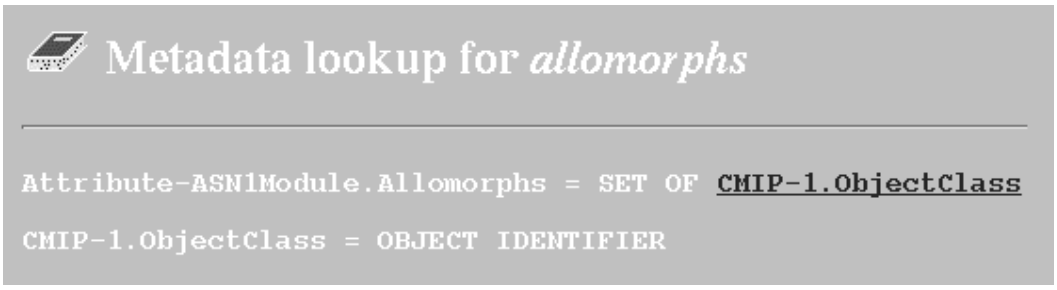
[SET] [CREATE] [DELETE] [BACK]

genericNetworkId="TelcoNet"@customerID=(name "IBM")

customerTitle	(nothing NULL)	SET
contactList	()	SET
nameBinding	1.3.18.0.0.6099	
packages	(OBJECT-IDENTIFIER 0.0.13.3100.0.4.10, OBJECT-ID	
allomorphs	(ObjectClass 1.2.124.360501.3.46, ObjectClass 2.	

Figure 25. CMIP M-GET

The attribute name is an anchor. If the user selects it, the droplet responsible for the metadata services queries the OSI stack and then fetches the syntax information from it. The following picture shows the `allomorphs` syntax produced by the metadata droplet.



Metadata lookup for *allomorphs*

```
Attribute-ASN1Module.Allomorphs = SET OF CMIP-1.ObjectClass

CMIP-1.ObjectClass = OBJECT IDENTIFIER
```

Figure 26. Metadata Lookup

Regarding the figure 25 on page 117, it should be noted that the SET button is only present for those attributes that can be set. If multiple instances have to be set, the M-SET operation is performed by selecting the SET icon. It allows multiple instance attributes to be set as well as scope, synchronization, and filter parameters to be specified.

VRML: Adding 3D to Network Management

Over the past few years, graphical computer capabilities have been improved significantly. Pushed by the game industry and by vertical markets such as computer graphics and medical visualisation, computer manufacturers are producing increasingly faster chips and video systems able to draw and animate realistic images and mathematical models. This quick evolution in computer hardware enabled the move from 2D to more realistic 3D representations. After some attempts to create APIs and modeling languages for 3D, Silicon Graphics, a pioneer of computer graphics, released a graphics library called OpenInventor [SGI94], available on many platforms. Its wide acceptance in the industry contributed to the unification of the various existing APIs for 3D visualisation, apart from being used to originally implement *VRML*, an acronym for Virtual Reality Modeling Language [VRML]. VRML is a modeling language, hence it describes a set of 3D elements usually called a 3D virtual world. A software application called VRML viewer interprets VRML to render the world, allowing people to explore and navigate it. One of the most interesting features of VRML is its ability to link virtual worlds with the Web as it associates an HTML anchor with each VRML element. This allows users to jump to other VRML worlds and HTML documents and vice versa, exploring the Web as if wandering through a vast universe.

Aside from some rare exceptions, network management visualisation is still limited to 2D. In many cases, classic 2D representations are too limited and do not allow complex information to be represented easily. Recently, the great diffusion of the Web promoted the development of Web-based network management systems, such as Liaison, which shows how greatly management systems can benefit from their integration into the Web. The idea to apply 3D visualisation techniques to selected network management problems [Deri97b] is derived from the need to represent management information in a way that is as close as possible to reality. Conventional 2D visualisation systems have many limitations, some of them being:

- a lack of realistic representation of information whenever such information is implicitly in 3D format;
- a lack of expressiveness whenever a large quantity of sparse information has to be combined in order to build a compound view of it;
- an inability to display topological information as it is in reality.

Apart from all this, 3D visualisation offers several interesting advantages. It allows people to represent the information in a way very similar to reality: to change a perspective, move a viewpoint, and add or eliminate details by getting closer to the information. Beyond these benefits, it is not straightforward to identify how to apply 3D to network management and where to prefer it to 2D. 3D is significantly more computation-costly than 2D and it is usually not platform-independent, in the sense that applications written using standard 3D graphic libraries cannot run unmodified on other platforms. In addition, the cost of writing an application for 3D visualisation is high in terms of development time and expertise. The solution to all these problems has been the adoption of a modeling language because it allows different worlds to be represented easily, leaving to the language viewer the task to visualise the information. VRML has been selected because it is currently the standard modeling language: it is also well integrated with the Web because:

- web browsers usually come with a VRML viewer,
- VRML files are retrieved using HTTP, the same protocol used by the Web to retrieve documents,
- it is possible to jump transparently from HTML files to VRML and vice versa.

The integration of the Web with network management is becoming increasingly important because it allows network resources to be managed in a simple, cheap, platform-independent way directly from within a standard Web browser. In this view VRML has been preferred over more powerful modeling languages like 3DMF (QuickDraw 3D Metafile Format) [3DMF] because:

- HTML is a simple, platform-independent and elegant way to display information that can be represented in 2D. The ability to jump from HTML to VRML and vice versa enables developers to use the best visualisation format for each situation;
- VRML can be visualised efficiently on standard PCs without the need to purchase additional custom hardware;
- VRML is a very simple yet powerful language that is easy to learn and thus allows developers to create new VRML worlds or enhance existing ones without technical knowledge of 3D visualisation because the VRML viewer is responsible for this.

In the context of network management, VRML has been applied in the following situations:

1. Network Topology

   Network topology deals with the visualisation of network elements. Topology can be either logical or physical. Logical topology is used to visualise how elements are interconnected and what the connections are, without any constraint on an element's physical location or topological distance. Physical topology instead requires that elements be placed where they are really located and that distance constraints be satisfied. In the case of logical topology, the goal is to display the information in the most readable way. This is in order to show human operators the current network status without adding additional information such as element size or distance, which are not meaningful in this context and may confuse operators.

2. Hierarchical Information

   Quite often in network management, information is aggregated in hierarchical structures. In the OSI world for instance, object instances are stored in a containment tree. This means that an object instance can contain one or more subinstances in a tree structure. In SNMP the same structure can be obtained by splitting the information according to the MIB groups, which are identified using nested object identifiers. The TCP/IP protocol identifies hosts by assigning addresses of the type X.Y.W.Z, where X, Y, W and Z are integers; host addresses are then grouped in subnets according to a subnet mask.

3. Compound Information View

   Very rarely can all the potential information be represented in the same view because different users may be interested in different aspects of the same information. Moreover, in many cases the information to be represented is quite rich, so ways to "compress" such information have to be identified. This means that a representation has to be as clear as necessary and as rich as possible in order to depict most of the information in a single picture. Compound information means that several aspects of the same information have been combined to produce a simple representation that removes or hides any information that is irrelevant for a given representation. VRML can help in this respect because it allows a great amount of information to be included in a single virtual world.

The following picture shows an example of compound information in which topology information has been combined with management information.

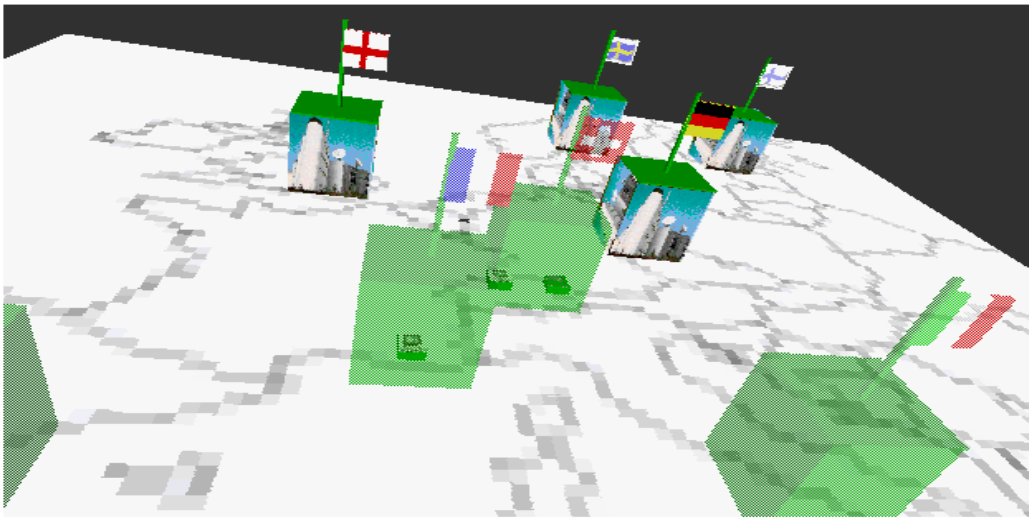


Figure 27. Simple VRML Compound Information View

A map of Europe has been wrapped over a 3D flat surface, national hosts are identified by buildings (boxes) located where they really are and labelled with the flag of the country. Services are depicted in a colour representing their state. Each element has an HTML anchor that links it to the corresponding resource details or that allow one to jump to other VRML/ HTML pages. The use of a VRML feature called the level of detail helps simplify the exploration of the world because the VRML viewer removes/adds objects to the view according to the current camera position. When the user is far away from a location of a host, a box with a flag on top of it is displayed. As the user approaches a box, the more details are shown. It is like navigating the containment tree with a camera: users do not have to enter boxes to see what is inside. It is also possible to represent much information in one view without demanding users to click several times to arrive at the object of their interest, although this possibility is provided as well.

Like in the case of HTML, droplets are responsible for generating the VRML world. In order to report to the HTML browser, which usually contains a VRML viewer, that the returned HTTP response is in VRML format, the HTTP response header specifies that its type is `x-world/x-vrml`. This allows browsers to handle both HTML and VRML files properly. As VRML is much more sophisticated than HTML, the VRML droplets do not generate VRML code directly but instead create an internal representation of the relevant data necessary to build the VRML world. Then, based on this information, the droplet expands some of the variables into pseudo-VRML files, which define the structure of the VRML world. This solution allows the VRML world to be modified easily by simply changing the content of these pseudo-VRML files.

Conclusion

This section has shown how the world of network management can benefit from using the web. Its integration enabled the creation of powerful CMIP/SNMP network management tools in an easy way. Its major advantages are that is:

- Internet-ready, simple, and platform-independent based on HTML/VRML, which are portable, compact, and widely established languages;
- an established and open technology;
- a limited configuration: Liaison automatically discovers network resources;
- based on cheap and standard web technology: Liaison can be accessed from almost any platform using conventional web browsers;
- a way in which users can manage network resources using a machine with limited power able to run a web browser;
- low in cost: a single Liaison can be used by multiple users;
- able to provide security through the web in addition to standard management protocols security mechanisms;
- able to display 3D data using their native format instead of flattening them to 2D and able to mix 2D and 3D data.

5.5. HTTP-based Management

A special case of web-based management is *HTTP-based management*, which concerns the system and network management using the HTTP protocol. Basically, management applications communicate with a mid-level manager using HTTP, and that manager performs management operations by converting the HTTP requests into management primitives, for instance by issuing CMIP, SNMP, or CORBA requests. Liaison is a mid-level manager suitable for HTTP-based management as depicted in figure 20 on page 108. HTTP-based management was proposed by the author to the management community in an Internet Draft at the end of 1996 [Deri96c] as a way to overcome common problems in the current class of management applications.

Towards HTTP-based Management

The increasing complexity and heterogeneity of modern networks has pushed industry and research towards finding a single and consistent way to manage 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 proved to have many drawbacks, most of which are 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.

HTTP-based management is based on the idea that network management has yet to be considered a special software engineering problem for which solutions must be built ad-hoc instead of reusing widely established concepts. Today most network management experts come from the "Vi, Unix and C" school and ignore new concepts and innovations like software components and truly object-oriented software development (most of the code is object-based but not object-oriented) [Cox91]. In addition, it is common practice to pretend to solve a problem by generating code for all possible situations (for instance XOM/XMP and many CMIP/SNMP to CORBA mappings generate a class for each datatype) instead of defining a way to simplify the problem. The advent of Java and TCL [Ousterhout94] demonstrated that the short reign of native-code-generating-object-oriented compilers is about to end [Udell94]. Internet and the market demand light, machine-independent applications capable of roaming from machine to machine. This requires light applications simple enough to be downloaded from the network and that do not require excessive amount of system resources.

These days, programmers want to use programming concepts instead of protocol concepts. All modern programming languages support exceptions and programmers are used to this; therefore it is time to replace protocol errors with exceptions and avoid executing a lot of code or converting information many times merely to obtain the value of an integer attribute.

In this context HTTP-based management was born. The idea is to simplify management protocols by introducing a mid-level manager which shields management applications from the complexity of underlying protocols, thus ceasing to move the entire complexity of these protocols up to the management applications as usually happens. HTTP has been selected because it is the protocol used by the web, it is reliable, open, and above all simple. For the latter reason, is quite improbable that users will jeopardise it in order to transport complex data. What is likely, is that the mid-level manager will find a method for simplifying the management protocols, and hence will not infect management applications with complex datatype or primitives. This is exactly what Liaison does: it unifies the various management standards in a light and simple object model.

Merging Network Management Standards

Network programmers need a single way to manipulate instances of various object models [Joseph90]. The main problem arises from the datatypes that have to be managed. In SNMP this is easy to handle because there are about ten different datatypes. CMIP is much more flexible in this respect and it allows the user to define new datatypes. Hence, the number of datatypes that a network management application has to handle is not determined a priori. Therefore a solution has to be defined in order to handle different datatypes of arbitrary complexity.

The solution proposed here is based on string notation [Geiger94], i.e. every datatype is represented by strings. Aggregate datatypes such as sequences or sets are a composition of such basic datatypes as integer or boolean. The fact having a single datatype makes things simple and allows applications written in any given language to use it, even if this representation slows down the system code. Nevertheless experience derived from using string representation for various commercial applications has shown that this inefficiency is relatively limited, especially on Unix™ systems [Bourne83] [Rochkind85], which are able to handle string very efficiently. Moreover, comparing a string representation to classical representation using C structures [ISODE] [Pavlou93], there are obvious advantages in terms of application size and ease of use/programming [Pavlou91]. 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 provided in order to make the conversion smooth and efficient.

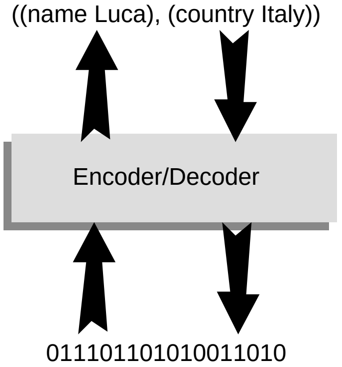


Figure 28. Data Values Encoding using BER

Programmers define data values using string representation and the encoder/decoder module converts this string into BER (Basic Encoding Rules) [ISO8825] and back.

NOTE

BER are encoding rules defined by OSI to convert ASN.1 (abstract syntax) into binary data (concrete syntax) exchanged between network applications.

The conversion is based on metadata information. In the IBM stack [IBM95a] for instance, the ASN.1 and GDMO compilers compile input documents into a data file that is read by the encoder/decoder at start-up time. These data files contain the datatype as well as object-model-dependent information. In the case of CMIP, data files contain information about managed object classes, name binding, actions, and notifications. In the 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 facilitate their work preventing for instance, the request of wrong operations. Once the problem of defining a single format to represent various syntax 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 with associations and 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 is 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 analysed, 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 automatic association handling, it is now possible to manipulate remote instances transparently using both SNMP and CMIP in a single and uniform way.

Mapping Management Requests to HTTP

In "Mapping Management Requests to URLs" on page 113, a mapping between URLs and management request has been defined. In that case, the format of the HTTP responses was not relevant because HTML code was returned. In the case of HTTP-based management it is necessary to define exactly the format of the HTTP responses, to design it in order to reduce the amount of information exchanged with the remote client, and to be flexible enough to allow not only CMIP and SNMP but even new protocols to be handled. Supposing we select the following URL, `http://kis.zurich.ibm.com/SNMP/GET/sysDescr.0?Host=bal.zurich.ibm.com&Community=public`, the HTTP-client (for instance a web browser) will send the following data:

```
GET /SNMP/GET/sysDescr.0?Host=bal.zurich.ibm.com&Community=public HTTP/1.0
[empty line]
```

The HTTP response returned by the HTTP server is always positive unless the requested URL cannot be found or if some other HTTP problems arise (for instance authentication problems). If the HTTP response is positive it will contain the SNMP response, which can be either positive or negative. The HTTP response contains a set of pairs (`<identifier>`, `<value>`) separated by a carriage return. If the SNMP response is negative, the last pair is (`<empty line>`, `<error code>`) where `<error code>` contains the error code corresponding to the SNMP request in numeric or string format (for instance `noSuchName` or `2` as defined in the SNMP standard). Identifiers are object identifiers, usually in symbolic form, whereas values are strings encoded using the encode scheme used by the HTTP protocol. A positive response for the previous requests is the following:

```
HTTP/1.0 200 OK
Server: IBM ZRL Proxy Server
Date: Fri, 28 Jun 1996 12:30:16 GMT
Content-type: text/x-www-form-urlencoded
Content-length: 35

sysDescr.0
IBM+RISC+System%2F6000
```

IMPORTANT

Note that responses are not sent in plain text but in the standard encoding format used to encode URLs. This is because encoding guarantees a unique representation of the value independent of the platform and prevents carriage returns contained in the returned values from being confused with those used to separate the entries.

The proposed solution allows either a single response or multiple responses encapsulated in a single HTTP response to be returned. In the latter case, the response contains multiple pairs separated by a carriage return. HTTP responses can contain additional fields, such as the value type, which can be used by the client application (for instance the web browser) to display the returned value properly. Similar considerations can be done for CMIP. The only difference with respect to SNMP is that CMIP scoped requests can return multiple CMIP responses, each of which contains multiple attributes relative to a specific object instance. In this case CMIP responses are separated by (`<empty line>`, `<empty line>`). Note that if the first line of the HTTP response is an `<empty line>` then the response is negative, otherwise it is positive. Hence there is no ambiguity between (`<empty line>`, `<empty line>`) and (`<empty line>`, `<error code>`) if `<error code>` is empty.

5.5.1. Application Side Bindings

Because clients communicate with Liaison over HTTP and because the data exchange type is based on strings, it is easy to write bindings - called *external bindings* - in any given programming language, whether, object-oriented or not. These bindings allow developers to create applications that manage CMIP/SNMP network resources exploiting the management services provided by Liaison.

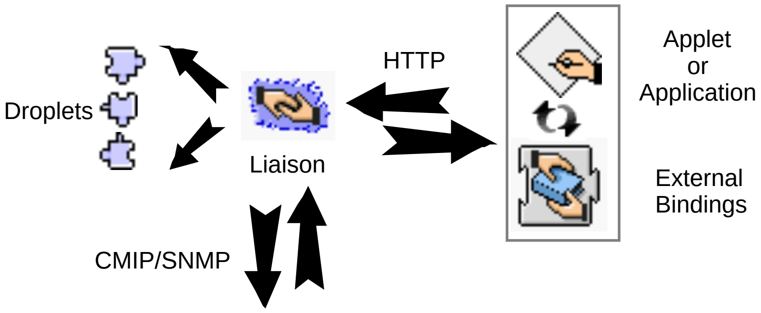


Figure 29. Liaison External Bindings

Liaison droplets handle the HTTP requests seen in the previous section. For each management request type, for instance `/SNMP/GET`, there is a droplet responsible for handling it. Management applications that use external bindings are linked with a shared library containing the bindings (in the case of C/C++ bindings) or compiled with them (in the case of Java). For the sake of simplicity, bindings described in this section are written using Java. Similar considerations can be made for the C++ bindings that come with the basic Liaison configuration. The class hierarchy is quite simple, see following figure.

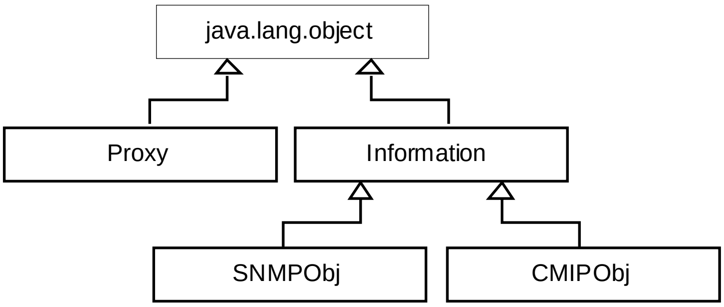


Figure 30. Java-bindings Class Inheritance Hierarchy

NOTE
The diagram notation used throughout this thesis is covered in “Diagram Notation” on page 193.

The class `Proxy` is responsible for handing communications with Liaison. It transparently sends requests and receives responses. The class `Information` contains information relative to the request and to the response(s), stored in an object of class `java.util.Hashtable`. This information is passed as input parameters to an instance of class `Proxy`. Subclasses `SNMPObj` and `CMIPObj` implement certain 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 `Information` and `Proxy` classes and must be regarded as pure facilities. Requests can have single or multiple responses. When multiple responses are returned, they are inserted in a `java.util.Vector` that is returned as output parameter. In the case of a 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.

```
CMIPObj cmip;
try {
  cmip = new CMIPObj("MIBCTL" /* MIB name */);
  cmip.UseProxy("adl.zurich.ibm.com" /* Host where Liaison is running */);
  cmip.SetObjectClass("system");
  cmip.SetObjectInstance("genericNetworkId=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); }
```

Figure 31. Code Fragment that Retrieves the `systemTitle` Attribute Value from an OSI Agent

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

The `Information` class and its subclasses `SNMPObj` and `CMIPObj` greatly simplify and reduce the amount of code users have to write:

- a `SNMPObj` or `CMIPObj` object represents a reference to an instance independent from the operation that will be issued: this allows different operations to be issued 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 Liaison, which will then use the defaults;
- default values are expressed using empty (“”) values instead of using special flags or data structures.

In addition, this solution saves bandwidth because only the necessary attributes are exchanged between Liaison 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 metadata information to be retrieved and object identifiers to be converted from numeric to symbolic form and vice versa.

NOTE

For a complete reference of the Java/C++ bindings, see “Java/C++ Bindings” on page 194.

The decision to base the bindings on Liaison 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 Liaison in every network management application. Moreover, in the case of CMIP, Liaison should be installed by those who install the stack and the OSI agent, if any, and Liaison users should not be responsible for configuration or maintenance tasks. Nevertheless, Liaison is quite compact and, thanks to its droplet-based architecture, it can be installed, replicated on various hosts, and tailored easily using very little space. In total, the Java classes just described require about 8 Kb. This allows them to be easily used inside applets downloaded by remote web browsers.

As C++ bindings are very similar to Java bindings, no special considerations are necessary. C bindings instead are slightly different. *C bindings* provide a set of functions (the total size of C bindings is about 30 Kb) that allow users to take advantage of CMIP and SNMP from within the RAD environment (see “Rapid Network Management Application Development” on page 134). The bindings are quite small because they rely on the functionality of Liaison, which is supposed to run on a machine reachable from the network. The bindings are multithread-aware and perform memory management. In other words the bindings include a simple garbage collector, which ensures that the strings passed/returned from/to the application are correctly freed. This feature also simplifies application development because programmers do not have to allocate/free the memory of the strings used to communicate with the bindings and makes the application more robust because it prevents the application from crashing due to bad memory management. Furthermore the parameters passed to the bindings are carefully verified in order to eliminate the risk of crashing the entire application if a bad value is passed to the bindings.

The overhead incurred by Liaison (mapping C bindings⇔URL⇔Liaison⇔CMIP/SNMP) compared to that of a direct C bindings⇔CMIP/SNMP mapping is less than 10-15% and is significantly lower in the case of a multithreaded application in which requests are issued concurrently. It is worth mentioning that the use of Liaison has many other advantages with respect to the latter solution:

- the size of the bindings is very small because they basically contain the garbage collector and a set of functions based on C++ bindings (i.e. the actual processing is performed by Liaison);
- Liaison handles the metadata, stores network events on behalf of the bindings (event reports and traps) and handles the communication with the OSI stack (in the case of CMIP). This helps keep the binding structure simple and uses very little memory on the application side;
- multiple Liaisons running on different machines can be accessed by a single application based on the C bindings in order to exploit Liaison's distributed architecture and to deliver the performance the application requires [Sloman94];
- bindings are very simple, do not rely on a specific OS, and can run on almost any platform. Thus they are likely to support heavy protocols such as CMIP on platforms without OSI support and to manage networks using machines of limited computing power;
- bindings can be easily modified and rewritten in another programming language (SmallTalk, FORTRAN, Objective C) to allow users to work in their favourite language. They also facilitate the integration of management capabilities into existing legacy code.

The following table compares this work with similar efforts.

	Java/C++ Bindings [Deri96b]	Tcl-MCMIS [Pavlou96]	Scotty [Schönwälder95]	XMP [XMP]	GOM [Ban97]
Application Size	Small	Medium	Medium	Medium/ Large	Small
Object Oriented	Yes	No	No	No	Yes
Ease of Use	Easy	Easy	Easy	Difficult	Easy
Typing	Weak	Weak	Weak	Strong	Weak
Type Checking	Runtime	Runtime	Runtime	Runtime	Runtime
Supported Object Models	CMIP/ SNMP	CMIP	CMIP/ SNMP	CMIP/ SNMP	CMIP/ CORBA
Language Bindings	Java/C/C++	TCL	TCL	C	Java/C++[a]
Data Representation	String	String	String	XOM	GOM (15 Types)

Table 9. Java/C++ Bindings vs. Similar Solutions

	Java/C++ Bindings [Deri96b]	Tcl-MCMIS [Pavlou96]	Scotty [Schönwälder95]	XMP [XMP]	GOM [Ban97]
Metadata Access	Yes	No	No	Implement. Dependent	Yes
Prerequisites	None. (Java VM for Java bindings)	TCL	TCL	XOM/XMP	ORB

Table 9. Java/C++ Bindings vs. Similar Solutions

a. In future versions, any language for which CORBA bindings exist.

This table shows that the proposed solution is preferable over the listed alternatives in much important aspects as application size and ease of use. Other solutions based on TCL, despite their simplicity and their similarity with the approach described here, have a larger 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 into the web and Internet. This topic is covered in detail in the section "Rapid Network Management Application Development" on page 134.

5.5.2. CORBA Interfaces

With the growing impact of CORBA in the telecommunications industry sector, the need has arisen for CORBA to manage CMIP/SNMP agents. As the CORBA object model is easier to learn than CMIP and SNMP, anyone who is able to create CORBA applications can immediately use services offered by CMIP/SNMP agents, given a CORBA interface to them, without specific knowledge of CMIP or SNMP [Dittrich96]. It is the author's opinion that this asset will become a widespread need as more applications in the telecommunications business will be programmed in CORBA. Given the large investment of carriers in CMIP/SNMP, however, the need to manage CMIP/SNMP agents will continue in the future. If CORBA can be employed to transparently manage these agents, then a smooth transition/ cooperation between the two worlds can be achieved. For this reason a company that intends to move to CORBA needs a way to access their legacy agents, and, maybe, gradually to phase them out and replace them with CORBA applications [Quinn93].

There are several approaches, some orthogonal, some overlapping, that use CORBA for CMIP/SNMP management (see "Static Mapping" on page 66). In "OMG Network Management" on page 57 the XoJIDM approach is outlined. XoJIDM statically maps the management information into CORBA datatype, resulting in a large amount of generated code and data structures. Other approaches such as GOM (see "Dynamic Mapping" on page 68) use a semidynamic approach, which exploits a metadata repository and which maps the management information in a set of basic datatypes, which are then composed to form more complex types, just like ASN.1 does. In this section a novel, fully dynamic approach is covered and compared with the other two approaches just mentioned.

Based on external bindings, some CORBA interfaces to the CMIP/SNMP protocols have been defined. An important design choice has been not to map each CMIP/SNMP object to a CORBA object as seen in other translation methods, but to map the CMIP/SNMP protocols to CORBA in a very generic way. This choice has been motivated by the following reasons:

1. the ability to fully support CMIP/SNMP from CORBA;
2. low complexity and high flexibility since there is no need to generate new CORBA classes for new CMIP/SNMP objects to be supported;
3. user-defined abstraction level: depending on their needs users can define additional CORBA classes based on the basic ones without the complexity of having a CORBA class for each CMIP/SNMP object, even if not all of them are currently used.

CORBA-Liaison interfaces (CL) for CMIP/SNMP, defined using the IDL language, have been implemented using DSOM [DSOM], IBM's CORBA compliant ORB. As we do not rely on any specific characteristic of DSOM, similar considerations can be made for other CORBA implementations. ASN.1 datatypes, like external bindings datatypes, have been mapped to strings, hence to the native string IDL datatype. CL interfaces representing CMIP and SNMP objects, defined in a way very similar to Liaison's external binding, are depicted in the following figure.

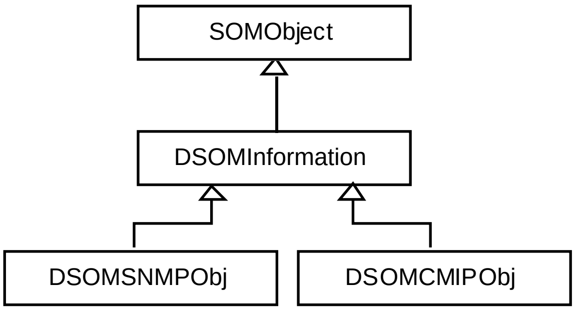


Figure 32. CORBA-Liaison Interfaces Inheritance Hierarchy

The interface `DSOMInformation` contains information relative to the request and the response(s). Internally the values are stored in a hash table, where the `attributeId` constitutes the key and the `attributeValue` the value of each table entry. The use of a hash table associated with the mapping of values into strings allows objects to be handled independently of their class and complexity. In the case of CMIP, the presentation layer or a thin layer on top of the stack converts attribute values into strings and vice versa, whereas in the case of SNMP, Liaison handles the conversion. Given that `DSOMInformation` is built upon a hashtable, it is possible to retrieve and store elements efficiently and to have only a few methods that handle all situations. In order to simplify the attribute manipulation further, the classes `DSOMSNMPObj` and `DSOMCMIPObj` have been defined. These classes simplify access to `DSOMInformation` by defining macros such as `DSOMCMIPObj::GetObjectIn-`

stance(), which are mapped into calls to DSOMInformation methods (DSOMInformation::GetAttribute("objectInstance") in this case). C++ methods of CL are almost identical to the ones defined in the corresponding class part of the external bindings, hence the stub implementation has been very straightforward. The similarity between these interfaces and the corresponding class part of the external bindings has the advantage that developers can use both DSOM and the external bindings, having to learn just one object model. In addition, code can be written once and then slightly modified to use either the C++/Java external bindings or the C++/Java language bindings of the DSOM interfaces. This is because methods and classes have the same names and parameters. The following example shows how a simple program based on the external bindings can make use of DSOM interfaces simply by adding the code shown in boldface type. The code fragment below reads the value of the attribute systemTitle of the instance netId=Net1@systemId=(name Telco) using the CMIP protocol.

```
Environment ev;
SOM_InitEnvironment(&ev); SOMD_Init(&ev); // Initialization

try {
  Liaison_DSOMCMIPObj *cmip = new Liaison_DSOMCMIPObj(&ev);
  cmip->UseProxy(&ev, "adl.zurich.ibm.com"); // Liaison is running on adl
  cmip->SetAgentAET(&ev, p, "MIBCTL" /* CMIP Agent AE-Title */);
  cmip->SetObjectClass(&ev, "system");
  cmip->SetObjectInstance(&ev, "netId=Net1@systemId=(name Telco)");
  cmip->SetAttribute(&ev, "systemTitle", "");
  cmip->CMIPGetAttributes(&ev); // Issue the CMIP M-GET request

  if(somExceptionId(&ev) == NO_EXCEPTION)
    printf("systemTitle is: %s\n", cmip->GetAttribute(&ev, "systemTitle"));

  delete cmip;
} catch(char *exc) { printf("Caught exception: %s", exc); }

SOMD_Uninit(&ev); SOM_UninitEnvironment(&ev); /* Termination */
```

Figure 33. Simple C++ Program based on CL

This is because methods and classes have the same names and parameters. Basically the only code that has to be added is related to:

1. DSOM initialisation/termination;
2. the Environment parameter required in every DSOM method call;
3. exception handling that cannot catch all the DSOM exceptions using the try/catch mechanism because DSOM may use the Environment parameter to report error conditions.

The design choice to implement DSOM stubs using the external bindings instead of wrapping the entire Liaison into a DSOM object has the following advantages:

- as the external bindings are quite light, the DSOM interface implementation is very light (about 80 Kb);
- DSOM has to be installed only by users who need to access Liaison using DSOM, i.e. applications based on external bindings do not require that DSOM be installed in order to run;
- DSOM allows objects to be created on hosts where Liaison is not installed. In such case a remote Liaison is exploited, and there is no need to have DSOM installed on the host where this Liaison runs (the communication DSOM server/Liaison is HTTP based);
- depending on the situation, users can decide to access services provided by Liaison using HTTP, DSOM or both (if Liaison were wrapped in a CORBA object, then users would need DSOM to access Liaison);
- it is possible to manage hosts outside firewalls using a local Liaison and DSOM interfaces because they are based on HTTP (DSOM cannot cross firewalls, but HTTP can).

The drawback of this solution is that every time a management operation is to be issued, communication takes place among the DSOM client, the DSOM server, and Liaison instead of having Liaison contained inside the DSOM server. In the tests we have performed, the slowdown of the proposed solution is no more than 10-20% with respect to a full integration of Liaison inside a DSOM object. Considering the many advantages of this solution with respect to total DSOM integration, this overhead is acceptable and in fact almost negligible if client applications can perform multiple operations concurrently (multithread) without incurring an active wait.

To compare CL with other approaches it is necessary to differentiate between the two relevant schools of thinking (see “Interdomain Management” on page 66):

1. Static Approach

   X/Open's Joint Interdomain Management task force (XoJIDM) works on the mapping between GDMO/ASN.1 and IDL and vice versa (only the first mapping is of interest to us here) as described in “OMG Network Management” on page 57. XoJIDM has proposed to statically translate GDMO/ASN.1 to generate code which is included by management applications that therefore know at compile time the extent of classes that they can handle.

2. Dynamic Approach

   Approaches such as CI or GOM fall in this category [Ban96], as they are not dependent on compile time knowledge because they are either string- or metadata-based.

The following table lists the major differences between the various approaches:

	CL [Deri97a]	GOM [Ban96]	Static Approaches
Mapping Type	Dynamic	Semi-Dynamic	Static
Mapping Tools	Not Needed	Off-line Compiler	Off-line Compiler
Mapping Repository	None (uses the metadata information used by the OSI stack)	Metadata Repository	C++ Code
Typing [Mathews90]	Untyped (string)	Runtime-type checked	Strong
Type Checking	Runtime (by Liaison and the OSI Stack)	Runtime (using metadata)	Compile Time
Implementation Size	Small (≈80 Kb regardless of the type/number of managed objects)	Medium	Large (includes numerous generated types/ methods)
ASN.1/CORBA Type Mapping	All datatype are mapped to a string.	Datatype are mapped to a small set of GOM types (15)	Every datatype is mapped to one or more CORBA types
CMIP/SNMP vs. CORBA Classes	N:1	N:15	N:M (N <= M)
CMIP Support	Yes	Partial (missing support for M-EVENT-REPORT and M-ACTION)	Yes
SNMP Support	Yes	No (no SNMP adapters currently available)	Yes

Table 10. Comparison of Static vs. Dynamic Approaches

The CORBA Liaison (CL) interface approach is completely untyped because all types are mapped to strings. Conversion between strings and the desired datatype of the host language (e.g. C++) has to be done by the programmer, although this is usually not necessary because in the case of CL, programmers should be accustomed to natively expressing values using strings. This may be easy for simple types such as strings or numbers, but the complexity for the programmer increases considerably for aggregate types such as structs or sequences. Also, the probability of introducing errors in user-written conversion functions increases. This trade-off, however, was accepted by CL because its main goal was to create a lightweight model for network management that is flexible (no compiled-in knowledge) and able to be integrated into the web, which uses strings as the major datatype anyway. Moreover, network management is still predominantly based on SNMP, which uses mainly atomic datatypes such as strings or integers. The programmer specifying the types in a string-based syntax which will be checked at runtime by Proxy in the case of SNMP, or by the OSI stack in the case of CMIP. Compared to the static approaches with their strong typing enforced at compile time, GOM enforces typing at runtime using metadata. Contrary to CL, which knows only the string type, it has types for representing classes (`GenObj`), attributes (`Attribute`), and values (`Val`, `Integer`, `String`, `Struct`, `Sequence` etc.). Whereas CL maps all types to strings, GOM maps them to an instance of this set of fixed types, and the static approach maps each type to a corresponding IDL type. Whereas the static approaches fully integrate the translated code into the target type system using the target's native types (e.g. C++), GOM offers an abstraction of the target's type system (ca. 15 types) as API to the users, whereas the API of CL is the single type string. In the case of the static approaches, the API client may mix types of the target system and the generated code because they are the same, whereas using GOM, native (C++) types have to be converted to/from GOM types (e.g. the `int` to instance of `Integer`). The dynamic approaches have two major advantages over the static ones: they typically produce smaller client applications and are much more flexible. As clients do not possess a priori knowledge of classes available in the system, but rather use strings or metadata, they are independent of class modifications and can continue working, whereas clients using the static approach may need to be recompiled. This is an essential asset in areas such as topology browsers and roaming agents (cf. above) that do not know all the classes they will encounter when compiled. Including at compile time a fixed set of classes may yield potentially large client applications that have to pay (in terms of size) for all the classes they carry with them even if only a few are actually used. In the dynamic approaches, when a client needs to handle a new/modified class, either the latter's metadata is dynamically loaded (GOM) or it need not be loaded if a string type represents all types (CL).

5.6. Rapid Network Management Application Development

In the past few years, one of the most frequently used terms in the software engineering field, and even in the network management field [Schmidt95], is the word ‘visual’. This term is often used to identify packages that allow a certain task to be performed efficiently and easily by visually specifying a certain activity in an interactive way. Although this term has often been misused, it frequently refers to how rapidly applications can be built using a certain tool. This is because:

- visual development is interactive and faster than classic edit-compile-run application development. Hence it allows one to immediately see the effects of a certain operation immediately without the need to build and run the application;
- visual tools are simpler to use and more powerful than traditional tools/languages, so average programmers can build very complex applications in a limited amount of time without having to be software gurus.

It is worth mentioning that visual developments also have some drawbacks in terms of flexibility if the development tools do not support all the facilities a developer needs, although they are present in the operating system (for instance balloon help). Visual development is simply an aspect of application development. To build an application visually does not necessarily mean that the application can be built quickly. Moreover, the visual construction process is no guarantee of ease of development. For this reason a new term, RAD (Rapid Application Development), has been invented to identify the tools and systems that allow applications to be built rapidly in a relatively simple fashion. RAD tools are characterised by (most of) the following properties:

- the overall time needed to build an application is minimised;
- RAD tools exploit the visual development paradigm (the reverse is not necessarily true);
- the development environment handles many low-level problems such as application linking;
- RAD tools do not generate code but rather allow the composition of basic elements in order to build the final application.

In the past few years, research and industry have invested much effort in the field of component-based applications and architectures [Joch96] [Nierstrasz92] [Nierstrasz95] [Udell94] [Deri97d]. Software composition appears to be a very promising way to cut development time and to build applications that need to be often modified as requirements change [Chen93] [Chen94].

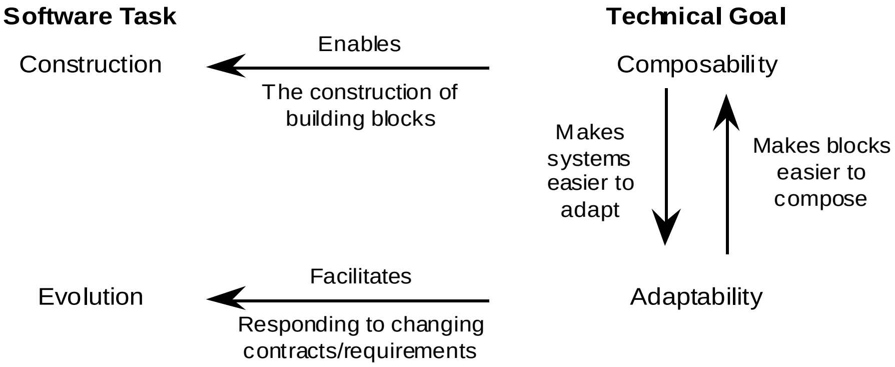

Software Task
Technical Goal
Enables
Construction
Composability
The construction of building blocks
Makes systems easier to adapt
Makes blocks easier to compose
Facilitates
Evolution
Adaptability
Responding to changing contracts/requirements

Figure 34. Issues in Software Composition

RAD tools such as IBM VisualAge™ [IBM95b] or Borland Delphi™ [Pacheco96] provide a rich set of basic components ranging from visual interfaces to remote application communication. As the component interface usually follows a well-defined guideline such as OpenDoc and OLE, it is possible to use the same component in different contexts and employ components produced by different people. Quite often the component one needs has already been built by someone else and put in a public repository.

Although RAD is becoming increasingly important in the software industry, the network management world is apparently uninterested in this new technology. This is because:

- RAD is diffused primarily throughout the PC industry, whereas most network management applications run on Unix boxes;
- RAD is usually employed to produce applications having a graphical user interface with particular emphasis on database communication and multimedia;
- RAD has to be simple enough to be used by an average programmer. Hence programming languages used by RAD tools are usually simple and (often) interpreted, so they are slightly less performant that languages such as C/C++ used in the management world.

The increasing use of PCs for everyday business has contributed to the replacement of many Unix terminals and demonstrated that a graphical interface can quite often substitute for the shell interface. The obvious consequence of this trend is that many old-fashioned character-based applications have acquired a graphical interface in order to be used not only by administrators but also by advanced end users who need to control certain critical resources for their activity. As most of these end users now have a PC, they need applications that run on their PC with the same 'look and feel' as the other applications they use, such as word processor or spreadsheets.

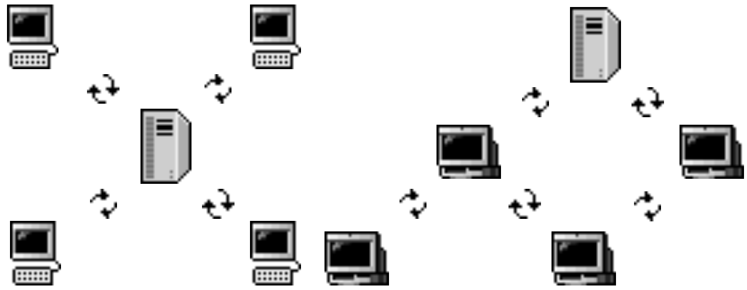


Figure 35. Trend in the Computer Industry: from Terminals to PCs

Nevertheless business-critical resources are still managed by an administrator, which often uses a character-based application that is usually faster and more powerful than an equivalent graphical user interface, especially for repetitive tasks. Owing to this tendency, it is becoming increasingly important to develop and maintain simple yet powerful applications that run on PCs and are a subset of the ones running on the corporate host and are used to manage mission-critical resources. In addition, because many end users run their applications on portable machines, it is necessary to ensure that management applications do not require large computing capabilities or rely on local network resources, which may become a bottleneck when they are accessed remotely over a slow link.

Besides this trend, users of large management platforms realised that quite often they need to develop or customise the applications part of the platform suit their needs. Owing to the high costs of training and maintenance, it is important that developers/user can develop/customise the applications they need in a relatively short time and that this task be performed by average programmers and not by highly paid specialists.

In conclusion, it is becoming necessary to rapidly and easily develop simple yet powerful management applications that run primarily on PCs, because:

- average end users are accustomed to simple graphical user interfaces whilst administrators can still use the shell interface, which is more powerful and faster but more difficult to use;
- average developers must be able to develop and maintain the application their company needs without being experts of both PC application development and network management;
- mobile computing demands simple applications that can run from remote locations over slow links.

Once the necessity of developing management applications that may rely partially on the corporate management platform is clear, it is time to decide which platform the application has to run on and how complex the user interface has to be. The following table compares three techniques that can be used to build management applications efficiently.

	HTML/VRML	Java, Java Beans, JMAPI	TCL	RAD Tools
Development/ Maintenance Time	Average/Low	Average	Average	Low
Development Skills	Basic	Average/High	Average	Average
Application Size (Disk/Memory)	Small	Average	Large	Average/Large
Platform Independence	Yes	Yes	Yes	No
Internet Awareness	Yes	Yes	Low	Low
Desktop Integration	Average	Average/High	Very Low	High
Dynamic Data Handling	No	Yes	Yes	Yes

Table 11. Comparison of Techniques for Rapid Application Development

In the first case the application is composed of several HTML/VRML pages that allow people to manage network resources using a basic user interface. End users interact with HTML elements such as buttons and menus, and an HTTP server application interprets the user commands, which have been mapped transparently to URLs. The developer is not responsible for the user interface because the Web browser is the environment in which the application runs, hence it is the one that handles the graphical appearance. One advantage of this solution over the other two proposed approaches lies in the simplicity of HTML/VRML, which makes it suitable whenever the interaction user-management interface is limited. This is because HTML/VRML are information description languages, which cannot handle dynamic data because every time the data changes, a new HTML/VRML page has to be generated. Aside from this limitation, HTML/VRML can be easily handled by basic applications started by the HTTP server and built upon simple tools such as the ones provided by the standard Unix environment. Java and TCL are two programming languages available on almost every platform. Java is a full-featured object-oriented programming language, whereas TCL is an interpreted scripting language. Both languages are network-aware and allow developers to build applications with a graphical appearance in a relatively short time [Wayner96]. Although platform independence is usually regarded as a positive feature, it is worth mentioning that this characteristic prevents Java/TCL applications from fully interacting with the environment in which they run, namely the desktop. Java/TCL applications can hardly use local resources, print on one's favourite printer, or drag elements to the trashbin because they cannot rely on specific characteristics of the environment in which they run. Nevertheless the previous statement has to be slightly changes since with the introduction of JavaBeans, Java applications are now more integrated in the desktop although their integration is still limited. Beside the desktop integration issue, Java development tools are still quite primitive and unreliable, and hence difficult to use by average programmers who are not familiar with object-oriented programming, threads, and networks. In the case of TCL it is fairly difficult to build relatively large applications due to the intrinsic language limitations. Although this situation is likely to change in the near future, Java/TCL are not yet suitable for developing large applications that have to interact closely with the local environment and be developed by average skilled programmers. Applications developed using a RAD tool are very simple to develop and maintain because the development environment masks their complexity and helps the developer during the graphical construction of the application. In addition these applications are not meant to be platform-independent, so they can fully exploit the local environment. In conclusion, RAD tools are preferable to Java/TCL whenever emphasis is placed on development complexity/cost and desktop integration rather than on portability.

The following sections will show three methods to rapidly build applications using the facilities provided by Liaison. Developers will select the appropriate method based on user requirements.

Development of HTML/VRML-based Management Applications

HTML/VRML applications are built by exploiting the basic HTML/VRML-based CMIP/SNMP management facilities offered by Liaison. Developers can:

- create HTML pages that contain HTML anchors pointing to URLs defined in a format accepted by Liaison;
- develop new droplets or applications to implement any missing functionality;
- exploit the shell commands provided by Liaison and based on the external bindings that allow simple CMIP/SNMP operations to be performed.

As HTML is quite powerful and easy to use, average programmers can rapidly create applications without having to learn complex development tools but simply by composing URLs.

Development Management Applications using Java/TCL

Java external bindings enable the development of Java applets/applications in a simple way using a string-value representation. These applets are able to manage CMIP/SNMP resources exploiting Liaison’s services. Unfortunately the creation of Java applications is still rather complicated and based on the craftsman paradigm that everything has to be custom built for a certain task. Recently, the Java community acknowledged that it is necessary to enable the creation of software components, called beans (see “Java Beans” on page 42) that can be reused for various applications. Although this seems the preferred way to create Java components, the beans are not extremely simple to develop and do not allow developers to build applications visually (as of today). It is the author's opinion that it is necessary to enable the visual composition of Java applications and that the classic edit-compile-link time can be reduced by creating a development environment in which the modifications can be tested while the application is being developed. In any case there is a large variety of Java development tools available on the market that facilitate the visual creation of applications and allow graphical user interfaces to be built in a matter of minutes. Similar considerations can be applied to the development of C++ applications that exploit the C++ version of the external bindings.

In the case of TCL, it is possible to attach management capabilities by using the C version of the bindings, seen above. Although TCL comes with the Tk toolkit, which significantly simplifies the development of graphical applications, it suffers from the lack of a standard way to reuse code or to produce reusable components in order to reduce development time/effort. This limits reuse to the library level, which is quite primitive and not always straightforward [Biggerstaff89]. Nevertheless the ability of TCL to write code by binding small programs into large applications is a way to reuse code to some extent and benefit from the many TCL programs/libraries freely available on the network. Finally the advanced GUI facilities offered by Tk make it reasonably simple to build network graphical applications in a fraction of the time it takes to do so with conventional development tools and languages.

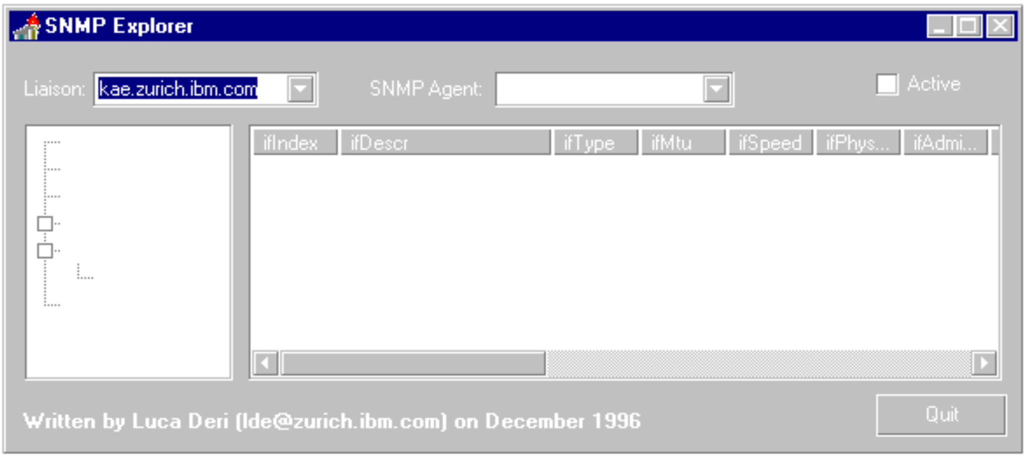
Development of Management Applications Using RAD Tools

The development of applications using RAD tools is probably the most interesting and promising way to build management applications rapidly because RAD tools:

- allow development time to be reduced dramatically;
- are so simple to use that even an average programmer can create an outstanding application.

An application that formerly would have required several months of a skilled programmer’s time can now be done by an average developer in a matter of weeks. Unfortunately RAD tools have hardly ever been used in the area of network management, basically because of the lack of facilities necessary to “glue” the RAD environment to the management environment, which has never be done before. In order to fill this gap, Liaison provides C external bindings, which are basically a C interface on top of the C++ bindings packaged in a DLL (Dynamic Loadable Library) in order to be easily called from within the RAD environment (see “Application Side Bindings” on page 125). C bindings come with a set of files, which can be used with widely used Pascal/Basic environments, and some basic examples, which allow developers to become productive quickly.

C bindings provide a set of functions that allow developers to take advantage of CMIP and SNMP from within the RAD environment. The bindings are multithread-aware and handle memory management. In other words the bindings include a simple garbage collector, which ensures that the strings passed/returned from/to the application are correctly freed. This feature also simplifies the application development because programmers do not have to allocate/free the memory of the strings used to communicate with the bindings and makes the application more robust because it prevents the application from crashing due to bad memory management. In addition the parameters passed to the bindings are carefully verified to eliminate the risk of crashing the entire application in case a bad value is passed to the bindings. In order to demonstrate how easily and fast applications can be developed using RAD tools, an example is shown in the following figure.



SNMP Explorer

Liaison: kae.zurich.ibm.com

SNMP Agent:

□ Active

Written by Luca Deri (lde@zurich.ibm.com) on December 1996

Figure 36. Simple SNMP MIB Explorer

This simple SNMP MIB explorer allows one to manipulate the SNMP MIB of a remote host by exploiting the services of Liaison, which can run on a local or a remote host. This application has been written by an average programmer in a couple of hours and it has the look and feel of the operating system on which it runs. The same application written using different tools or a different language would have been much more difficult to write and would have required a much more skilled programmer with deep knowledge of the underlying operating system. Similar applications supporting the CMIP protocol and resembling the one just described have also been developed by the author.

Conclusion

Liaison provides three ways to develop management applications, including a new method for rapid application development. Developers can select among the following ways to create applications, depending on the user needs and the global application requirements:

- HTML/VRML for simple static applications accessible remotely via inexpensive Web browsers;
- Java/TCL for developing of platform-independent applications that make use of external bindings in order to add management capabilities without incurring high costs in terms of development time, application complexity, and size (quite important if the application has to be downloaded on demand);
- RAD tools, which allow average programmers to develop outstanding applications at a fraction of the time/cost needed to develop the same application using traditional methods.

The era in which "one management platform does everything" is about to end and will be replaced with one that enables users to build management applications they need easily and rapidly. This does not mean that large and powerful management platforms will disappear because such applications constitute the backbone of corporate management systems. It means that in the future, users will increasingly demand tools that allow them to write the applications they need, tuned to their environment, instead of delegating this task to specialised and expensive developers. One reason for the limited diffusion of management tools lies in the cost of the tools and their extreme complexity. This work is a small contribution towards the construction of simple and powerful network management tools that can be used by many people and not only by rich or large organisations but also by universities and small institutions.

5.7. Final Remarks

Liaison traced a novel promising way to develop management applications. Contrary to most current management applications, Liaison does not integrate management functionality in a complex framework. Instead it offers several services accessible concurrently from remote locations using the HTTP protocol. Liaison is the only entity that handles directly the management protocols and its idiosyncrasies. Management applications based on Liaison range from simple web pages to complex graphical applications. Users can decide, based on their needs, which level of sophistication and ease of use they want. In any case, Liaison-based applications do not demand significant computing resources, can be run from remote and can scale up by simply issuing concurrent requests to multiple Liaison instances. The following table summarises the Liaison effort.

	Liaison
Space Requirements	< 1 Mb RAM, about 1 Mb disk space (depending on the configuration and on the platform).
Source Code	16 Klocs (mainly C++, some C/Java code).
Press Review	PC Week, Linux Magazine, Apple Developer Web Site.

Table 12. Liaison at a Glance

	Liaison
Users	> 2,000 Worldwide (in constant growth).
Currently Supported Platforms	AIX, OS/2, MacOS, Linux, Win95/NT.
Availability	Free of charge from http://misa.zurich.ibm.com/Webbin/, http://www.alphaworks.ibm.com/.
Research Contribution	Liaison is the first management application that: • allows users to manage networks using HTML/VRML; • allows users to deliver a real, seamless, integrated multidomain management application[a] supporting CMIP, SNMP, and CORBA; • enabled users to create simple, light, efficient management applications; • brought software technologies such as software components into the management world; • introduced HTTP-based management; • demonstrated that CMIP and CORBA management no longer have to be considered a challenge; • thanks to Yasmin, has been able to bring management capabilities to platforms considered unsuitable for this task (for instance MacOS).
Papers	13 papers/disclosures have been written on this topic, including an Internet Draft.

Table 12. Liaison at a Glance

[a] Note that XoJIDM and GOM are significantly incomplete and that other approaches shown in earlier sections do not fully cover interdomain management.

Besides all these features, there is still a significant amount of work to do. The most important activities concern providing:

- secure droplets (for instance digitally signed droplets) to avoid security problems;
- a more robust droplet architecture that prevents problems in a droplet (for instance an operation which causes the droplet to crash) from affecting the entire system;
- support for protocols other than HTTP, such as IIOP [IIOP] which is part of CORBA version 2;
- extension of the Java external bindings to support the same functions supported by the Java Management API (JMAPI) [Sun96b]. This will allow JMAPI applet to exploit Liaison's services.

Probably the most important achievement of Liaison has been the demonstration that it is possible to build efficient management applications which seamlessly integrate the three major management models (OSI, SNMP, and CORBA). In addition, because Liaison is freely available for download on many platforms, many people have been able to experiment with it for free. This is quite important especially considering that in OSI and CORBA (this does not apply to SNMP) it is rare to find implementations with which one can play freely, without having to purchase an expensive management platform.

6 Validation

This chapter validates Yasmin against the thesis requirements previously identified. The reader interested in a comparison of Liaison with relevant efforts undertaken in the field of management systems is referred to the preceding chapter, and the section “Evaluating Liaison” on page 203.

6.1. Thesis Validation

In order to validate this work it is necessary to return to “Thesis Requirements” on page 23 to see which requirements are satisfied and, if not, why.

Mandatory Issues

The following are the mandatory requirements placed on Yasmin.

1. Extensibility

   Yasmin’s extensibility has been achieved by splitting the architecture in two parts: kernel services and user services. The kernel services are general and are used by all applications based on Yasmin. The user services contain the application-dependent services and are implemented using droplets. As droplets can be added (at runtime) it is easy to extend applications by adding new services and functionality which will then be folded into a droplet. This mechanism allows extensibility in addition to facilitating application tailoring, which can be done by adding/ removing droplets while the kernel application remains unchanged.

2. Evolution

   Yasmin achieves evolution, like extensibility, through droplets. Applications that must modify their behaviour due to changing requirements can do so by replacing droplets with new droplet versions. Diagonally, because droplets are linked at runtime with the core application, they can migrate to other networked machines, thus allowing the application topology to evolve (at runtime) and be adapted to new network topologies.

3. Ease of use and development

   Ease of use is determined by various factors. In the field of open distributed systems, it is strongly influenced by the way installation, configuration, and tailoring are performed. As Yasmin is based on the idea that

[Taligent95a] resources should notify their presence to the operator and not the other way round, Yasmin comes with a directory service which permits locating relevant network resources on behalf of the user, hence minimising the configuration. One side effect of this design choice is that configuration files never become outdated. They are virtually absent because the information is fetched automatically from the network. Other ease of use aspects concern the provision of on-line help, whether metadata can be accessed and the ability to prevent wrong requests from being issued. As far as installation and tailoring are concerned, they are facilitated by the droplet paradigm and by the fact that Yasmin-based applications automatically load new droplet versions without human intervention.

The ease of development is guaranteed by the fact that the droplet interface is rather simple, thus limiting the amount of information that has to cross it. This facilitates programming because the simpler programming is, the fewer errors are made and the more productive the programmer is. In addition, Yasmin-based applications do not statically link droplets, hence the traditional final linking step is not necessary. Moreover, because droplets are individually linked, the link time is reduced and a modification in a droplet does not influence the entire application, which remains untouched.

4. Distributed environment support

   Yasmin integrates facilities for peer communications and allows remote services to be invoked. Based on user needs, it is possible to configure applications such that ‘heavy’ services are performed on powerful hosts or the load can be distributed over various instances of the same application by assigning droplets appropriately. It is also possible to exploit the network to improve application reliability through service replication by distributing the same droplet to multiple applications. In this way, if an application that implements service X crashes, another remote application can still offer service X.

5. Use of software components

   This has been achieved by basing Yasmin on the droplet paradigm.

6. Promotion of reuse

   Design reuse has been achieved though the use of droplets. They allow a generic application which implements the core services to be developed, and then can be reused on different fields by adding droplets.

   Code reuse is achieved with droplets too. Moreover, because service requests are issued using platform-independent protocols (in the case of Liaison, the protocol is HTTP) it is possible to reuse services present on hosts which use different operating systems. This is a more general definition of reuse based on service reuse rather than on code reuse.

7. Efficient resource utilisation and ability to run on environments of limited resources

   Loading droplets on demand and delegating computational intensive services to remote applications allow applications to run on hosts of limited computing power. Moreover because droplets are loaded on demand and unloaded when no longer in use based on a certain user policy, Yasmin-based applications make wise use of system resources and avoid using unnecessary resources, and this, contributes to enhance global system performance.

8. Portability and genericity

   The use of personalities facilitates code portability and allows system-dependent code not to cross the personality boundary. Platform-specific features such as multiprocessing and multithreading, when available, are used by the personality layer, which is responsible for exploiting platform-dependent features while not exposing them to the rest of the application.

9. Based on open standards

   Yasmin is fully documented hence it is not a black-box which uses proprietary technologies. Yasmin does not define new protocols nor does it specifies the set of protocols that is suitable for a certain activity. Liaison, a Yasmin-based application, is based on open protocols such as HTTP and it fully supports management protocols without introducing 'implementation shortcuts' like many programmers do whenever a certain management protocol/service is difficult to implement. This proves that applications built on open standards can be built using Yasmin.

10. Internet-awareness

    Yasmin-based applications, such as Liaison, support remote communications and implement them using HTTP, an Internet protocol. This also allowed a new management paradigm, HTTP-based management, to be created and which is now quite well accepted in the industry.

11. Slim and efficient architecture

    The architecture is very slim because all non-necessary services are implemented inside droplets. Moreover the kernel is very efficient because it merely glues the various droplets together without adding further overhead. The efficiency of the architecture is demonstrated in "Evaluating Liaison" on page 203, where the performance of Liaison is analysed and compared with that of similar applications.

12. Full support for (management) standards

    Yasmin does not impose particular limitations on the way applications have to be implemented. This facilitates the implementation of management standards that are quite complex, and hence difficult to be fully supported. The proof of this claim is Liaison, which has supports HTTP, CMIP, SNMP, and CORBA without any limitation and with good performance (see "Liaison: Yasmin at Work" on page 105).

13. Scalable and performant applications

    Scalability of Yasmin-based applications is achieved by replicating the same application on networked machines, i.e. by distributing the load by transparently rerouting (delegation) requests. This allows the necessary performance to be achieved. Even if a single application is used, the performance achieved is satisfactory even on small machines as demonstrated by Liaison.

14. Independence from specific technologies and languages

    The fact that Liaison has been ported on very different platforms demonstrates that Yasmin is independent of specific technologies. Moreover, droplets do not rely on specific C++ features and have been implemented using a different language such as Java [Ban95].

Optional Issues

1. Full distributed environment support

   As mentioned above, Yasmin fully exploits distributed environments and allows droplets to be migrated and replicated using the peer services part of the architecture.

2. Security support

   Security is a very complex field encompassing several aspects ranging from peer authentication to encrypted communications. Because security support varies according to the communication protocol being used, its support has to be included in Yasmin's communication services (see "Communication Services" on page 98) that deal with all the communication issues. This is because the structure of Yasmin-based applications should not vary whenever the communication protocol or the security features being implemented vary.

   In the case of Liaison the communication protocol being used is HTTP. Liaison implements support for HTTP authentication, peer authentication and encrypted/secure communications via SSL (Secure Socket Layer) an industry standard defined by Netscape. This is the maximum security support currently defined for the HTTP protocol.

3. Ease of installation and tailoring

   As mentioned above, this has been achieved through the use of droplets.

4. Support for visual application development

   The author has experimented with visual development tools for the creation of management applications, as discussed in "Rapid Network Management Application Development" on page 134. The flexibility of Yasmin/Liaison allowed some glue software in the form of shared libraries to be developed in order to interface rapid/visual development tools with the management world. This allowed existing rapid/visual development tools to be used for developing management applications rather than having to create new ones merely for this purpose.

Secondary Issues

1. Exploitation of specific platform features

   This has been achieved by the use of personality, which allows platform-specific features to be exploited while preventing platform-dependent code from crossing the personality boundary.

2. Integration with commercial products/frameworks and ability to embed components into commercial frameworks

   This issue has not been dropped because none of the available components listed in "Component-based Architectures" on page 33 is suitable for use in the context of network management. Moreover the integration of Yasmin in commercial frameworks presents certain problems. This is because Yasmin attempts to solve problems typical of conventional frameworks, hence the integration of Yasmin into commercial frameworks completely jeopardises this work.

6.2. What's New in Yasmin?

In section 3.5. on page 68 it has been shown that none of the existing architectures and frameworks available on the market satisfy all the requirements listed in section 2.9. on page 23. This does not mean that Yasmin has not adopted existing principles or techniques. In fact during Yasmin's design phase interesting pieces of existing works have been adopted and combined with novel ideas and concepts. The most relevant ones will be given later in this section.

Existing Principles and Techniques Used in Yasmin

- The personality principle of the Mach™ kernel [Accetta86], in which a thin layer masks operating system-dependent (hardware-dependent in the case of Mach) services, thus allowing applications to be developed independently of the underlying operating system (see "Personality Abstraction Layer" on page 92).
- Scalability through the replication on the network of (Yasmin-based) server applications accessible from remote clients [Lindenberg90] [Sawitzki92]. An application similar to the CORBA trader [OMG95] is then responsible to transparently distribute client requests to servers according to well-defined policies (for instance, load balance). Client performance is then directly proportional to the number of the available servers (see "Scalability" on page 78).
- The proxy principle/pattern as described in [Shapiro86] and [Gamma94] (see "Welcome to Liaison" on page 109) which accesses services/resources on behalf of other applications shielding them details concerning their implementation and location.

- Use of cooperation [Grégoire94], delegation [Goldszmidt93] [Yemini91] [Johnson91b] and subcontracting [Hamilton93] allows droplets to collaborate in order to obtain a global result (see "Yasmin's Design Choices" on page 98).
- The system is able to transparently locate, register and load droplets without any user intervention or configuration just like OpenDoc does [Apple95] (see "Droplet Manager" on page 93).

Principles and Techniques Introduced by Yasmin

- Ability to replace binary software components at runtime (see "Yasmin's Design Choices" on page 98) in order to update and extend the global application behaviour while the application is running.
- An application must be minimal (microkernel application): every service or functionality provided by the application must be implemented inside components which are glued using the basic facilities provided by the application (see "Conceiving Yasmin" on page 73). This allows the whole application behaviour to be extended and modified while the application is running.
- Delayed events (see "Event Manager" on page 95) which allow periodic tasks and activities that have to be performed at a certain time without having to maintain a separate thread of execution for each of these tasks. Repetitive tasks are quite common in network applications hence the use of delayed events becomes very important as the number of these periodic tasks increases. This is because the number of threads per application is limited hence the classic way of implementing these tasks using threads cannot be used as explained in "Event Manager" on page 100.
- "Firewall-like" droplet interface (see "Droplet Interface" on page 81) which prevents external entities from having access to droplet internals. This also prevents direct dependencies among components because services provided by droplets can only be accessed via managers, part of the architecture (see "Service Manager" on page 96 and "Resource Manager" on page 97). The reader interested in further details about the implementation of the droplet interface is referred to the section "From Theory to Practice: Implementing Droplets" on page 207.
- Simple component service signature which discourages developers to pass complex datatype to services, hence forcing them to split complex services into simpler cooperating services according to the well-known 'divide et impera' principle (see "Service Manager" on page 96).

6.2.1. Conclusion

This section has shown that Yasmin is a new architecture for software applications which combines existing ideas with new concepts. This is a very common approach in the software engineering world, in which most of the innovations are not completely new but are an evolution of well-known principles wisely mixed with novel concepts.

6.3. Further Remarks

The goals contained in the section "Research Goals" on page 28 have been fulfilled because:

1. Liaison has demonstrated (in particular see "Rapid Network Management Application Development" on page 134) that network management applications can be developed with a relatively low effort by averagely skilled programmers. This result has been achieved also because Liaison has greatly simplified the management protocols by defining a simple object model based on strings. In addition the use of external bindings combined with the scalability offered by Liaison allow efficient applications that exploit the distributed environment to be developed.
2. The use of the HTTP protocol to transport management requests and the ability to manage networks using the web demonstrated that web-based management is both feasible and very promising.
3. Yasmin makes extensive use of cooperation and delegation. This allowed the distributed environment to be exploited because:
    - cooperation avoids duplicating resources and services, thus encouraging reuse of existing resources, either local or remote;
    - delegation is an effective way to avoid services duplication; an entity (for instance the proxy) is able to serve N remote peers, hence to preventing duplication of the same service N times;
    - delegation and cooperation are used by Yasmin to achieve scalability in a distributed environment (see "Scalability" on page 78);
    - applications based on external bindings exploit the client/server paradigm, which is suitable for a distributed environment.
4. external bindings and CI interfaces demonstrated that interdomain network management is feasible and simple when a suitable architecture is selected (Yasmin in this case).

For a comparison of Yasmin with other component-based architectures see "Comparison with Other Architectures" on page 101. A similar comparison has been compiled for Liaison and can be found in the chapter "Liaison: Yasmin at Work" on page 105. Finally, note that:

1. Yasmin, thanks to the use of droplets, has solved problems typical of management applications such as:
    - Ability to modify and extend applications at runtime.
    - Resource usage
Droplets are loaded on demand and unloaded according to a specified policy (for instance limit system resource utilisation).

- Development time
  Droplet modifications do not affect the entire system, hence only the pertinent droplet has to be modified, compiled, and linked, saving significant development time with respect to rebuilding the entire application.
- Tailoring and configuration.

2. Liaison has demonstrated that:
   - efficient, slim, simple, and powerful management applications can be created;
   - the distributed environment can be exploited for the purpose of management;
   - Internet technologies can be used profitably in network management;
   - development of management applications for interdomain management is feasible and hence no longer a challenge.

7 Conclusion

7.1. Lessons Learned

This work has been and still is a challenge. It has integrated concepts from the software engineering world with others derived from network management and human-computer interaction experience. It is not revolutionary in the sense that, besides the droplet and the delayed event concepts, it has not invented something new such as a new programming language or a new management protocol. The strong point of this work resides in having distilled and enhanced known ideas and concepts such as software components, and having applied them for the first time to a new field, open distributed systems (see “What’s New in Yasmin?” on page 147). Mixing known technologies with totally new solutions (as far as the author’s knowledge, droplets are the only type of binary software components that can be reloaded and added at runtime to running applications) allowed a new type of application to be created, which has contributed solving typical problems that affect many applications such as being monolithic and difficult to extend.

The section “Yasmin’s Design Choices” on page 98 explains how key architecture components have been designed, what other possible design alternatives there have been, and why have been dropped. This section is very important not only in terms of describing the motivation for some design choices but also in terms of sharing with the reader some lessons learned during Yasmin’s design.

Yasmin and Liaison have shown that the ‘divide et impera’ principle is still valid. They have demonstrated that the complexity of activities such as network management can be mastered if such a complexity is not able to penetrate as far as the user application but is confined to the lower layers. In other words, the underlying system on which the user application is based must shield the application from the complexity of the underlying protocols. In addition, whatever service is provided by the underlying system has to be accessible from both local and remote clients when security allows this. The use of cooperation and delegation has greatly prevented existing functionality from being duplicated and allowed slim and efficient applications to be built.

This work demonstrated that often somebody else has invented the wheel we need. Computer science is so broad that is likely that the solution we need already exists and we merely have to adapt it to the current problem. This is probably a broader definition of reuse (or recycling).

7.2. Which Results can People Reuse in Other Projects?

Given the broad scope of this thesis, which encompasses two distinct areas such as architectures and management of open systems, it is worthwhile to summarise those results of this research that can be applied to other projects.

Architectural Concepts

1. Architectures, as applications, are never completed. In order to support evolution, an architecture for software applications must clearly separate mandatory and optional components. Mandatory components implement the set of services that are necessary to every application built using the architecture. Optional components implement application-specific functionality. Mandatory components constitute the application kernel, whereas optional ones are implemented in a way that allows them to be loaded on demand by the application and replaced while the application is running.
2. It is very difficult to know in advance a) the problem size, b) future requirements, and c) the complete set of services an application must support. The consequence of this is that every architecture must provide a way to extend existing applications by adding new components.
3. Architectures last longer than applications that were developed using them. Therefore an architecture must be fully specified and allow room for future additions. This is because as soon as the architecture is used to implement applications/frameworks, it is too late to change it without having also modifying the applications/frameworks which make use of it. An application, on the other hand can be developed in several stages.
4. Generic users do not exist. Every user always has very specific needs. Therefore an architecture must allow applications to be built that can be customised/adapted/tailored by the final user and rather than only by developers.
5. Architectures must be suitable for building applications that offer the implemented services/functionality to other applications. This allows incremental application development and avoids existing services/ functionality to be reimplemented every time a new application needs to be developed.
6. Modern users are accustomed to application ease of use. Developers increasingly demand architectures/frameworks that render their task easier.

7. An architecture that delegates too many key decisions to the developer is not good. This is because this policy will increase the probability that two applications developed using the same architecture cannot interoperate or share/reuse functionality/services.
8. Performance is always an issue, although opinion regarding this aspect diverge. An architecture that produces elegant applications with poor performance is useless. Usually the best compromise is to balance design and performance in order to obtain a good design with good performance.
9. Architectures, as applications, must not be monolithic thus allowing developers to incrementally develop their application/framework using portions of it, without forcing them to implement the entire architecture at once.

Implementation Issues

1. Droplets are very general and can be profitably used in different contexts whenever:
   - different object models, libraries or frameworks need to be combined into the same application;
   - it is necessary to achieve runtime application evolution;
   - applications must provide many different functionalities most of which will be used very seldom or only by specific users; hence it is necessary to load the functionality on demand only when needed;
   - applications have many components that need to be selectively updated without having to modify the entire application;
   - applications can be split into different, autonomous parts, which can then be built/modified independently from the rest of the application.
2. Use of delayed events is suitable whenever a need exists to perform periodic tasks without having to use limited resources such as threads that can instead be used for other application tasks.
3. Portability is always an issue, hence “dirty tricks” must be avoided or limited to very specific parts of the code and not be spread through the code. A developer should not assume that an application will always run on the same operating system (version).
4. Object-oriented mechanisms should be wisely used. Their abuse can produce unpleasant side effects such as the creation of monolithic applications. A way to prevent these problems is to use droplets which can be implemented internally using object-oriented mechanisms.
5. Hardware and software resources must always be wisely used because sooner or later they will no longer suffice. Architectures and applications must take this into account.

6. An application can always be optimised. Therefore optimisation must be performed at the end of the development cycle, and code should be written in such a way that not the entire application will have to be rewritten when optimised pieces of it are merged with an existing application.

7.3. Open Issues

Owing to the time constraints imposed on a PhD thesis, this work focuses on only a subset of open systems, namely network management. Indeed it would be interesting to investigate in detail which aspects of open systems have not been covered by this work. In particular it remains to be identified how the architecture has to be extended and refined in order to satisfy requirements present on open systems that are not relevant to network management.

Concerning software components, however there are generally numerous open issues to which research has not yet been able to give general answers [Szypersky95] such as:

1. Global Application Analysis

   Component software is extensible by definition, i.e. never complete. As component-based applications similar to Yasmin are (runtime) extensible, there is no final integration phase in which all components are added to the application. Therefore a global application analysis cannot take place; it is only possible to guarantee that the components are working correctly. Nothing can be said about the global application behaviour. It would be necessary to identify the parameters to be taken into account in order to perform a limited global application analysis.

2. Application Decomposition

   What are the criteria to follow in order to decompose an application into components, thereby minimising the coupling among components?

3. Application Safety

   When a component-based application fails, it is not always simple to identify where the problem occurred because a single component may have modified the global application which then failed while performing other operations. Application safety means that invariants can be guaranteed. What are the component/global invariants, how do they have to be identified and how can they be enforced in order to guarantee application safety?

4. Component Robustness

   When a new component is added or replaced at runtime, it might happen that owing to a bug in the component, the component fails and the entire application crashes. When an application deals with coarse-grained components, it is possible to run components in separate address spaces and make them interact via communication channels (for instance pipes or shared memory). In the case of droplets or coarse-grained components, the application and the components share the same address space, so if a component fails, the entire application crashes. What are the mechanisms that prevent a global application failure? Is it sufficient to have a sort of security ring like the ones used for OS/2™ drivers [Young89], where components can be promoted as soon as their robustness has been verified?

5. Remote Component Communication

   When can a component part of a remote application migrate and attach to a remote application? What are the parameters involved in this choice (performance, available resources)? Are these parameters static or do they vary depending on the application type?

6. Memory Management

   Especially in the case of fine-grained components, component interaction leads to cross references on the level of individual objects. In a truly extensible system, it is quite difficult, if not impossible, to know when an object can be released again. Not releasing an object no longer in use may lead the application to use all the available memory. Even in the cases of COM and DSOM, which keep an object reference count, the system relies on the developer which is responsible for manipulating the counters properly. Is there a solution to this problem or is to having a system that implements a garbage collector the only way to circumvent it?

In the case of Liaison, the issues listed above can be rephrased as follows:

1. Global Application Analysis

   This is an open issue because Liaison is extensible at runtime, so a global application analysis should be done at runtime, which would not be useful (too late), however.

2. Application Decomposition

   Liaison's main application is very slim and every functionality is implemented inside the droplets. Therefore this issue can be modified as follows: how can components be decomposed? What should their granularity be? It is obvious that the smaller the granularity, the better is, because this makes the application more flexible, especially when selective components have to be replaced. Unfortunately due to this policy, the application may become more fragile because, the larger is the number of components, the more invariants each component has to respect.

3. Application Safety and Component Robustness

   These are open issues in Liaison and their solution is left to future work because there is no obvious or simple solution to the problem.

4. Remote Components Communication

   In the case of Liaison, components can be migrated quite simply by copying them into the `Droplets/` directory. To be investigated is whether it is possible to create smart applications that can decide whether it is preferable to migrate a component or use it remotely. Based on the author's experience, these smart applications can be created if and only if the droplet interface (see "Droplet Interface" on page 81) is enhanced by adding information that allows the application to decide whether it is the case to migrate the droplet. Possible extensions are:

```
typedef struct {
  [...] /* Former Droplet Interface */
  unsigned short migrationSupport, requiredOStype, requiredOSversion;
  unsigned long minMemoryRequirements, minDiskSpaceRequirements;
} DropletInfo;
```

Figure 37. Enhanced Droplet Interface

5. Memory Management

   Liaison solves this problem by delegation:

   - all the memory allocated by a droplet is local to the droplet itself (i.e. it is not passed to other droplets);
   - the droplet that allocates the memory is responsible for releasing it;
   - if a droplet X requests a service contained in droplet Y:
     - droplet X delegates to the SM to issue the service;
     - droplet Y executes (on behalf of the SM) the service request. Hence if there is some memory manipulation involved, droplet Y manipulates the memory local to the service (not droplet X);
     - if the service has to return a response, it is the responsibility of service X to allocate the memory block, which will be passed to the service request and which will contain the response.

7.4. Future Work

As in every other thing in life, this work is far from being completed. In addition to the work identified in "Further Remarks" on page 149, the following items have also been identified requiring future work:

1. Study how the droplet paradigm can be enhanced in terms of:

   - Robustness
     Droplets run on the same address space of the application that uses them. If a droplet performs a wrong operation the entire application is affected. It would be interesting to study how droplets can be turned into truly independent units that do not share the same address space.

   - Security
     Identify which security mechanisms have to be added to droplets in order to avoid applications loading/running droplets that do not satisfy the basic security requirements.

2. Enhance Yasmin's service manager (see "Service Manager" on page 96) in order to support transparent marshalling/unmarshalling in the case of remote requests.
3. Identify how new component types can be adapted and plugged into a Yasmin-based application. For instance: how can a COM component be wrapped in a droplet in order to use it from a Yasmin-based application?
4. Study the requirements for a visual application composer based on droplets which is able to compose applications using some basic droplets and to develop visually new droplets as well as the code to glue droplets.
5. Investigate how Liaison can be extended to support new management APIs such as JMAPI and further management protocols used for very peculiar activities not covered in this thesis.

8 Glossary

Architecture

See “Software Architecture” on page 161.

ASN.1

Standard OSI notation to define abstract syntax in order to allow data to be exchanged between non heterogeneous hosts.

CMIP

Protocol used by CMIS to transfer management information. It defines the procedures for information transmission between two applications (abstract CMIP syntax), the procedures for protocol control, and the conformance testing used to test CMIP implementations.

CMIS

General communication service used by OSI for system management. It is used to exchange management operations and notifications among managed objects. It defines the set of service primitives, their parameters, and the information necessary to describe each service primitive.

Component

See “Software Component” on page 162.

Cooperation

Interaction of several components to achieve a collective task. The activities of a cooperative entity are the result of the cooperation of its components, rather than of a single entity.

CORBA (Common Object Request Broker Architecture)

Architecture defined by OMG which specifies how instances contained in an heterogeneous environment can communicate with each other regardless of their physical location.

Delegation

Act of delegating certain activities (otherwise performed locally) to external entities, which perform them on behalf of the requestor.

Droplet

Software component having the following specific properties:

- it is not statically linked to the application but is loaded at runtime;

- it has the ability to be replaced (i.e. a new version of the droplet can replace a previous one) at runtime while the application is running.

Extensible System

A system that allows functionality at runtime to be added which loads only the functionality used currently. It adds further functionality only if necessary.

Framework

A framework defines an architecture by specifying and restricting the way architecture components interact.

GDMO

Notation defined by ISO in order to harmonise the management information definition. GDMO is a formal language that allows people to specify managed object characteristics such as attributes, properties, and behaviour.

HTML (HyperText Markup Language)

Simple markup language used to create hypertext documents that are portable from a platform to another. Such documents are identified by URLs (Uniform Resource Locator) and are visualised by an HTML browser.

HTTP (HyperText Transfer Protocol)

Transport protocol used to transmit multimedia documents such as hypertext documents, images and video. In particular, it is the protocol used to transport HTML documents.

HTTP-based Management

Concerns system and network management using the HTTP protocol.

Independently Extensible System

Extensible system that can cope with the late addition of extensions, possibly developed by different people in complete ignorance of each other, without requiring a global integrity check.

Interdomain Management

Management of multiple domains by means of a single domain. A domain is a system by means of which management is accomplished.

Management Domain

Set of resources that share a common set of attributes or that are managed by the same entity.

MIB (Management Information Base)

Database containing the managed objects that can be added, removed, and modified by means of management protocols.

Managed Object

Any resource managed by an open system environment. A MO represents an abstraction of logic or physical resources (for instance bridge, router, protocol elements) and their properties. Each MO is characterised by a set of attributes, operations that can be performed on the object, behaviour, and asynchronous notifications.

Multithreading

Software application with multiple concurrent execution flows called threads.

Network Management

Set of tools that allow heterogeneous networks to be controlled and managed both locally and remotely.

Object

Instance of an abstract data type characterised by the following properties: encapsulation, inheritance, and polymorphism.

Object Model

Provides an organised presentation of object concepts and terminology by defining a partial model for computation that embodies the key characteristics of such objects. The object model describes concepts useful for client application such as the use of objects, object creation and identity, requests and operations, types, and signatures. It then describes concepts related to object implementations such as methods, execution engines, and activation.

Open System

System that conforms to open standards and is proven to be portable, not relying on a specific topology and open to extensions and to changing requirements (evolution). Because a system implements open standards it must be able to interact with other systems that implement the same set of standards.

OSI Management

OSI Management concerns the set of activities necessary to control, coordinate, and monitor relevant OSI resources.

PICS (Protocol Implementation Conformance Statement)

Document provided by vendor of an OSI system or application, in which all implemented capabilities are clearly specified for each implemented OSI protocol.

SNMP (Simple Network Management Protocol)

Standard protocol used to manage Internet devices, made of SMI, MIB, and SNMP itself. SMI and MIB define and monitor the set of managed objects by using the SNMP protocol. The SNMP protocol allows management stations to exchange management information in a client/server fashion.

Software Architecture

The structure of the components of a program/system, their interrelationships, and principles governing their design and evolution over time.

Software Component

Static abstraction with bidirectional plugs, communication channels which allow the component to interact and communicate with the outside world. The word “static” highlights the fact that components are long-lived entities which can be stored in a software database independently of the applications that have used it. Abstract means that the component shields the software it encapsulates from the outside by putting an opaque boundary around it. Bidirectional emphasises the fact that other components can communicate with the component but also that the component can communicate with other components, i.e. peer-to-peer vs. client-server mode. If the bidirectional constraint is relaxed, then the component is called *plug-in*, because other components can communicate with it but not the other way round.

System Management

Set of mechanisms used to monitor, control, and coordinate OSI resources, and standard communication protocols used to transmit management information (see also “Network Management”).

VRML (Virtual Reality Modeling Language)

Modeling language used to describe a set of 3D elements usually called a 3D virtual world, which are then rendered by a VRML viewer.

Web-based Management

System and network management using web technologies.

World Wide Web

Allows people to work together by combining their knowledge in a web of hypertext documents. It is a medium for communication using computers as a largely invisible part of the infrastructure. The web was pioneered by an English engineer who envisioned a system able to handle various Internet protocols as well as different data formats using a single consistent user interface.

9 Abbreviations

3DMF	Quickdraw 3D Metafile Format
API	Application Programming Interface
ASE	Application Service Element
ASCII	American National Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CBS	CollaBoration Services
CCITT	Consultative Comitee on International Telephone and Telegraph
CD	Committee Draft
CGI	Common Gateway Interface
CL	CORBA-Liaison
CMIP	Common Management Information Protocol
CMISE	Common Management Information Service Element
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CS	Communication Services
DIS	Draft International Standard
DLL	Dynamic Loadable Library
DM	Droplet Manager
DP	Draft Proposal
DR	Draft Recommendation
EDCDIC	Extended Binary-Coded Decimal Interchange Code
EM	Event Manager
GOM	Generic Object Model
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IAB	Internet Architecture Board
ID	Internet Draft

IDL	Interface Definition Language
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force (http://www.ietf.org/)
IS	International Standard
ISO	International Standards Organization
ISODE	ISO Development Environment
IT&T	Information Technology and Telecommunication
ITU	International Telecommunications Union
JDBC	Java DataBase Connectivity
JMAPI	Java Management API
LAN	Local Area Network
LRPC	Light Remote Procedure Call
MIB	Management Information Base
MID	Metadata Information Database
MO	Managed Object
NCSA	National Center for Supercomputing Applications
NM	Network Management
NMF	Network Management Forum
NMS	Network Management System
OAS	OpenDoc Automation Services
OCX	OLE Controls eXtension
ODF	OpenDoc Framework
OID	Object Identifier
OLE	Object Linking and Embedding
OMG	Object Management Group
ORB	Object Request Broker
OOP	Object-Oriented Programming
OSA	Open Scripting Architecture
OSI	Open Systems Interconnection
OSIMIS	OSI Management Information Service
PEPY	Presentation Element Parser Yacc-based
PICS	Protocol Implementation Conformance Statement
PDU	Protocol Data Unit
QTCM	QuickTime Component Manager
RAD	Rapid Application Development
RFC	Request For Comment
RMI	Remote Method Invocation
SDK	Software Development Kit
SM	Service Manager

SMI	Structure of the Managed Information
SOM	System Object Model
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
STL	Standard Template Library
TCL	Think Class Library
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	Uniform Resource Locator
VBX	VisualBasic eXtension
VM	Virtual Machine
VRML	Virtual Reality Modeling Language
WD	Working Draft
WIMP	Windows Input Mouse Pointers
XoJIDM	NMF-X/Open Joint Inter-Domain Management Task Force
XOM	X/Open OSI-Abstract-Data-Manipulation API
XMP	X/Open Management Protocols API

10 References

[3DMF] Apple Computers Inc., *QuickDraw 3D Metafile Format (3DMF)*, October 1995.

[Abowd93] G. Abowd, R. Allen and D. Garlan, *Unsing Style to Understand Descriptions of Software Architecture*, Proceedings of ACS SIGSOFT '93, Redmondo Beach, California, December 1993.

[Accetta86] M. Accetta et al., *Mach: A New Kernel Foundation for UNIX Development*, Proceedings of the Summer 1986 USENIX Congerence, Atlanta, 1986.

[ACSE] International Standards Organization, *Information Processings Systems - OSI - Management Information Services - Service Definition for Association Control Service Element (ACSE)*, CCITT Recommendation X.217, ISO/IEC 8649, 1992.

[Adobe96] Adobe Systems Inc., *Adobe Photoshop SDK*, Version 3.05, February 1996.

[Apple89] Apple Computer Inc., *MacApp 2.0: Programmer's Guide*, 1989.

[Apple93] Apple Computer Inc., *Inside Macintosh: More Macintosh Toolbox*, 1993.

[Apple95] Apple Computer Inc., *Components Made Easy*, OpenDoc Technical White Paper, March 1995.

[Apple96] Apple Computer Inc., *Human Interface Toolbox*, WWDC '96 Edition, May 1996.

[ASN1] International Standards Organization, *Specification of Abstract Syntax Notation One (ASN.1)*, CCITT Recommendation X.208, ISO/IEC 8824, 1988.

[Atkinson91] C. Atkinson, *Object-Oriented Reuse: Concurrency and Distribution*, Addison-Wesley/ACM Press, 1991.

[Ban95] B. Ban and L. Deri, *Abstract Factory Revised: a Design Pattern*, IBM Zurich Research Laboratory, RZ 2787, September 1995.

[Ban96] B. Ban, *Towards a Generic Object-Oriented Model for Multi-Domain Management*, Proceedings of ECOOP '96 Workshop on Systems and Network Management, Linz, Austria, July 1996.

[Ban97] B. Ban, *A Generic Management Model for CORBA, CMIP and SNMP,* PhD Thesis, University of Zurich, Institut für Informatik, 1997.

[Barillau97] F. Barillau, L. Deri and M. Feridun, *Network Management Using Internet Technologies*, Proceedings of INM '97, San Diego, April 1997.

[Bernard89] G. Bernard, A. Duda, Y. Haddad, and G. Harrus, *Primitives for Distributed Computing in a Heterogeneous Local Area Network Environment*, IEEE Transactions on Software Engineering, December 1989.

[Biggerstaff89] T. Biggerstaff and A. Perlis, *Software Reusability, Volume I, Concepts and Models*, ACM Press, 1989.

[Birrell84] A. Birrell and B. Nelson, *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, Vol. 2, February 1984.

[Booch91] G. Booch, *Object-Oriented Design with Applications*, ISBN 0-8053-0091-0, Benjamin/Cummings, 1991.

[Booch96] G. Booch and J. Rumbaugh, *Unified Modeling Language for Object-Oriented Programming,* Version 0.9, Rational Software Corporation, 1996.

[Bourne83] S. R. Bourne, *The UNIX System*, Addison-Wesley, Reading, MA, 1983.

[Box95] D. Box, *Building C++ Components Using OLE2*, C++ Report, pp. 29-34, March-April 1995.

[Brockschmidt93] K. Brockschmidt, *Inside OLE : The Fast Track to Building Powerful Object-Oriented Applications*, Microsoft Press, 1993.

[Brown90] P. Brown, *Concepts and Paradigms of Object-oriented Programming,* OOPS Messenges, 1(1), August 1990.

[Budd91] T. Budd, *An Introduction to Object-Oriented Programming,* ISBN 0-201-54709-0, Addison-Wesley 1991.

[Caldiera91] G. Caldiera, V. Basili, *Identifying and Qualifying Reusable Software Components*, IEEE Computer, 24(2), February 1991.

[Cardelli94] L. Cardelli, *Obliq: A Language with Distributed Scope,* Digital Technical Report, 1994.

[CGI] NCSA, *CGI Specification 1.1,* http://hoohoo.ncsa.uiuc.edu/cgi/, National Center for Supercomputing Applications, March 1996.

[Chambers93] C. Chambers, *The Cecil Language: Specification and Rationale,* Technical Report 93-03-05, University of Washington, Seattle, March 1993.

[Champeaux93] D. de Champeaux, D. Lea and P. Faure, *Object-Oriented System Development*, Addison-Wesley, 1993.

[Chen93] D. J. Chen and S. K. Huang, *Interface for Reusable Software Components*, Journal of OO Programming Languages, January 1993.

[Chen94] D. J. Chen and D. T.K. Chen, *An Experimental Study of Using Reusable Software*, Journal of OO Programming Languages, May 1994.

[Ciupke96] O. Ciupke and R. Schmidt, *Components as Context-Independent Units of Software*, ACM Computing Surveys, December 1996.

[Claris93] Claris Corporation, *Claris XTND SDK*, Version 2.0, 1993.

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

[CMIS] International Standards Organization, *Information Technology - OSI, Common Management Information Service Definition (CMIS)*, CCITT Recommendation X.710, ISO/IEC 9595, 1990.

[CMIS++] NMF and X/Open, *CMIS++: CMISE and ACSE++ Application Programming Interface*, Issue 1.0, Draft 8, January 1996.

[Coad91] P. Coad and E. Yourdon, *Object Oriented Analysis*, 2nd edition, Prentice Hall, 1991.

[Collins89] W. Collins, K. Korostoff, *The Reality of OSI Management*, Network World, N. 128, October 1989.

[Cook92] W. Cook, *Interfaces and Specification for the Smalltalk-80 Collection Classes*, Proceedings of Object-Oriented Programming Systems, Languages and Applications, Vancouver, Canada, October 1992.

[Coplien95] O. Coplien and D. Schmidt, *Pattern Languages of Program Design*, Addison-Wesley, 1995.

[Cox91] B. Cox, and A. Novobilski, *Object-Oriented Programming: An Evolutionary Approach*, 2nd Edition, ISBN 0-201-54834-8, Addison-Wesley, 1991.

[Crelier94] R. Crelier, *Separate Compilation and Module Extension*, PhD Thesis no. 10650, Swiss Federal Institute of Technology Zürich, ETH Zürich, 1994.

[Damocles95] Deutsche Telecom Berkom, *Damocles v. 1.0*, 1995.

[Deri92] L. Deri and P. Artico, *System and Network Management*, Proceedings of AICA '92, Genova, Italy, July 1992.

[Deri95a] L. Deri and A. Weder, *Webbin' CMIP*, Poster proceeding of the 4th International WWW Conference, Darmstadt, Germany, April 1995.

[Deri95b] L. Deri and E. Mattei, *An Object-Oriented Approach to the Implementation of OSI Management*, Computer Networks and ISDN Systems, Vol. 27, 1995.

[Deri95c] L. Deri, *Droplets: Breaking Monolithic Applications Apart*, IBM Research Report RZ 2799, September 1995.

[Deri95d] L. Deri, *Mapping Protocol Requests to URLs*, IBM Technical Disclosure SZ8-95-054, December 1995.

[Deri95e] L. Deri, Java Dynamic Class Loader, IBM Research Report SZ8-95-060, December 1995.

[Deri96a] L. Deri, *Surfin′ Network Management Resources Across the Web*, Proceedings of 2nd Int. IEEE Workshop on Systems and Network Management, Toronto, June 1996.

[Deri96b] L. Deri, *Network Management for the 90s*, Proceedings of ECOOP '96 Workshop on Systems and Network Management, Linz, Austria, July 1996.

[Deri96c] L. Deri, *HTTP-based CMIP/SNMP Management*, Internet Draft Draft (draft-deri-http-mgmt-00.txt), November 1996.

[Deri97a] L. Deri and B. Ban, *Static vs. Dynamic CMIP/SNMP Network Management Using CORBA*, Proceedings of IS&N ‘97, Como, Italy, May 1997.

[Deri97b] L. Deri and D. Manikis, *VRML: Adding 3D to Network Management*, Proceedings of IS&N ‘97, Como, Italy, May 1997.

[Deri97c] L. Deri, *Rapid Network Management Application Development*, Proceedings of ECOOP '97 Workshop on Object Oriented Technology for Telecommunications Services Engineering, Jyväskylä, Finland, June 1997.

[Deri97d] L. Deri, *Yasmin: a Component-based Architecture for Software Applications*, IBM Research Report RZ 2899, Proceedings of STEP ‘97, London, July 1997.

[Dittrich96] A. Dittrich and M. Höft, *Integration of a TMN-based Network Management Platform into a CORBA-based Environment*, Proceedings of NOMS '96, 1996.

[DSOM] IBM Corporation, *DSOM Development Toolkit*, October 1994.

[Ellis90] M. Ellis and B. Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley, 1990.

[Gamma91] E. Gamma, *Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools*, PhD Thesis, University of Zurich, Institut für Informatik, 1991.

[Gamma94] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[Garlan93] D. Garlan and M. Shaw, *An Introduction to Software Architecture*, Advances in Software Engineering, Vol. 1, World Scientific Publishing Company, 1993.

[Garlan95] D. Garlan, R. Allen and J. Ockerbloom, *Exploiting Style in Architectural Design Environments*, Proceedings of ACM SIGSOFT '94, ACM Press, December 1994.

[GDMO] International Standards Organization, *Information Technology - OSI - Management Information Services - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects*, CCITT Recommendation X.722, ISO/IEC 10165-4, 1992.

[GDMO_XOM] X/Open Company Ltd., *GDMO to XOM Translation Algorithm*, X/Open Document P319, ISBN 1-85912-023-7, March 1994.

[Geiger94] G. Geiger, W. Allen, A. Majtenyi and P. Reder, *IBM cmipWorks: Technical Paper*, IBM Corporation, March 1994.

[Genilloud96] G. Genilloud, *Towards a Distributed Architecture for Systems Management*, PhD Thesis no. 1588, Ecole Polytechnique Fédérale de Lausanne, December 1996.

[Goldberg83] A. Goldberg and D. Robson, *Smalltalk-80: The language and its Implementation*, Addison-Wesley, 1983.

[Goldszmidt93] G. Goldszmidt and G. Yemini, *Evaluating Management Decisions via Delegation*, IEEE/IFIP Int. Symposium on Network Management, April 1993.

[Grégoire94] J.-Ch. Grégoire, *Models and Support Mechanisms for Distributed Management*, Proceedings of DSOM '94, 1994.

[Halsall90] F. Halsall, N. Modiri, *An Implementation of an OSI NM System*, IEEE Network Magazine, July 1990.

[Hamilton93] G. Hamilton, M. Powell and J. Mitchell, *Subcontract: A Flexible Base for Distributed Programming*, Proceedings of the 14th Symposium on Operating Systems Principles, Asheville NC, December 1993.

[Helm90] R. Helm, I. Holland and D. Gangopadhyay, *Contracts: Specifying Behavioral Compositions in Object-Oriented Systems*, ACM SIGPLAN Notices, 25(10), October 1990.

[Hierro94] J. Hierro, *Architectural Issues For Using Corba Technology in OSI Systems Management*, Append of draft to XoJDM forum, August 1994.

[Hook88] S. Hook, *Objective-C Compiler Version 4: User Reference Manual*, StepStone Corporation, 1998.

[HP_DM] Hewlett-Packard Corporation, *HP OpenView Distributed Management: Platform Developer's Kit*, October 1996.

[HTML] D. Raggett, *HyperText Markup Language Specification Version 3.0 (HTML)*, Internet Draft, April 1995.

[HTTP] T. Berners-Lee, R. Fielding and H. Nielsen, *HyperText Transfer Protocol (HTTP/1.0)*, Internet Draft, October 1995.

[Hudis96] I. Hudis and A. Sinclair, *Introduction to HyperMedia Management*, Internet Draft, December 1996.

[IBM94a] IBM Corporation, *SOM Development Toolkit: An Introductory Guide to the System Object Model and its Accompanying Frameworks*, October 1994.

[IBM94b] IBM Corporation, *The System Object Model (SOM) and the Component Object Model (COM): a comparison of technologies summarized*, July 1994.

[IBM95a] IBM Corporation, *Agent User's Guide for IBM NetView TMN Portable Agent Facility, Release 2.1*, IBM TMN Products, GC31-8209-00, October 1995.

[IBM95b] IBM Corporation, *IBM VisualAge SmallTalk: User's Guide*, 1995.

[IEEE94] IEEE Software, *Software Reuse*, 11(5), 1994.

[IIOP] Object Management Group, *CORBA 2.0/IIOP Specification*, Version 2.0, PTC/96-08-04, June 1995.

[ISO8825] International Standards Organization, Information Processings Systems - OSI - *Specification for Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, CCITT Recommendation X.208, ISO/IEC 8825, 1989.

[ISO9092-1] International Standards Organization, *Text Communication - Open System Interconnection - Remote Operations (ROSE): Model, Notation and Service Definition*, CCITT Recommendation X.219, ISO/IEC 9092-1, 1989.

[ISO9092-2] International Standards Organization, *Text Communication - Open System Interconnection - Remote Operations (ROSE): Protocol Specification*, CCITT Recommendation X.229, ISO/IEC 9092-2, 1989.

[ISO10040] International Standards Organization, *Information Processing System - Open System Interconnection - System Management Overview*, CCITT Recommendation X.701, ISO/IEC 10040, 1992.

[ISO10164-1] International Standards Organization, *Information Processing System - Open System Interconnection - Object Management Function*, CCITT Recommendation X.730, ISO/IEC 10164-1, 1993.

[ISO10164-2] International Standards Organization, *Information Processing System - Open System Interconnection - State Management Function*, CCITT Recommendation X.731, ISO/IEC 10164-2, 1993.

[ISO10164-3] International Standards Organization, *Information Processing System - Open System Interconnection - Objects and Attributes for Representing Relationships*, CCITT Recommendation X.732, ISO/IEC 10164-3, 1993.

[ISO10164-4] International Standards Organization, *Information Processing System - Open System Interconnection - Alarm Reporting Function*, CCITT Recommendation X.733, ISO/IEC 10164-4, 1992.

[ISO10164-5] International Standards Organization, *Information Processing System - Open System Interconnection - Event Report Management Function*, CCITT Recommendation X.734, ISO/IEC 10164-5, 1993.

[ISO10164-6] International Standards Organization, *Information Processing System - Open System Interconnection - Log Control Function*, CCITT Recommendation X.735, ISO/IEC 10164-6, 1993.

[ISO10164-9] International Standards Organization, *Information Processing System - Open System Interconnection - Objects and Attributes for Access Control*, CCITT Recommendation X.741, ISO/IEC 10164-9, 1995.

[ISO10164-10] International Standards Organization, *Information Processing System - Open System Interconnection - Accounting Meter Function*, CCITT Recommendation X.742, ISO/IEC 10164-10, 1995.

[ISO10164-11] International Standards Organization, *Information Processing System - Open System Interconnection - Workload Monitoring Function*, CCITT Recommendation X.739, ISO/IEC 10164-11, 1995.

[ISO10165-1] International Standards Organization, *Information Technology - OSI - Management Information Services - Structure of Management Information - Part 1: Management Information Model*, CCITT Recommendation X.720, ISO/IEC 10165-1, 1992.

[ISO10165-2] International Standards Organization, *Information Technology - OSI - Management Information Services - Structure of Management Information - Part 2: Definition of Management Information*, CCITT Recommendation X.721, ISO/IEC 10165-2, 1992.

[ISO10165-5] International Standards Organization, *Information Technology - OSI - Management Information Services - Generic Management Information*, CCITT Recommendation X.721, ISO/IEC 10165-5, 1992.

[ISO7498-4] International Standards Organization, *Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework*, ISO/IEC 7498-4, 1989.

[ISODE] M. Rose, J. Onions, C. Robbins, *The ISO Development Environment: User's Manual*, Version 8.0, June 1992.

[Jander96] M. Jander, *Web-based Management: Welcome to the Revolution*, Data Communications, pp. 39-53, November 21, 1996.

[JavaBeans] Sun Microsystems, *JavaBeans: API Specification*, Version 1.00A, JavaSoft, December 1996.

[Jazayeri95] M. Jazayeri, *Component Programming: a Fresh Look at Software Components*, Fifth European Software Engineering Conference, Barcelona, Spain, September 1995.

[Jeffree92] Jeffree et al., *Technical Guide for OSI Management*, Version 1.2, British Crown, January 1992.

[JMAPI]	Sun Microsystems, *Java Management API: Programmers's Guide*, Part No. 802-6616-01, Revision A, November 1996.
[Joch96]	A. Joch, *Killer Components*, Byte Magazine, January 1996.
[Johnson88]	R. Johnson and B. Foote, *Designing Reusable Classes*, Journal of Object-Oriented Programming, June/July 1988.
[Johnson91a]	R. Johnson and V. Russo, *Reusing Object-Oriented Designs*, Univ. of Illinois, TR UIUCDCS 91-1696, 1991.
[Johnson91b]	R. Johnson and J. Zweig, *Delegation in C++* , JOOP, 4(11), November 1991.
[Johnson93]	R. Johnson, *How to Design Frameworks*, Notes from OOPSLA, 1993.
[Joseph90]	C. Joseph, K. Muralidhar, *Integrated Network Management in an enterprise environment*, IEEE Network Magazine, July 1990.
[Joymer92]	I. Joymer, *A C++ Critique*, April 1992.
[Karlsoon95]	E. Karlsson, *Software Reuse: A Holistic Approach*, ISBN 0-471-95819-0, John Wiley & Sons, 1995.
[Kasteleijn97]	W. Kasteleijn, *Web based Management*, M.Sc. Thesis, University of Twente, Department of Computer Science, April 1997.
[Kernighan88]	B. Kernighan and D. Ritchie, *The C Programming Language*, Prentice Hall, 1988.
[Klerer88]	S. Klerer, *The OSI Management Architecture: an Overview*, IEEE Network, 2(2), March 1988.
[Knight94]	G. Knight, S. Bhatti and L. Deri, *Secure Remote Management in the Esprit MIDAS Project*, Proceedings of IFIP '94, Barcelona, Spain, June 1994.
[Knight95]	G. Knight, *Wombats*, University College of London, 1995.
[Knut73]	D. Knut, *The Art of Computer Programming*, Vol. 1, 2 and 3, Addison-Wesley, 1973.
[Kong96]	Q. Kong and G. Chen, *Integrating CORBA and TMN Environments*, Proceedings of NOMS '96, 1996.
[Krüger92]	C. Krüger, *Software Reuse*, ACM Computing Survey 24(2), June 1992.
[Leeb96]	A. Leeb, *A Flexible Object Architecture for Component Software*, Master's Thesis, MIT, June 1996.
[Lewis95]	T. Lewis, et al., *Object-Oriented Application Frameworks*, ISBN 1-884777-06-6, Manning Publications, 1995.
[Lindenberg90]	J. Lindenberg, *NetWork Communications*, Universität Karlsruhe, Institut für Betriebs und Dialogsysteme, 1990.
[Linton87]	M. Linton and P. Calder, *The Design and Implementation of Interviews*, Proceedings of USENIX C++ Workshop, Santa Fe, NM, November 1987.

[Löhr93] P. Löhr, *Concurrency Annotations for Reusable Software*, Communications of the ACM, 36(9), September 1993.

[Maes87] P. Maes, *Concepts and Experiments in Computational Reflection*, Proceedings of the 2nd OOPSLA Conference, pp. 147-155, 1987.

[Magee92] J. Magee, N. Dulay and J. Kramer, *Structuring Parallel and Distributed Programs*, Proceedings of Int. Workshop on COnfigurable Distributed Systems, London, March 1992.

[Magee95] J. Magee and J. Kramer, *Modelling Distributed Software Architectures*, Proceedings of the 1st Int. Workshop on Architectures for Software Systems, April 1995.

[Marais96] J. Marais, *Design and Implementation of a Component Architecture for Oberon*, PhD Thesis no. 11697, Swiss Federal Institute of Technology Zürich, ETH Zürich, 1996.

[Marben95] Marben Products, Inc., *CMIP Agent Toolkit*, Draft 1.2, November 1995.

[Mathews90] D. C. Mathews, *Static and Dynamic Type Checking*, Bancilhon & Buneman, 1990.

[Mätzel96] K. Mätzel and W. Bischofberger, *The Any Framework: A Progmatic Approach to Flexibility*, Proceedings of 2nd USENIX Conference of Object-Oriented Technologies and Systems, Toronto, June 1996.

[McIlroy69] D. McIlroy, *Mass-produced Software Components*, NATO Conference, 1969.

[Metrowerks96] Metrowerks Corporation, *Codewarrior IDE Plugin API Specification*, Version 1.5, April 1996.

[Meyer87] B. Meyer, *Reusability: The Case for Object-Oriented Design*, IEEE Software, 4(2), March 1987.

[Meyer88] B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, 1988.

[Meyer95] K. Meyer, et al., *Decentralizing Control and Intelligence in Network Management*, Proceedings Int. Symposium on Integrated Network Management, May 1995.

[Micallef88] J. Micallef, *Encapsulation: Reusability and Extensibility in Object Oriented Programming Languages*, Journal of OO Programming Lang., April/May 1988.

[Microsoft92] Microsoft Corporation, *Visual Basic*, Microsoft Press, 1992.

[Microsoft93a] Microsoft Corporation, *Object Linking and Embedding v.2 (OLE2): Programmer's Ref*erence, Vols. 1 and 2, Microsoft Press, 1993.

[Microsoft93b] Microsoft Corporation, *Component Objects: Technology Overview*, September 1993.

[Microsoft95] Microsoft and Digital Corporation, *The Component Object Model Specification*, Draft Version 0.9, October 1995.

[Nackman94] L. Nackman and J. Barton, *Base-Class Composition with Multiple Derivation and Virtual Bases*, IBM T.J. Watson Research Center, 1994.

[Nierstrasz90] O. Nierstrasz and M. Papathomas, *Viewing Objects as Patterns of Communicating Agents*, Proceedings OOPSLA/ECOOP '90, ACM SIGPLAN Notices, 25(10), October 1990.

[Nierstrasz92] O. Nierstrasz, S. Gibbs and D. Tsichritzis, *Component-Oriented Software Development*, Communications of the ACM, 35(9), September 1992.

[Nierstrasz95] O. Nierstrasz and D. Tsichritzis, *Object-Oriented Software Composition*, ISBN 0-13-220674-9, Prentice Hall, 1995.

[OMG91] Object Management Group, *The OMG Object Model v.0.9*, Object Management Group/Object Model Task Force, Boulder, CO, 1991.

[OMG92] Object Management Group, *Object Management Architecture Guide*, OMG TC Document 92.11.1, Revision 2.0, September 1992.

[OMG95] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, July 1995.

[OMNIPoint93] Network Management Forum, *Discovering OMNIPoint*, PTR Prentice Hall, New Jersey, 1993.

[Ousterhout94] J. Ousterhout, *The Tcl Language and the Tk Toolkit*, Addison-Wesley, 1994.

[OSIMIS] UCL Network and System Management, *The OSI Management Information Service: User's Manual*, Version 4, August 1996.

[Pacheco96] X. Pacheco and S. Teixeira, *Delphi 2 Developer's Guide*, 2nd edition, ISBN 0-672-30914-9, Borland Press, 1996.

[Pavlou91] G. Pavlou, G. Knight and S. Walton, *Experience of Implementing OSI Management Facilities*, Proceedings of 2nd Int. Symposium on Integrated Network Management, Washington, April 1991.

[Pavlou93] G. Pavlou, S. Bhatti and G. Knight, *The OSI Management Information Service: User's Manual*, Version 1.0, University College of London, February 1993.

[Pavlou96] G. Pavlou and T. Tin, *A CMIS-capable Scripting Language and Associated Lightweight Protocol for TMN Applications*, IEEE Communications, 34(9), September 1996.

[Petzold95] C. Petzold, *Programming Windows 95*, Microsoft Programming Series, Microsoft Press, 1995.

[Pfister96] C. Pfister and C. Szypersky, *Why Objects Are Not Enough*, Proceedings of 1st Int. Component Users Conference (CUC ‘96), Munich, Germany, July 1996.

[Quinn93] P. Quinn and G. Preoreasa, *Reconciling Object Models for Systems and Network Management*, UNIX System Laboratories Inc., 1993.

[Qwerin91] N. Qwerin, *Open Systems: a Handbook*, November 1991.

[Reilly97] J. Reilly, P. Niska, L. Deri and D. Gantenbein, *Enabling Mobile Network Managers*, Proceedings of the 6th Int. WWW Conference, Santa Clara, CA, April 1997.

[RFC1155] M. Rose and K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-based internets*, RFC 1155, May 1990.

[RFC1156] K. McCloghrie and M. Rose, *Management Information Base for Network Management of TCP/IP-based Internets*, RFC 1156, May 1990.

[RFC1158] M. Rose, *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*, RFC 1158, May 1990.

[RFC1351] J. Davin, J. Galvin and K. McCloghrie, *SNMP Administrative Model*, RFC 1351, July 1992.

[RFC1352] J. Galvin, K. McCloghrie and J. Davin, *SNMP Security Protocols*, RFC 1352, July 1992.

[RFC1902] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1902, January 1996.

[RFC1905] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1905, January 1996.

[RFC1907] J. Case, K. McCloghrie, M. Rose and S. Waldbusser, *Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1907, January 1996.

[Rice95] J. Rice, A. Farquhar, P. Piernot and T. Gruber, *Lessons Learned Using the Web as as Application Interface*, Knowledge Sharing Technology Project, Stanford University, 1995.

[Rochkind85] M. Rochkind, *Advanced Unix Programming*, Prentice Hall, 1985.

[Rose89] M. Rose, *The ISO Development Environment: User's Manual*, The Wollongong Group, 5.0 edition, March 1989.

[Rose90] M. Rose, *The Open Book: a Practical Perspective on OSI*, ISBN 0-13-643016-3, Prentice Hall, 1990.

[Rumbaugh91] J. Rumbaugh and others, *Object Oriented Modeling and Design*, Prentice Hall, 1991.

[Rumbaugh94] J. Rumbaugh, *The Life of an Object Model: how the Object Model Changes During Development*, Journal of Object-Oriented Programming, March/April 1994.

[Rutt94] T. Rutt, *Comparison of the OSI management, OMG and Internet management Object Models*, report of Joint XOpen/NM Forum Inter-Domain Management Task Force, March 1994.

[Sawitzki92] G. Sawitzki, *The Network Project: Distributed Computing on the Macintosh*, develop, Issue 11, August 1992.

[Schmidt95] C. Schmidt and M. Sevcik, *Do-It-Yourself TMN Applications by Visual Programming Methods*, IEEE Communications Magazine, November 1995.

[Schönwälder95] J. Schönwälder and H. Langendörfer, *Tcl Extensions for Network Management Applications*, Comp. Science Dept., Univ. Of Braunschweig, May 1995.

[Schürfeld94] U. Schürfeld, D. Gantenbein, *Bilingual Agent: DSOM Access to X.700 Agent*, IBM Zurich Research Laboratory, March 1994.

[SGI94] Silicon Graphics Inc., *OpenInventor C++ Reference Manual*, Addison-Wesley, 1994.

[Shapiro86] M. Shapiro, *Structure and Encapsulation in Distributed Systems: the Proxy Principle*, 6th Int. Conference on Distributed Computing Systems, Boston, Mass., May 1986.

[Shaw94] M. Shaw and D. Garlan, *Characteristics of Higher-level Languages for Software Architecture*, CMU Technical Report CMU-CS-94-210, Carnagie Mellon University, December 1994.

[Shaw95a] M. Shaw and D. Garlan, *Formulations and Formalisms in Software Architecture*, Vol. 1000, Springer-Verlag Lecure Notes in Computer Science, 1995.

[Shaw95b] M. Shaw, *Architectural Issues in Software Reuse: it's not just the functionality, it's the packaging*, Proceedings on the Symphosium on Software Reuse, April 1995.

[Shaw96] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[SimpleTimes] The Simple Times Newsletter,*Emerging Management Technologies*, http://www.simple-times.org/, 4(3), July 1996.

[Sloman94] M. Sloman, *Network and Distributed Systems Management*, Addison-Wesley, 1994.

[Smith92] D. Smith and J. Susser, *A Component Architecture for Personal Computer Software,* in Languages for Developing User Interfaces, ed. B. Myers, Jones & Bartlett, 1992.

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

[Solstice] Sun Microsystems, *Solstice TMN Agent Toolkit*, 1996.

[Soukouti95] N. Soukouti, *Managing OSI Objects Using a CORBA Manager,* Append of draft to XoJIDM Forum, 1995.

[Staringer94] W. Staringer, *Constructing Applications from Reusable Components*, IEEE Software, 11(5), September 1994.

[Stevens90] R. Stevens, *Unix Network Programming,* ISBN 0-13-949876-1, Prentice Hall, 1990.

[Stroustrup91] B. Stroustrup, *The C++ Programming Language*, Second Edition, Addison-Wesley, 1991.

[Sun96a] Sun Microsystems, *The Java Programming Language*, ISBN-0-201-63455-4, Addison-Wesley, 1996.

[Sun96b] Sun Microsystems, *Java Management API (JMAPI): Programmer's Guide*, Part No. 802-6616-01, November 1996.

[Symantec93a] Symantec Corporation, *Bedrock Architecture*, 1993.

[Symantec93b] Symantec Corporation, *THINK Class Library Guide*, 1993.

[Szypersky95] C. Szypersky, *Component-Oriented Programming: A Refined Variation on Object-Oriented Programming*, The Oberon Tribune, 1(2), December 1995.

[Szypersky96] C. Szypersky, *Independently Extensible Systems: Software Engineering Potential and Challenges*, Proceedings of 19th Australasian Computer Science Conference, Melbourne, Australia, February 1996.

[Taligent93] Taligent Inc., *Leveraging Object-Oriented Frameworks*, 1993.

[Taligent94] Taligent Inc., *Taligent's Guide to Designing Programs: Well-Mannered Object-Design in C++*, Addison-Wesley, 1994.

[Taligent95a] S. Cotter and M. Potel, *Inside Taligent Technology,* ISBN 0-201-40970-4, Addison-Wesley, October 1995.

[Taligent95b] Taligent Inc., *The Power of Frameworks*, Addison-Wesley, 1995.

[Tanenbaum96] A. Tanenbaum, *Computer Networks*, 3rd Edition, ISBN 0-13-349945-6, Prentice Hall, 1996.

[Theimer89] M. Theimer and K. Lantz, *Finding Idle Machines in a Workstation-Based Distributed System*, IEEE Transactions on Software Engineering, November 1989.

[Tin95] T. Tin, G. Pavlou and R. Shi, T*cl-MCMIS: Interpreted Management Access Facilities,* Proceedings of DSOM '95, October 1995.

[TMN++96] X/Open and Network Management Forum, *TMN++: CMIP APIs*, Preliminary Specification, 1996.

[Tsichritzis89] D. Tsichritzis, *Object-Oriented Development for OpenSystems*, Proceedings of IFIP '89, North-Holland, San Francisco, August 1989.

[Udell94] J. Udell, *Componentware*, Byte, May 1994.

[Ungar87] D. Ungar and R. Smith, *Self: The Power of Simplicity,* OOPSLA '87 Conference Proceedings, October 1987.

[URL] T. Berners-Lee, *Uniform Resource Locators (URL)*, Internet Draft, 03/21/1994.

[Vertel97] Vertel Corporation and Microsoft Corporation, *Accessing TMN Through Web-Based Enterprise Management*, White Paper, February 1997.

[VRML] G. Bell, A. Parisi and M. Pesce, *The Virtual Reality Modeling Language (VRML)*, Version 1.0, May 1995.

[Wall96] L. Wall, T. Christiansen and R. Schwartz, *Programming Perl*, Second Edition, ISBN 1-56592-149-6, O'Reilly & Associates, 1996.

[Waskiewicz95] F. Waskiewicz, *An Object-Oriented Framework for Manufacturing Applications*, OMG BOMSIG, Ottawa, Canada, 1995.

[Wayner96] P. Wayner, *Net Programming for the Masses*, Byte, February 1996.

[Wayt94] G. Wayt, *Software's Chronic Crisis*, Scientific American, September 1994.

[Weck96] W. Weck, *Independently Extensible Component Frameworks*, Institute of Scientific Computing, ETH Zürich, 1996.

[Wirfs-Brock90] R. Wirfs-Brock, B. Wilkerson and L. Wiener, *Designing Object-Oriented Software*, Prentice Hall, 1990.

[Wegner87] P. Wegner, *Dimensions of Object-Based Language Design*, Proceedings of OOPSLA '87, ACM SIGPLAN Notices, Vol. 22, Orlando, Florida, December 1987.

[Weinand88] A. Weinand, E. Gamma and R. Marty, *ET++: An Object-oriented Application Framework in C++*, ACM OOPSLA '88 Conference Proceedings, San Diego, CA, 46-57, September 1988.

[XMP] X/Open Company Ltd., *Systems Management: Management Protocol API (XMP)*, X/Open Document C306, ISBN 1-85912-027-X, March 1994.

[XOBJ] Finsiel S.p.A., *X/OBJ Agent Platform: User Documentation*, 1994.

[XoJIDM95a] Joint Inter-Domain Working Group, X/Open and Network Management Forum, *Inter-Domain Management Specifications: Specification Translation*, April 1995.

[XoJIDM95b] Joint Inter-Domain Working Group, X/Open and Network Management Forum, *Inter-Domain Management Specifications: Preliminary CORBA/CMISE Interaction Translation Architecture*, April 1995.

[XOM] X/Open Company Ltd., *OSI-Abstract-Data Manipulation API (XOM)*, X/Open Document C315, ISBN 1-85912-008-3, February 1994.

[Yang96] Z. Yang and K. Duddy, *CORBA: a Platform for Distributed Object Computing*, Operating Systems Review, 30(2), April 1996.

[Yemini91] Y. Yemini, G. Goldszmidt and S. Yemini, *Network Management by Delegation*, Proceedings of 2nd Int. Symphosium on Integrated Network Management, April 1991.

[Young89] M. Young, *Software tools for OS/2: Creating Dynamic Link Libraries*, ISBN 0-201-51787-6, July 1989.

[Zwemmer96] M. Zwemmer, F. van Hengstum and M. Sinderen, *Developing a SNMP Protocol Entity in Object-Oriented Perl*, Proceedings of ECOOP '96 Workshop on Systems and Network Management, Linz, Austria, July 1996.

A Network Management

1.1. OSI Network Management

1.1.1. The OSI Management Standards

ISO (International Standards Organization), the international organisation for standardisation, and IEC (International Electrotechnical Commission), the international electrotechnical committee, collaborate in a programme of international standardisation for information technology, of which OSI (Open Systems Interconnection) is part. CCITT (Consultative Committee on International Telephone and Telegraph), the equivalent of OSI in the telecommunications world, is responsible for international recommendations governing interworking between public communication networks. Years ago, CCITT joined with ISO/IEC in an effort to develop standards for OSI. In the area of OSI network management, ISO/IEC and CCITT have jointly published a set of core standards. In addition, the two organisations will publish further standards and recommendations focusing on their respective interests, which do not overlap. For example, CCITT has released some recommendations concerning the management of the internals of public data networks and public telephone exchanges topics, which are not relevant to ISO/IEC.

The development of new standards and recommendations follows a well specified procedure. Once a topic of interest has been identified, a working group is formed. This group then produces drafts, which are circulated and reviewed by member organisations. Within ISO/IEC, when a draft document can potentially become a standard, it is turned into a *Working Draft* (WD). It then progresses to a *Committee Draft* (CD) also known as *Draft Proposal* (DP), and then *Draft International Standard* (DIS) before becoming ratified as a full *International Standard* (IS). Owing to this long process and to the set of formal ballots necessary, a CD is turned into an IS in not less than two years from submission of the CD. CCITT has *Draft Recommendations* (DR), which are turned into *Recommendations* on a four-year cycle although expedited procedures can be used when necessary. The relative slowness of the standardisation process is one reason that pushed many organisations to pay more attention to Internet standards, which are approved in a shorter amount of time, making them more suitable for rapidly changing technology.

Many management standards are currently under development, especially in the area of telecommunications networks, where new digital technologies are replacing analog ones. The following table contains a summary of the core OSI management standards.

ISO/IEC	CCITT	Standard Name
7498-4	X.700	OSI Management Framework
8824	X.208	Specification of Abstract Syntax Notation One (ASN.1)
8825	X.209	Basic Encoding Rules for Abstract Syntax Notation One (BER)
9072-1	X.219	Remote Operations: Model, Notation and Service Definition (ROSE)
9072-2	X.229	Remote Operations: Protocol Specification
9595	X.710	Common Management Information Service (CMIS)
9596-1	X.711	Common Management Information Protocol (CMIP)
10040	X.701	Systems Management Overview
10164-1	X.730	Object Management Function
10164-2	X.731	State Management Function
10164-3	X.732	Objects and Attributes for Representing Relationships
10164-4	X.733	Alarm Reporting Function
10164-5	X.734	Event Report Management Function
10164-6	X.735	Log Control Function
10164-9	X.741	Objects and Attributes for Access Control
10164-10	X.742	Accounting Meter Function
10164-11	X.739	Workload Monitoring Function
10165-1	X.720	Management Information Model (MIM)
10165-2	X.721	Definition of Management Information (DMI)
10165-4	X.722	Guidelines for the Definition of Managed Objects (GDMO)
10165-5	X.724	Generic Management Information (GMI)

Table 1. Core OSI Standards

1.1.2. OSI Reference Model

In order to reduce their complexity, computer networks have been divided into layers, where layer N of host A talks with layer N of host B. The rules and the conventions used during the communication are called *N-layer protocols*. Entities that are part of the same layer but on different hosts are called *peer processes*. Data are not transmitted directly from layer N of host A to layer N of host B but they cross the layers N..1 of each host according to the following picture.

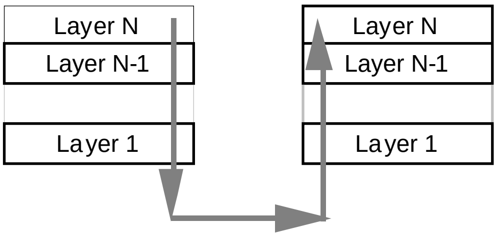


Table 2. Layered Network Communications

Between two adjacent services there is an *interface* which defines the primitives and the services offered by a level to the one immediately below it. The

set of these levels, usually called a *stack*, is the *network architecture*. The architecture specification contains the information necessary to enable people to write software applications or build network devices which can operate with the specified protocols.
OSI has defined a seven-layer network architecture, usually called the *OSI Stack*, and denoted the *OSI Reference Model*.

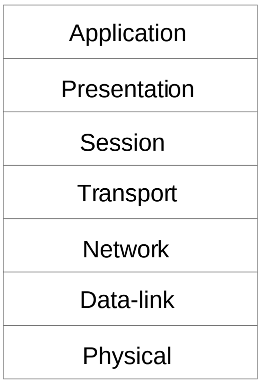


Figure 1. OSI Reference Model (OSI Stack)

Note that the OSI Reference Model is not a network architecture because it does not exactly specify the services and the protocols used by each layer but simply defines how each layer is supposed to behave.

The *physical layer* is responsible for the transmission of bits through a physical communication channel to a directly connected peer. In addition, it has to guarantee that every bit sent has been received by the peer. At this layer the mechanical and electrical specifications, the interface types, and the physical medium are defined.
The *data-link layer* is responsible for reliably transmitting data to a directly connected peer. This is done by splitting data into data frames (typically of 100 bytes), sending them sequentially, and handling the acknowledge frames returned by the peer. Because the physical layer transmit bits without caring about their structure, the data-link layer is responsible for splitting data into frames and packing them into the original structure.
The *network layer* is responsible for the operations concerning the various subnetworks. It has to be able to find a path through which data can arrive to the final destination (routing), to control traffic congestion (for instance through a push-back mechanism), to convert the information (for instance to encode the receiver address), and to split data into smaller units.
The *transport layer* receives data from the session layer, and splits them into smaller units, which are sent to the network layer. This entire process is performed in an efficient way in order to shield the session from the underlying network technology used. In addition, the transport has to guarantee that data sent/received are in the same order and that different data flows are multiplexed on the same channel.
The *session layer* allows hosts to establish communication sessions (either simplex or duplex) over which data are exchanged. This layer also allows data to be transferred in different sessions without having to do it at one time. In this way it is possible to transfer data over unreliable connections by exchanging data at different times when the connection is up.

The *presentation layer* deals with the syntax and the semantics of the exchanged information by converting data from different representations (for instance ASCII and EDCDIC), by compressing and encrypting data in order to save transmission time, and by guaranteeing privacy and authentication. *Application layer* protocols, mostly used by end users, include network virtual terminal, file transfer, electronic mail and other general purpose services.

1.1.3. ASN.1

Up to the presentation level of the OSI stack, data are exchanged in binary format. At the application level, data are much more complex and have to be represented using a notation independent of the format of the data being exchanged. The OSI language for abstract syntax representation is ASN.1. Its main characteristic is the separation of the data structure from its binary representation.

The ASN.1 *datatype* is defined as the set of data with a certain name that can assume a limited range of values. Each variable is a member of a certain datatype which limits the set of values that the variable can have. The concept of datatype is equivalent to the one present on strong typed programming languages like Pascal and Modula-2. The ASN.1 datatype are:

```
DataType    ::= BuiltinType | DefinedType
BuiltinType ::= BooleanType | IntegerType | OctetStringType | NullType |
                SequenceType| SequenceOfType | SetType | SetOfType |
                ChoiceType | SelectionType | TaggedType | AnyType |
                ObjectIdentifierType | CharacterStringType | UsefulType
DefinedType ::= ExternalTypereference | typereference
```

where a `typereference` is an arbitrary number of letters, numbers and underscores (_). Among the built-in types, the `AnyType` is the most important one. It is used to represent the type of a component whose type is known only at runtime (it is similar to the `void*` of the C language). Each datatype is associated with a *tag* to distinguish it among different types. Tags can be used in different contexts, hence they have been divided into four classes:

- universal, used for the types defined in the ASN.1 standard [ASN1];
- application, used by OSI application layer standards;
- private, used by companies and institutions for applications they develop which need not be standardised;
- context-specific, class used when tag value depends on the context in which the tag is used.

The `TaggedType` notation is (words all caps are ASN.1 reserved words):

```
TaggedType ::= Tag Type | Tag IMPLICIT Type
Tag ::= {Class ClassNumber}
ClassNumber ::= number | DefinedValue
Class ::= UNIVERSAL | APPLICATION | PRIVATE | empty
```

Users can employ ASN.1 as a programming language by defining macros which are a notation defined by the user to combine language structures. A *macro definition* specifies the syntax for the type definition and its value. `Mac-`

`roDefinition` syntax is:

```
MacroDefinition       ::= macroreference
                          MACRO
                          “::=“
                          BEGIN
                          MacroBody
                          END
MacroBody             ::= TypeProduction
                          ValueProduction
                          SupportingProductions
TypeProduction        ::= TYPE NOTATION
                          “::=“
                          MacroAlternativeList
ValueProduction       ::= VALUE NOTATION
                          “::=“
                          MacroAlternativeList
SupportingProductions ::= ProductionList| empty
ProductionList        ::= Production| ProductionList Production
Production            ::= productionreference
                          “::=“
                          MacroAlternativeList
```

ASN.1 macros are extensively used by SNMP to specify MIB definitions (see “The SNMP Protocol” on page 190).

1.1.4. CMISE

The Common Management Information Service Element (CMISE) is a member of the OSI *Application Service Element* (ASE) which is contained in layer seven of the OSI stack. CMISE defines the services and transfer procedures for CMIP Data *Units* (CMIP-PDUs). CMISE implementations are used by System Management Applications (SMAs). Such SMAs offer system management services by providing mechanisms for monitoring and controlling resources of a system management environment made up of managing and managed systems. Each of these systems need some SMA for their management process. Inside the SMAs, CMISE provides a way to exchange information and commands.
CMISE sits on top of ROSE and ACSE. ROSE services are invoked directly by CMISE, which uses them to exchange management information. CMIP-PDUs travel over an association (CMIP is a connection-oriented service) established using ACSE between two management entities. There is no direct communication between CMIP and ACSE primitives. Instead a CMISE user establishes an association using ACSE and reports the association establishment to CMISE. CMISE service primitives are divided into two groups: primitives for notification exchange and for operation exchange.

CMISE Service	Mode	Description
m-event-report	C/U[a]	Returns a notification concerning a managed object
m-get	C	Requests information about a managed object (attributes)
m-cancel-get	C	Cancels a previous m-get request still in progress
m-set	C/U	Sets one or more attributes of a managed object
m-action	C/U	Requests a CMISE user to perform an action on a managed object

CMISE Service	Mode	Description
m-create	C	Requests the creation of a new managed object instance
m-delete	C	Requests the deletion of a new managed object instance

a. C stands for confirmed service, U stands for unconfirmed service.

Table 3. CMISE Services

CMISE allows MOs, on which the operation has to be performed, to be selected by means of scoping and filtering mechanisms. Scoping identifies in the containment tree the objects to which the filtering can be applied. The filtering specifies the set of conditions a MO must satisfy in order to be selected for the requested management operation. For some service primitives, CMIS offers a synchronisation parameter, which specifies how the operation has to be performed, namely atomically or best-effort.

CMIS services are divided into five functional units. Namely:

1. Kernel

   All the seven services are part of a kernel. Linked responses (multiple responses for a single request), scoping, filtering, and synchronisation are not supported.

2. Multiple Object Selection

   It provides scoping and filtering support to the kernel services.

3. Filter

   It provides filtering support to the kernel services.

4. Multiple Reply

   It provides linked responses support to the kernel services.

5. Extended Service

   It provides additional services support to the ones described so far (extended service is not usually implemented).

The CMIS functional units are negotiated at the time of association establishment. In other words when an ACSE association is established, the peers agree on the set of the functional units that will be supported by the association. If no agreement is reached, the association cannot be established. The negotiation of functional units is necessary because the implementation of the CMISE is quite complex and mechanisms such as synchronisation are usually not supported by CMISE implementations because they are difficult to implement. Therefore, the negotiation of functional units allows the implementation to be simplified by not supporting functional units other than the kernel (mandatory).

1.2. Internet Network Management

1.2.1. Internet Standards

The *Internet Engineering Task Force* (IETF) is the body which takes care of the standardisation process within the Internet community. IETF is divided into *working groups* with a narrow focus on a certain topic, coordinated by a chair person who oversees precise goals and milestones. Working groups meet three times a year and group members subscribe to specific mailing lists where the issues are discussed. There is no formal voting, disputes are resolved by discussion and demonstration on the mailing lists, and final decisions are made via e-mail, never at meetings. Each working group produces Internet documents which can be either *Internet Drafts* (ID) or *RFCs* (Request For Comments). During the development of a specification, draft versions of the document are made available for informal review and comment in the form of an ID. An ID is a working document without a status of any kind, which is freely distributed and mirrored world wide on certain Internet hosts (the European host is nic.nordu.net) by the IETF Secretariat. When an ID remains unchanged for more than six months it is deleted. At any time, an ID may be replaced by a more recent version of the same specification, restarting the six-month timeout period. An ID can be turned into an RFC, which is a standard-related specification released under the general direction of the IAM (Internet Architecture Board) under the direct responsibility of the RFC editor.

Within the Internet standards process, RFCs are divided into three categories according to their maturity level: proposed standard, draft standard, and standard. Proposed standards are complete, credible specifications which have been demonstrated to be useful and which are pending for between six months and two years, at which point they are either elevated, depreciated, or recycled. Draft standards require multiple (at least two), independent and interoperable implementations of the standard being proposed and take from four months to two years before being either elevated, depreciated, recycled, or turned back into a proposed standard. Internet standards must demonstrate operation stability and can pend forever until they are depreciated to historic (see below).

Not every specification is on the standard track, because it may be intended for eventual standardisation but it is not yet mature. Specifications which are not on the standards track are labelled according to one of the three off-track maturity levels: experimental, informational, or historic.

The table below contains a summary of the core Internet standards for network management.

RFC	RFC Name
1155	Structure and Identification of Management Information
1156	Management Information Base for Network Management
1157	A Simple Network Management Protocol (SNMP)
1158	Management Information Base for Network Management: MIB-II

RFC	RFC Name
1351	SNMP Administrative Model
1352	SNMP Security Protocols
1902	Structure of Management Information for SNMPv2
1905	Protocol Operations for SNMPv2
1907	Management Information Base for SNMPv2

1.2.2. The SNMP Protocol

SNMP is a simple polling protocol used to manage Internet networks. It allows a centralised NMS to query SNMP agents, and to retrieve and modify the information contained on the MIB of such agents. SNMP is much simpler than CMIP and consists only of three types of operations:

- Get retrieves information from the MIB;
- Set alters the MIB information;
- Trap reports on an event.

In the cases of Get and Set, NMS is the initiator of the operation, whereas in the case of Trap the SNMP agent sends the Trap to the NMS when a specific situation occurs. Other operations are emulated by these simple operations. The Create operation is emulated using a Set on a nonexistent MIB variable, which is then created and added to the MIB. Deletion of MIB variables is emulated using a Set of a MIB variable to a non valid value. Creation and deletion of MIB instances can be performed only if the MIB supports these operations. In fact most of the MIB variables represent real resources such as Ethernet cards or bridges, which can be created/deleted only if the resource is physically added/removed to the system. The three types of operations are performed with a set of only five SNMP primitives:

SNMP Primitive	Description
Get-request	Queries the SNMP agent to fetch one or more MIB variables
Get-next-request	Queries the SNMP agent to fetch the next MIB variable after the specified one
Set-request	Requests the SNMP agent to set the specified MIB variable
Get-response	Response containing the return value of a XXX-request
Trap	Report about an event returned by the SNMP agent

Table 4. SNMP Primitives

As MIB variables are ordered according to their OID (Object Identifier) value, it is possible to walk the MIB using the Get-next-request primitive. In the SNMP version 2, which is currently being standardised, new primitives have been added to SNMP, mainly to improve the efficiency of the protocol and to reduce the number of SNMP requests necessary to perform an operation (for instance the retrieval of all the network devices connected to a machine).

Additional SNMPv2 Primitives are:

SNMPv2 Primitive	Description
Get-bulk-request	Allows the NMS to retrieve large blocks of data efficiently
Inform-request	Allows NMS-to-NMS communications

Table 5. Additional SNMPv2 Primitives

As SNMP uses only basic syntax types such as strings or numbers, aggregate types are emulated through the use of tables. Each table row contains an element, whereas each column represents an element of the aggregate type. For instance the table below shows how a C struct can be mapped to a SNMP table.

C Struct	SNMP Table

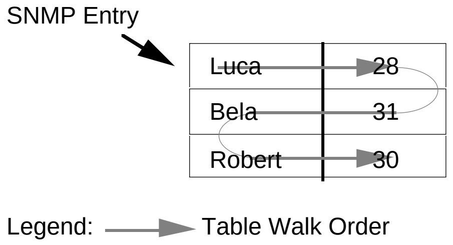
`struct SimpleStruct {`
`   char* name;`
`   int   age;`
`};`
	 SNMP Entry
	Luca 28
	Bela 31
	Robert 30
	Legend:
	Table Walk Order

Table 6. SNMP Tables vs. C Struct

SNMP implements basic authentication in order to prevent intruders from performing destructive operations. Authentication is based on the `community` string, which is a plain string containing a sort of password which allows the SNMP agent to identify the remote peer. In SNMPv2 further mechanisms will probably be present in order to enforce not only authentication but also privacy.

1.2.3. Management Naming Scheme: Object Identifiers

Object identifiers (OIDs) are used to identify uniquely objects relevant for network management. They are used by both OSI and Internet and are defined in ASN.1 as a sequence of integers usually represented in a human-friendly format such as integers separated by periods (for instance 1.3.6.1). OIDs have been designed to be unique and flexible, allowing standards organisations and companies to define their own OIDs in a unique way. Therefore, OIDs form a tree structure as represented in the following figure.

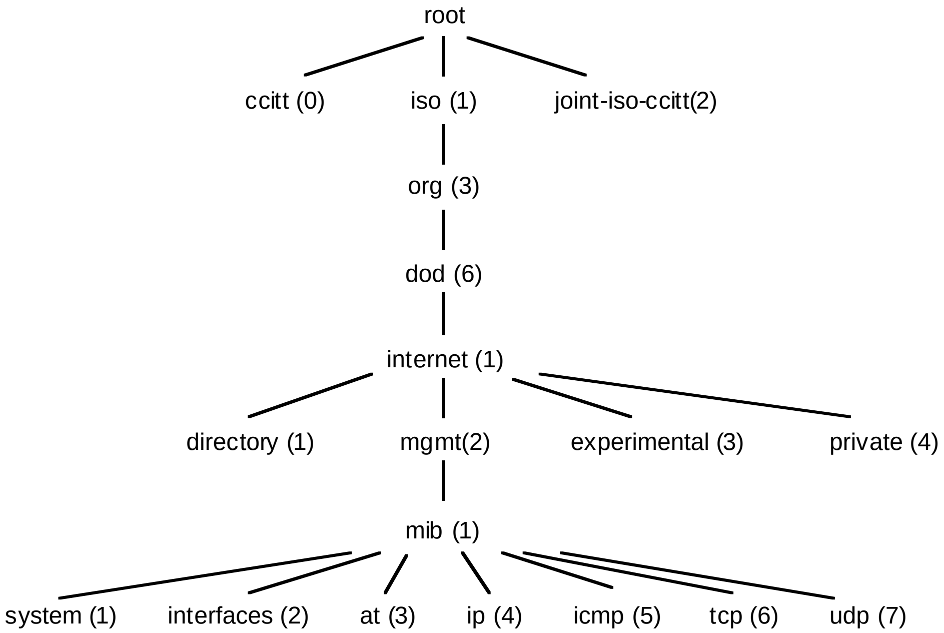


Figure 2. OID Tree Structure

Companies that need to assign OIDs must register with the corresponding body (ISO for OSI, IETF for Internet) and receive a number. Supposing that a company that wants to define a SNMP MIB has received from IETF the number Y, then the company can define OIDs under the subtree 1.3.6.1.4.Y and is then responsible for its subtree.

B Application Side Bindings

As seen in “Application Side Bindings” on page 125, external bindings are used by management applications to communicate transparently using HTTP with a mid-level manager such as Liaison. The following sections show the bindings interface.

2.1. Diagram Notation

The diagram notation used throughout this thesis is the *unified modeling language* [Booch96], which is a unification of the Booch and OMT methods. This unification provides the basis for a de facto standard for object-oriented analysis.

The unified method is a design method for specifying, visualising, and documenting object-oriented systems under development. It defines a number of diagrams, which are projections upon the elements of a model:

1. class diagram,
2. use case diagram,
3. message trace diagram,
4. object message diagram,
5. state diagram,
6. module diagram,
7. platform diagram.

In the remainder of this section, the class diagrams used in this thesis are examined. Class diagrams show the logical static structure of a system: its contents and their relationships to each other. *Class diagrams* show generic descriptions of possible systems. *Object diagrams* show particular instantiations of systems and their behaviour. Class diagrams contain classes whereas object diagrams contain objects, although it is possible to mix classes and objects for various purposes.

Classes are drawn as a solid line box with three compartments. The class name is placed in the top compartment, a list of attributes in the middle one and a list of operations in the bottom one. Types, initial values, and return types are optional.

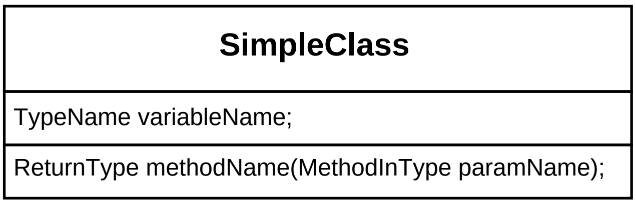


Figure 3. `SimpleClass` Class Diagram

Attributes and operations can be omitted. Omitting a compartment makes no statement about the absence of attributes and operation, whereas an empty compartment declares that there are no elements in that part.

Inheritance is the relationship between a superclass and its subclasses. Inheritance is drawn as a solid line from the subclass to its superclass with an unfilled triangular arrowhead on the superclass end.

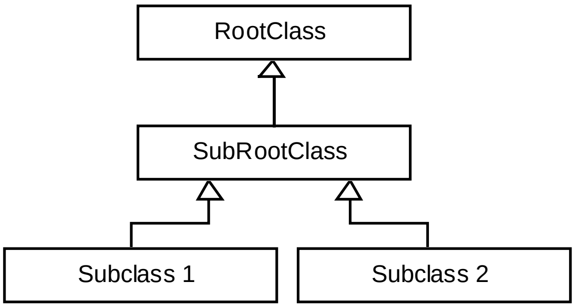


Figure 4. Class Diagram: Inheritance

Associations between classes are shown as solid lines between classes. Associations may have names and direction arrows. Each end of an association is called a *role*, which may have a name that shows how its class is viewed by the other class(es). A role also indicates the multiplicity of its class, which defines how many instances of the class can be associated with one instance of the other class.

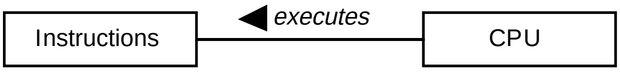


Figure 5. Class Diagram: Association

An *aggregation* relationship is indicated by placing a diamond on the role attached to the whole class.

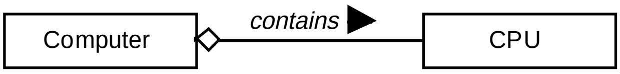
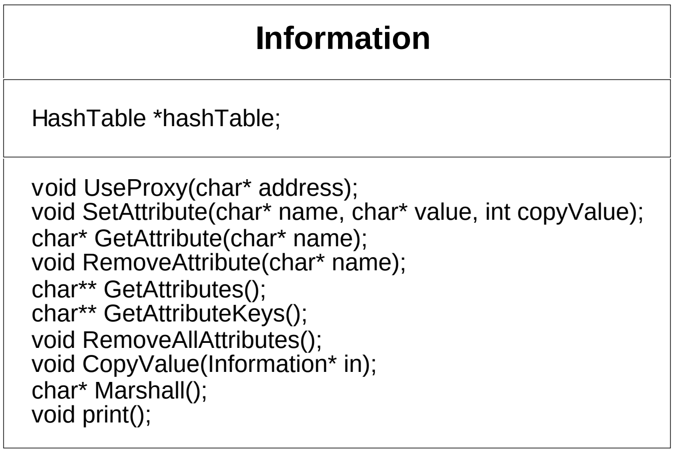


Figure 6. Class Diagram: Aggregation

2.2. Java/C++ Bindings

As stated in “Application Side Bindings” on page 125, Java and C++ bindings are quite similar. Owing to space constraints, only C++ bindings are shown.



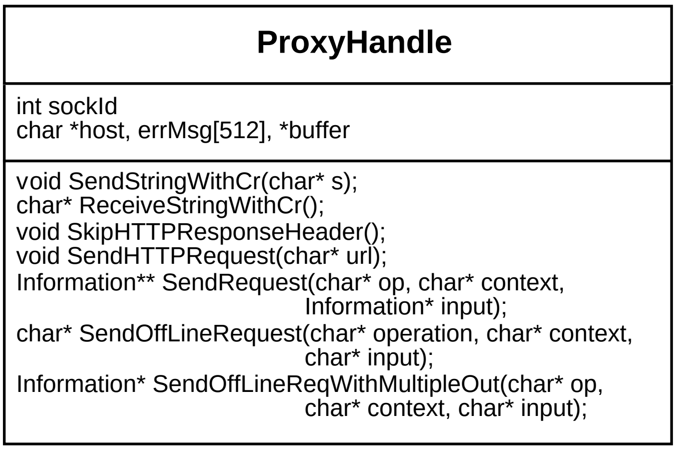
Information

HashTable *hashTable;

```
void UseProxy(char* address);
void SetAttribute(char* name, char* value, int copyValue);
char* GetAttribute(char* name);
void RemoveAttribute(char* name);
char** GetAttributes();
char** GetAttributeKeys();
void RemoveAllAttributes();
void CopyValue(Information* in);
char* Marshall();
void print();
```

Figure 7. Class `Information`

```
class Information {
protected:
  ProxyHandle *proxy;

 private:
  HashTable *hash;

  void Plain2HTMLString(char* in, char* out);

  public:
  Information();
  ~Information();
  Information* clone();
  void UseProxy(char* address);
  void SetAttribute(char* name, char* value, int copyValue=1);
  char* GetAttribute(char* name);
  void RemoveAttribute(char* name);
  char** GetAttributes();
  char** GetAttributeKeys();
  void RemoveAllAttributes();
  void CopyValue(Information* in);
  char* Marshall();
  void print();
};
```



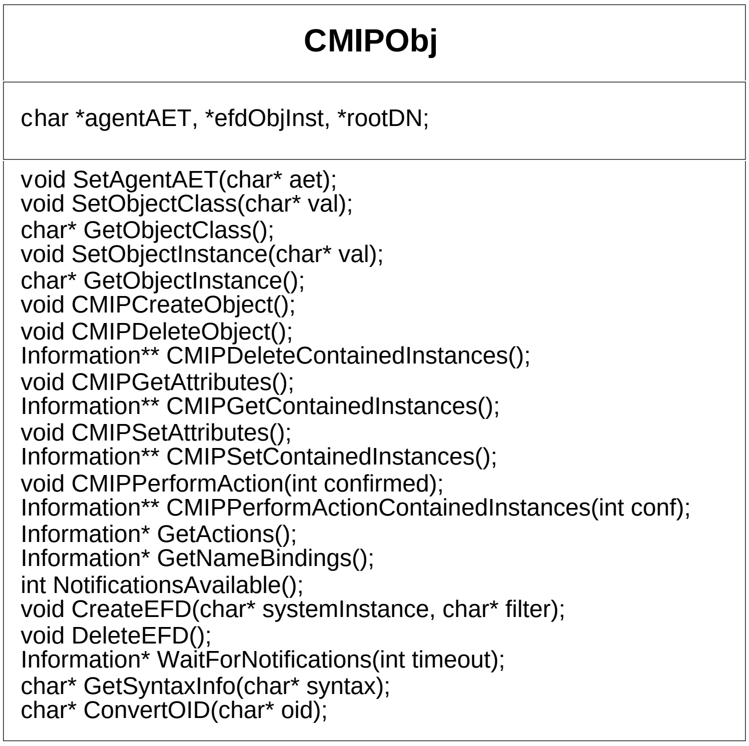
ProxyHandle

```
int sockId
char *host, errMsg[512], *buffer
```

```
void SendStringWithCr(char* s);
char* ReceiveStringWithCr();
void SkipHTTPResponseHeader();
void SendHTTPRequest(char* url);
Information** SendRequest(char* op, char* context,
                          Information* input);
char* SendOffLineRequest(char* operation, char* context,
                          char* input);
Information* SendOffLineReqWithMultipleOut(char* op,
                          char* context, char* input);
```

Figure 8. Class `ProxyHandle`

```
class ProxyHandle {

 private:
  int sockId;
  char *host, errMsg[512], *buffer;

  void SendStringWithCr(char* s);
  char* ReceiveStringWithCr();
  void SkipHTTPResponseHeader();
  void SendHTTPRequest(char* url);

public:
  ProxyHandle(char* host=NULL /* Local */) /* throw(char*) */;
  ~ProxyHandle();
  Information** SendRequest(char* operation, char* context,
                            Information* input);
  char* SendOffLineRequest(char* operation, char* context, char* input);
  Information* SendOffLineReqWithMultipleOut(char* operation,
                                             char* context, char* input);
};
```



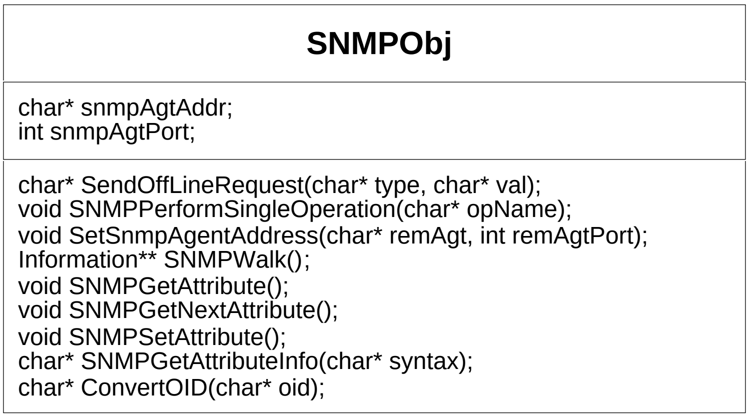
CMIPObj

char *agentAET, *efdObjInst, *rootDN;

void SetAgentAET(char* aet);
void SetObjectClass(char* val);
char* GetObjectClass();
void SetObjectInstance(char* val);
char* GetObjectInstance();
void CMIPCreateObject();
void CMIPDeleteObject();
Information** CMIPDeleteContainedInstances();
void CMIPGetAttributes();
Information** CMIPGetContainedInstances();
void CMIPSetAttributes();
Information** CMIPSetContainedInstances();
void CMIPPerformAction(int confirmed);
Information** CMIPPerformActionContainedInstances(int conf);
Information* GetActions();
Information* GetNameBindings();
int NotificationsAvailable();
void CreateEFD(char* systemInstance, char* filter);
void DeleteEFD();
Information* WaitForNotifications(int timeout);
char* GetSyntaxInfo(char* syntax);
char* ConvertOID(char* oid);

Figure 9. Class `CMIPObj`

```
#define CMIP_GET     "CMIP_Get"
#define CMIP_SET     "CMIP_Set"
#define CMIP_DELETE  "CMIP_Delete"
#define CMIP_CREATE  "CMIP_Create"
#define CMIP_ACTION  "CMIP_Action"
#define CMIP_EVR     "CMIP_Evr"
#define ASN_METADATA "ASN_Metadata"
#define OID_MAPPING  "OID_Mapping"
#define GET_ACTIONS  "Get_Actions"
#define GET_NM_BIND  "Get_NameBindings"

class CMIPObj: public Information {
private:
  char *agentAET, *efdObjInst, *rootDN;
  char* SendOffLineRequest(char* type, char* val);
  void CMIPPerformSingleOperation(char* opName, int confirmed=0);
  Information** CMIPPerformScopedOperation(char* opName, int confirmed=0);

public:
  CMIPObj(char* aet="MIBCTL");
  ~CMIPObj();
  CMIPObj* clone();
  void SetAgentAET(char* aet);
  void SetObjectClass(char* val);
  char* GetObjectClass();
  void SetObjectInstance(char* val);
  char* GetObjectInstance();
  void CMIPCreateObject();
  void CMIPDeleteObject();
  Information** CMIPDeleteContainedInstances();
  void CMIPGetAttributes();
  Information** CMIPGetContainedInstances();
  void CMIPSetAttributes();
  Information** CMIPSetContainedInstances();
  void CMIPPerformAction(int confirmed=0);
  Information** CMIPPerformActionContainedInstances(int confirmed=0);
  Information* GetActions();
  Information* GetNameBindings();
  int NotificationsAvailable();
  void CreateEFD(char* systemInstance, char* filter);
  void DeleteEFD();
  Information* WaitForNotifications(int timeout);
  char* GetSyntaxInfo(char* syntax);
  char* ConvertOID(char* oid);
};
```



SNMPObj

char* snmpAgtAddr;
int snmpAgtPort;

char* SendOffLineRequest(char* type, char* val);
void SNMPPerformSingleOperation(char* opName);
void SetSnmpAgentAddress(char* remAgt, int remAgtPort);
Information** SNMPWalk();
void SNMPGetAttribute();
void SNMPGetNextAttribute();
void SNMPSetAttribute();
char* SNMPGetAttributeInfo(char* syntax);
char* ConvertOID(char* oid);

Figure 10. Class SNMPObj

```
#define SNMP_GET         "SNMP_Get"
#define SNMP_GET_NEXT    "SNMP_GetNext"
#define SNMP_SET         "SNMP_Set"
#define SNMP_WALK        "SNMP_Walk"
#define SNMP_ATTR_INFO   "SNMP_Attr_Info"
#define SNMP_OID_MAPPING "SNMP_OID_Map"

class SNMPObj: public Information {
 private:
   char* snmpAgtAddr;
   int snmpAgtPort;

   char* SendOffLineRequest(char* type, char* val);
   void SNMPPerformSingleOperation(char* opName);
public:
  SNMPObj();
  SNMPObj(char* remAgt, int remAgtPort=160);
  ~SNMPObj();
  SNMPObj* clone();
  void SetSnmpAgentAddress(char* remAgt, int remAgtPort=160);
  Information** SNMPWalk();
  void SNMPGetAttribute();
  void SNMPGetNextAttribute();
  void SNMPSetAttribute();
  char* SNMPGetAttributeInfo(char* syntax);
  char* ConvertOID(char* oid);
};
```

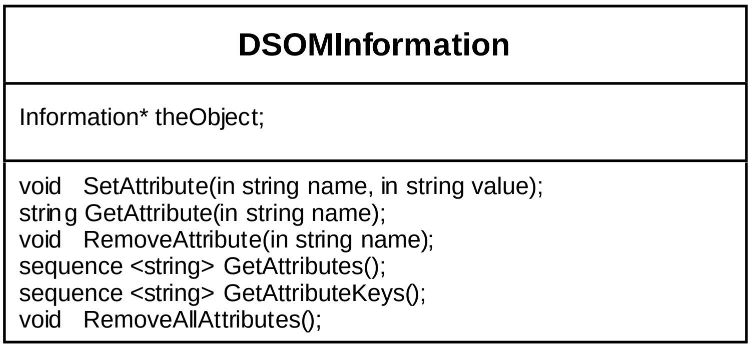
2.3. C Bindings

```
/******************* Information ************************/

void  UseProxy(char* theObj, char* address);
char* GetAttribute(char* theObj, char* oid);
void  SetAttribute(char* theObj, char* oid, char* value);
void  RemoveAllAttributes(char* theObj);
void  ResetAttributeList(char* theObj);
void  GetNextAttributeListElement(char* theObj, char** oid, char** value);
void  DeleteInformationObj(char* infoObj);

/***************** (Information**) *********************/

char* GetNextListElement(char* theObj);

/******************* SNMPObj ************************/

char* GetSNMPAttribute(char* LiaisonHost, int LiaisonPort,
                             char* SNMPAgentHost, int SNMPport,
                             char* community, char* attrId); /* Shortcut */

char* CreateSNMPObject(char* remAgt, int remAgtPort);
void  DeleteSNMPObj(char* snmpObj);

void  SetSnmpAgentAddress(char* snmpObj, char* remAgt, int remAgtPort);
void  SNMPGetAttribute(char* snmpObj);
char* SNMPWalk(char* snmpObj);
void  SNMPGetNextAttribute(char* snmpObj);
void  SNMPSetAttribute(char* snmpObj);
char* SNMPGetAttributeInfo(char* snmpObj, char* syntax);
char* ConvertSNMPOID(char* snmpObj, char* syntax);

/******************* CMIPObj ************************/

char* CreateCMIPObject(char* agentAET);
void  DeleteCMIPObj(char* cmipObj);

void  SetAgentAET(char* cmipObj, char* aet);
void  SetObjectClass(char* cmipObj, char* val);
char* GetObjectClass(char* cmipObj);
void  SetObjectInstance(char* cmipObj, char* val);
char* GetObjectInstance(char* cmipObj);
void  CMIPCreateObject(char* cmipObj);
void  CMIPDeleteObject(char* cmipObj);
char* CMIPDeleteContainedInstances(char* cmipObj);
```

```
void  CMIPGetAttributes(char* cmipObj);
char* CMIPGetContainedInstances(char* cmipObj);
void  CMIPSetAttributes(char* cmipObj);
char* CMIPSetContainedInstances(char* cmipObj);
void  CMIPPerformAction(char* cmipObj, int confirmed);
char* CMIPPerformActionContainedInstances(char* cmipObj, int confirmed);
char* GetActions(char* cmipObj);
char* ConvertOID(char* cmipObj, char* oid);
char* GetNameBindings(char* cmipObj);
char* ConvertOID(char* cmipObj, char* oid);
int   NotificationsAvailable(char* cmipObj);
void  CreateEFD(char* cmipObj, char* systemInstance, char* filter);
void  DeleteEFD(char* cmipObj);
char* WaitForNotifications(char* cmipObj, int timeout);
char* GetSyntaxInfo(char* cmipObj, char* syntax);
char* ConvertOID(char* cmipObj, char* oid);
```

2.4. CORBA-Liaison Interfaces

CORBA-Liaison interfaces have been implemented using DSOM, IBM's ORB implementation. Nevertheless, as we do not rely on any specific characteristic of DSOM, similar considerations can be made for other CORBA implementations. The code fragments contained within the__SOMIDL__ preprocessor directive are hints to DSOM, and do not affect the portability of these bindings.



DSOMInformation

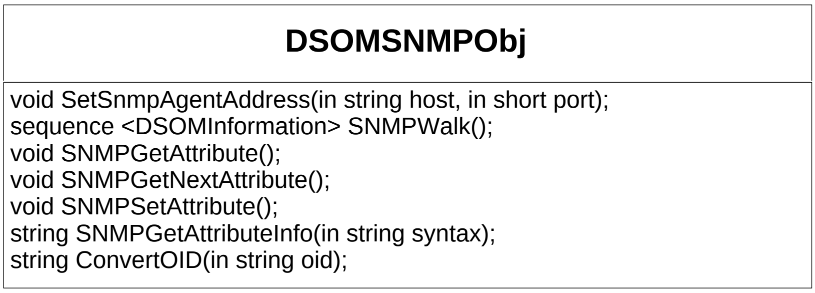
Information* theObject;

void   SetAttribute(in string name, in string value);
string GetAttribute(in string name);
void   RemoveAttribute(in string name);
sequence <string> GetAttributes();
sequence <string> GetAttributeKeys();
void   RemoveAllAttributes();

Figure 11. Interface `DSOMInformation`

```
interface DSOMInformation: SOMObject {
  void   SetAttribute(in string name, in string value);
  string GetAttribute(in string name);
  void   RemoveAttribute(in string name);
  sequence <string> GetAttributes();
  sequence <string> GetAttributeKeys();
  void   RemoveAllAttributes();

#ifdef __SOMIDL__
  implementation {
    abstract;
    somInit    : override;
    somUninit  : override;
    privateInformationPtr : noset;
    dllname="DSOMBindings.dll";
    releaseorder: SetAttribute, GetAttribute, RemoveAttribute,
                  GetAttributes, GetAttributeKeys, RemoveAllAttributes;
```

```
      Information* theObject;
      passthru C_xih = "#include \"ProxyBindings.h\""
            "extern ProxyHandle *globalProxy; /* global instance */";
   };
#endif
};
```



DSOMSNMPObj

void SetSnmpAgentAddress(in string host, in short port);
sequence <DSOMInformation> SNMPWalk();
void SNMPGetAttribute();
void SNMPGetNextAttribute();
void SNMPSetAttribute();
string SNMPGetAttributeInfo(in string syntax);
string ConvertOID(in string oid);

Figure 12. Interface `DSOMSNMPObj`

```
interface DSOMSNMPObj: DSOMInformation {
   void SetSnmpAgentAddress(in string host, in short port);
   sequence <DSOMInformation> SNMPWalk();
   void SNMPGetAttribute();
   void SNMPGetNextAttribute();
   void SNMPSetAttribute();
   string SNMPGetAttributeInfo(in string syntax);
   string ConvertOID(in string oid);

#ifdef __SOMIDL__
  implementation {
    somInit    : override;
    somUninit  : override;
    dllname="DSOMBindings.dll";
    releaseorder: SetSnmpAgentAddress, SNMPWalk, SNMPGetAttribute,
                  SNMPGetNextAttribute,  SNMPSetAttribute,
                  SNMPGetAttributeInfo, ConvertOID;
  };
#endif
};
```

DSOMCMIPObj

```
void SetAgentAET(in string aet);
void   SetObjectClass(in string value);
string GetObjectClass();
void   SetObjectInstance(in string value);
string GetObjectInstance();
void CMIPCreateObject();
void CMIPDeleteObject();
sequence <DSOMInformation> CMIPDeleteContainedInstances();
void CMIPGetAttributes();
sequence <DSOMInformation> CMIPGetContainedInstances();
void CMIPSetAttributes();
sequence <DSOMInformation> CMIPSetContainedInstances();
void CMIPPerformAction(in unsigned short confirmed);§
sequence <DSOMInformation> CMIPPerformActionContainedInstances(
                                        in unsigned short confirmed);
DSOMInformation GetActions();
DSOMInformation GetNameBindings();
unsigned short NotificationsAvailable();
void CreateEFD(in string systemInstance, in string filter);
void DeleteEFD();
DSOMInformation WaitForNotifications(in short timeout);
string GetSyntaxInfo(in string syntax);
string ConvertOID(in string oid);
```

Figure 13. Interface DSOMCMIPObj

```
interface DSOMCMIPObj: DSOMInformation {

  void SetAgentAET(in string aet);
  void   SetObjectClass(in string value);
  string GetObjectClass();
  void   SetObjectInstance(in string value);
  string GetObjectInstance();
  void CMIPCreateObject();
  void CMIPDeleteObject();
  sequence <DSOMInformation> CMIPDeleteContainedInstances();
  void CMIPGetAttributes();
  sequence <DSOMInformation> CMIPGetContainedInstances();
  void CMIPSetAttributes();
  sequence <DSOMInformation> CMIPSetContainedInstances();
  void CMIPPerformAction(in unsigned short confirmed);§
  sequence <DSOMInformation> CMIPPerformActionContainedInstances(
                                         in unsigned short confirmed);
  DSOMInformation GetActions();
  DSOMInformation GetNameBindings();
  unsigned short NotificationsAvailable();
  void CreateEFD(in string systemInstance, in string filter);
  void DeleteEFD();
  DSOMInformation WaitForNotifications(in short timeout);
  string GetSyntaxInfo(in string syntax);
  string ConvertOID(in string oid);

#ifdef __SOMIDL__
  implementation {
    somInit    : override;
    somUninit  : override;
    dllname="DSOMBindings.dll";
    releaseorder: SetAgentAET, SetObjectClass, GetObjectClass,
      SetbjectInstance, GetObjectInstance, CMIPCreateObject,
      CMIPDeleteObject, CMIPDeleteContainedInstances, CMIPGetAttributes,
      CMIPGetContainedInstances, CMIPSetAttributes,
      CMIPSetContainedInstances, CMIPPerformAction,
      CMIPPerformActionContainedInstances, GetActions, GetNameBindings,
      NotificationsAvailable, CreateEFD, DeleteEFD, WaitForNotifications,
      GetSyntaxInfo, ConvertOID;
  };
#endif
};
```

C Implementation Issues

This chapter evaluates the performance of Liaison, describes how droplets have been implemented in Liaison, and contains code fragments of some parts of Liaison that can be used to learn how the application has been implemented.

IMPORTANT

3.1. Evaluating Liaison

In order to evaluate Liaison's features and performance it is necessary to find a set of applications against which Liaison will be compared. Because Liaison implements many types of functionality ranging from simple network browsing using a Web browser to advanced CORBA management, the comparisons below are listed according to the different categories. For each category, Liaison is compared with a commercial or state-of-the-art application.

Web-based Management

In this area there are no applications supporting both CMIP and SNMP as Liaison does. Therefore Liaison is compared just for the CMIP side with a commercial browser produced by IBM called MBE and part of the IBM TMN product suite.

	Liaison	IBM TMN MBE
Architecture	Client/Server (Liaison/Web browser)	Monolithic (MBE does all)
Server Platform	AIX, OS/2, MacOS, Win95/NT, Linux	Not applicable.
Client Platform	Virtually All (the client application is a web browser)	AIX
Prerequisites	IBM OSI Stack	IBM OSI Stack, X11 Server

	Liaison	IBM TMN MBE
Configuration	None	Required (it must be performed by the system administrator)
Application Size (including shared libraries)	≈900 Kb (Liaison: only the CMIP side)	5.7 Mb (not including OSF Motif™ libraries)
Memory Usage	1 Mb	3.5 Mb
User Customisation	Yes	Very Limited (Window Colours)
Security	HTTP Security and SSL (Secure Socket Layers)	None
Scriptability[a]	Yes	Very Limited (macro)
Throughout (operations per second)	Limited by the OSI stack	1
Wizard	Yes (M-CREATE, M-ACTION)	No
Metadata Access	Yes	Yes
Metadata Configuration	No	Yes (it must be performed by the system administrator)
Multiple Concurrent Requests	Yes	No
Remote Access	Yes	No

[a] Application scriptability is the ability to use a scripting language that allows batch tasks to be executed.

Table 7. CMIP browser comparison: Liaison vs. IBM TMN MBE

From the table above it is evident that web-based management has several advantages with respect to conventional applications especially in terms of ease of customisation and security. Nevertheless the major advantage of Liaison is due to its client/server architecture. Liaison's client is a conventional web browser available on virtually every platform, whereas IBM's MBE runs only on AIX. In addition, Liaison is capable at exploiting AIX features such as multithreading which are not used by MBE. This is a major advantage especially in terms of performance, because Liaison's performance is limited only by the OSI stack, whereas MBE cannot issue more than one request at a time. Finally, thanks to Yasmin, Liaison's memory requirements are very limited, the application is slim, and it allows multiple clients to connect to the same Liaison application. This is a great advantage in large organisations where multiple users can share the same Liaison instead of each having to start an application, as it happens in the case of IBM's MBE.

Java Management

As in the previous case, it is not possible to compare Liaison with implementations that support both CMIP and SNMP because there are none available. Thus, Liaison will be compared with the market leader application for SNMP management. Liaison is not compared with the JMAPI

[JMAPI], because at the moment version 1 has not yet been released.

	Liaison's Java Bindings	AdventNet[a]
Architecture	Client/Server (Liaison/Java Applet)	Monolithic (everything is implemented in the Java Size)
Server Application Size	480 Kb (Liaison: only the SNMP side)	0 Kb (everything is implemented in Java)
Java Classes Size	9 Kb	81 Kb
Configuration	None (New MIBs are compiled automatically)	
Supported SNMP Version	V1	V1 (V2 in a future release)
Memory Usage at the Client Side	Minimal (this is because the Java classes delegate the processing to Liaison)	Medium (the client must load MIBs information)
Load on the Client Side due to SNMP	Minimal (this is because the Java classes delegate the processing to Liaison)	Medium (all the computations are performed on the client size)
Throughout (operations per second)	High (Limited by the bandwidth available between Liaison/Applet)	Low/Medium (limited by the speed of the Java VM)
Metadata Access	Yes	No
Ability to mix applets with HTML	Yes	No
SNMP Programming Skills Required	Minimal (developers deal only with strings which are converted transparently to real types)	Advanced (developers must know exactly the type of the MIB attributes they are manipulating)

[a] The AdventNet web site address is http://www.adventnet.com/.

Table 7. Java-based management comparison: Liaison vs. AdventNet

The main difference between the two implementations is the architecture. Liaison implements SNMP in the server side whereas the client side is very light in order to:

- have short client application download time;
- use a small amount of memory on the client side, which is usually a web browser with limited resources;
- create small client applications.

AdventNet instead is a classic monolithic application which implements everything in Java, providing high portability and platform independence. Currently, the performance of Java VMs are not outstanding hence both implementations suffer from poor performance. Nevertheless Liaison performs differently with respect to AdventNet because Liaison performs most of its computations on the server side, a fast multithreaded application, whereas AdventNet is implemented completely in Java. As seen before, SNMP is suitable for LAN management, and LAN response times are usually fast. This means that in the case of Liaison, the performance loss due to client/server communication is balanced by the much higher throughput that Liaison has with respect to AdventNet. On average the performance of

Liaison vs. that of AdventNet is comparable, but when the management application has to perform several operations at once or manage several devices, Liaison's performance is better than that of AdventNet. In addition, if Liaison's performance is not sufficient, it is possible to start multiple Liaison instances and distribute client requests to those applications in order to obtain a performance improvement proportional to the number of active Liaison applications.

In conclusion, although Liaison needs to be ported on a specific platform in order to be used, Liaison's client/server application offers a better performance than AdventNet, provides scalability, minimal client download time, and low memory requirements on the client side.

NOTE

Because Liaison is accessible using the standard HTTP protocol, it is not necessary to port Liaison on every platform, but it is sufficient to be able to find a machine attached to the LAN running Liaison. There is a high probability that the machine exists because Liaison has been already ported on most of the platforms currently being used.

CORBA-based Management

Comparing Liaison with similar efforts undertaken in the area of CORBA-based management is not an easy task because there is a lot of information available concerning implementation issues but very little regarding the measurement of application performance. Therefore Liaison is compared only with [Genilloud96] and only with respect to CMIP, which contains some information about memory requirement and application performance.

	Liaison's CORBA Bindings	Proxy-agent [Genilloud96]
Architecture	Client/Server (Liaison/CORBA client)	Client/Server (Proxy-agent/CORBA client)
Memory Usage at the Server Side	≈700 Kb (related to the CMIP CORBA-related droplets)	3.4 Mb (it includes support for the AMT-C-System MOC only)
Memory Usage at the Client Side	≈100 Kb (regardless of the type/number of managed objects)	650 Kb (it includes support for the AMT-C-System MOC only)
Server Application Size	≈900 Kb (only the CMIP side)	≈3 Mb (this is a speculation considering the memory requirements)
Client Application Size	≈80 Kb (regardless of the type/number of managed objects)	≈500 Mb (this is a speculation considering the memory requirements)
Memory required for the Creation of a Simple CORBA Proxy Object	A few Kb (regardless of the managed object class)	2 Mb
Memory Needed to Provide Support for a MOC	None (regardless of the type/number of the MOC attributes)	204 Kb

Table 8. CORBA-based management comparison: Liaison vs. Genilloud's Proxy-agent

It is not possible to evaluate application performance owing to the lack of performance data concerning Proxy-agent and because of the inability to compare the two implementations on a fair scenario. The reason for this is mainly related to the different OSI stacks and CORBA implementations being used. In general Proxy-agent's performance should be slightly better that Liaison's, because Liaison performs dynamic CMIP-CORBA translation which produces a limited runtime overhead. In general the performance of the two implementations should be equivalent because Liaison's CMIP-CORBA translation is very simple; hence this overhead is very limited. In the tests I have conducted, the overhead due to the translation is much less than 10% of the total time needed by an OSI client to issue the same operation without any translation, hence the performance is quite reasonable.

Conclusion

From the above comparisons, it results that:

- the benefits of Yasmin with respect to conventional applications do not have a negative impact on the performance of Yasmin-based applications such as Liaison;
- the use of droplets limits the application size and the memory requirements as droplets are loaded only when needed;
- Yasmin has really overcome problems such as monolithic application structure and limited tailoring without introducing major limitations;
- the client/server architecture of Liaison-based applications (for instance for Java-based management) has several advantages in terms of scalability and limited client requirements, which abundantly balance the minimal performance loss due to the client/server communications.

In conclusion, this section demonstrates that Yasmin has kept its promises not only in terms of architecture but also in terms of application requirements and net performance.

3.2. From Theory to Practice: Implementing Droplets

Section "Droplets" on page 80 covers droplets. It defines a droplet, describes its properties, shows how droplets look, and how applications use the services implemented inside them. This section shows how Liaison's droplets are implemented explaining step by step what developers have to do in order to create a new droplet which can be added to their copy of Liaison downloaded from the network.

What do I have to store inside a droplet?

A droplet must contain services and functionality that are of general use. Private routines and internal information should not be exported outside the droplet because there is no need of doing so. Remember: keep the droplet simple and avoid showing private information which may be altered in future versions of the droplet to external applications.

What does a droplet look like?

The skeleton of a droplet is the following.

```
#include "ProxyBasicIncludes.h"
#include "Proxy.h"

static JumpInfo jumpInformation[] =
{
  TOOL_VERSION,
  LOAD_ON_DEMAND,
  INFINITE_LIFETIME,
  "This is the droplet name", /* For instance /SNMP/GET */
  "This field describes what this droplet does",
  InitProc,  /* Pointer to the function called when the droplet is first loaded */
  TermProc,   /* Pointer to the function called when the droplet is unloaded */
  ExternFunction, /* Pointer to droplet's main function
  RemServices, /* Pointer to the list of services available from remote */
  LocServices /* Pointer to the list of services available only locally */
};

JumpInfo* JumpProc() { return(jumpInformation); }

static void ExternFunction(int peerSocket, char* reqString,
                    char* queryString, int httpdPort)
{
  /* This is the droplet's main function */
}
```

Supposing we develop a Liaison's droplet for issuing a function that selects records from an SQL database, the droplet name will be `/SQL/SELECT`, the `ExternFunction` function will parse the HTTP arguments and return the response over the socket `peerSocket` used by the remote client to issue the request.
If the droplet has to issue the following SQL command "`SELECT cust_num, cust_name FROM customer WHERE cust_num < 5100 ORDER BY cust_name;`" and the developer believes that another droplet may need to issue the SQL SELECT command, it is wise to create a service called `SQL_SELECT` that allows a droplet to issue the SQL SELECT command. In this case `LocServices` will be defined as follows:

```
static LocalService locServices[] = {
  { SQL_SELECT,   /* Pointer to the function which implements the service */
    "SQL_SELECT", /* Name of the service */
    "Service which implements the SQL SELECT",
    "", /* Write here what are the input parameters */
    ""  /* Write here what the service returns */ },
};
```

In case the service is accessible remotely, `RemService` will contain the following entry:

```
static RemoteService remServices[] = {
 { SQL_SELECT_REMOTE,  /* Pointer to the function which implements the service */
    "SQL_SELECT", /* Name of the service */
    "Service which implements the SQL SELECT",
    "", /* Write here what are the input parameters */
    ""  /* Write here what the service returns */ },
};
```

NOTE

The implementation of SQL_SELECT is different from the SQL_SELECT_REMOTE implementation because in the remote version the input/output parameters (if present) are strings that contain the input/output in a marshalled form. Basically SQL_SELECT_REMOTE first has to unmarshall the input parameter, then call SQL_SELECT, and finally marshall the response.

What are the services exported by a droplet?

Owing to the droplet structure, the only global symbol that can be exported is JumpProc(), which returns the pointer to the droplet entry point. Readers may ask why the droplet entry point is not directly jumpInformation. The reason is that some operating systems allow as shared library entry point only a function and not a symbols such as jumpInformation.
In any case, if the droplet has several symbols that can be exported (i.e. they are not static), the linker prevents their export because the only exported symbol is JumpProc__Fv, which corresponds to JumpProc(). Even supposing that the user wants to export further symbols, Liaison’s droplet manager loads the droplets and searches for JumpProc(), which is always used as entry point.

Conclusion

Building droplets is relatively simple. Things are even simpler if a droplet wizard, which drives the developers through droplet development, is used. Droplet consistency and adherence to the rules is enforced by the linker which exports only one global symbol, JumpProc(), and which prevents a droplet from being built if the information contained in the droplet interface, jumpInformation, is not correct.

3.3. Liaison’s Code Fragments

This chapter contains some files used to implement Liaison. The first two files are the core files Proxy.h and Proxy.cpp, which implement the core part of Liaison. Then a file containing a personality is shown. Finally an example containing a droplet is presented. Please note that these files cannot be used to implement a Liaison-like application, but are included only for non-commercial research purposes.

Proxy.h

```
#ifndef _PROXY_H_
#define _PROXY_H_

class Proxy;

#include "ProxyGlobals.h"
#include "ProxyService.h"
#include "ProxyTable.h"
#include "ProxyTypes.h"
#include "ProxyTools.h"
#include "ProxyMsg.h"
```

```
class JumpEntry; /* Forward Declaration */

#ifndef STHREAD
class ThreadController {
public:
  CondVariable *condVariable;

  ThreadController();
  ~ThreadController();
  int GetNumEntries();
  void PushEntry(int, char*);
  void PopEntry(int*, char*);

private:
  RequestBkt waitingSockets[32];
  int popIdx, pushIdx, numEntries;
  MutexSemaphore *threadMutex;
};
#endif

class SecurityEntry {
public:
  char *header, *trailer, *user;
  SecurityEntry();
  ~SecurityEntry();
};


class Proxy
{
private:
  FILE *logFile, *httpLogFile;
  char *localHostName;
  short callUpdateTools, numProtectedURLs;

#ifndef STHREAD
  long toolUpdateThreadId;
#endif
  int reloadDroplets;
  int httpDaemonPort; /* Port used by the default HTTPd */
  RPCSockStruct rpcSockets[MAX_RPC_SOCKETS];
  HashTable *passwds, *hostAddresses;
  SecurityEntry *protectedURLs[48];

  /* Tool management infos */
  ToolMgr toolMgr;
  EventQueue *eventQueue;
  EventHandlerEntry evtHdlr[MAX_HANDLED_EVENT+1];

  void InitSNMP();
  void ReadPasswordFile();
  int  LoadTools();
  void UnloadTools();
  void InstallBasicEvtHandlers();
  void ProcessQueuedEvent();
  void ProcessRPCRequest();
  void WriteLogEntry(FILE* file, char* msg);
  int  DecodeString(char *bufcoded, unsigned char *bufplain, int outbufsize);
  int  ClientAuthorized(char*, char*, char*);
  int  ReadURLConfigurationFile(char* name);
  int  ReadPasswordConfigurationFile(char* name);

public:
  unsigned short numHttpThreads;
  int httpSocket, sendUDPSocket, sendUDPSocketPort, rpcSocket;
  int rpcSocketPort, sendTCPSocket, sendTCPSocketPort;
  Personality activePersonality;
#ifndef STHREAD
  ThreadController *httpThreadController;
#endif

  Proxy();
  ~Proxy();
#ifndef STHREAD
  MutexSemaphore mutexUpdateSem, mutexTCPSocket, mutexUDPSocket, mutexLoadTools;
#endif

  void Listen();
  void ProcessRequest(int newSockFd, char* udpBuf);
  void ToggleToolsUpdate(int value) { callUpdateTools=value; }
  inline int  GetToolsUpdateState() { return(callUpdateTools); }
  int UpdateTools();
  int LoadTool(ToolInfo*); /* Explicitly load a tool */
  void UnloadToolByName(char* name, int removeFromFileList=1);

  /* Services available for other classes */
  int GetHTTPDaemonPort() { return(httpDaemonPort); }
  int OpenTcpSocket(int* port);
  int OpenUDPSocket(int* port);
  void PrintSocketError(int err);
  int TranslateBody(int, int, char*, Symbols*, char[], int*,
                    int*, char*[], char[]);
  Symbols GetToken(int, int, char[], int *indentLevel, char[]);
  char* Type2Name(int typeValue, int* indentLevel, short* simpleType);
  int InsertAnchor(char* name, int* theAnchorIdx, char* theAnchorList[]);
  int SendString(int peerSocket, char* theString);
  inline char* GetLocalHostName() { return localHostName; };
  inline int GetTCPSocketPort() { return sendTCPSocketPort; };
  inline int GetTCPSocket()     { return sendTCPSocket; };
  inline int GetUDPSocketPort() { return sendUDPSocketPort; };
  inline int GetUDPSocket()     { return sendUDPSocket; };
#ifndef STHREAD
  inline MutexSemaphore* GetTCPSocketMutex() { return(&mutexTCPSocket); };
  inline MutexSemaphore* GetUDPSocketMutex() { return(&mutexUDPSocket); };
  inline MutexSemaphore* GetUpdateMutex()    { return(&mutexUpdateSem); };
#endif

  /* Local/Remote Service facilities */
  int CallLocalService(char* serviceName, void* in_data, void* out_data);
  int CallService(Message* theMsg, char* remoteHostName,
                  int remoteHostPort, int sockWithRemPeer);
  void HandleRemoteServiceCall(InParam* parm, OutParam*, int sockWithRemPeer);
  void ProcessRPCRequest(int sockFd);
  /* whoever can dynamically provide a service */
  void ProvideLocalService(LocalService*);
  void ProvideRemoteService(RemoteService*);

  /* Event facilities */
  int InstallEventHandler(int componentId, int eventType,
                          EvtHandlerProc eventHandler);
  int RemoveEventHandler(int componentId, int eventType);
  int PostEvent(EventInfo* eventInfo);

  /* Support facilities */
  void LogMsg(char* msg);
  int  HTML2PlainString(char* in, char* out);
  void Plain2HTMLString(char* in, char* out);
  void Plain2HTMLDisplayString(char* in, char* out);
  void PrintConnectionRefusedError(int);
  void PrintInvalidDataError(int);
  int hex2int(char c);
  char* Err2Str(int errId);
  int IsAnOID(char* theStr);
  int IsOctetString(char*);
  char* OctStr2Long(char*, char*, int);
  char* OctStr2Real(char*, char*, int);
  char* OctStr2String(char*, char*);

  int IssueMIBRequest(void*, char*, int, int);
  void Lookup2HTML(int peerSocket, int infraSock, char* reqString);
  inline ToolInfo* GetDroplet(char* toolName)
        { return(toolMgr.GetToolByName(toolName)); }
};

Proxy* GetProxy();
#endif /* Proxy.h */
```

Proxy.cpp

```
#include "ProxyBasicIncludes.h"
#include "Proxy.h"

extern char shortVersStr[], applName[], threadSupport[], localDirString[];
static int numRunningComponents;
static Proxy* _HTTPproxy;


#define ENABLE_LOG
//#define THREAD_DEBUG

/* Prototypes */
void IncrementNumComponents();
void DecrementNumComponents();
int readline(int sockId, char* buffer, int bufferSize);
int sendstring(int sockId, char* buffer, int bufferSize);
void RemoveJunk(char* str);

typedef struct {
  int    theSocket;
  char theDataSocket[256];
} ThreadArgs;


typedef struct {
  RPCSockStruct *socks;
  int    sockId;
} RPCThreadArgs;

/* Function prototypes */
extern "C" {
  THREAD_RET_TYPE ServeRPCRequest(void* thrArgs);
  THREAD_RET_TYPE CheckToolDirectory(void* notUsed);
  THREAD_RET_TYPE CreateHTTPServer(void* notUsed);
}
/* End of Function prototypes */


void IncrementNumComponents()
{
#ifndef STHREAD
  (_HTTPproxy->mutexUpdateSem).access(); /* Block tools update process */
  numRunningComponents++;
  (_HTTPproxy->mutexUpdateSem).release(); /* Enable tools update process */
```

```
#endif
}

void DecrementNumComponents()
{
#ifndef STHREAD
  (_HTTPproxy->mutexUpdateSem).access(); /* Block tools update process */
  numRunningComponents--;
  (_HTTPproxy->mutexUpdateSem).release(); /* Enable tools update process */
#endif
}

#ifndef STHREAD
THREAD_RET_TYPE CreateHTTPServer(void*)
{
  static int thId=0;
  int sockId, _thId=thId++;
  char udpBuf[256];

#ifdef THREAD_DEBUG
  printf("CreateHTTPServer\n");
#endif

  _HTTPproxy->activePersonality.SetThreadAttributes();

  for(;;)
    {
      if(_HTTPproxy->httpThreadController->GetNumEntries() == 0)
          _HTTPproxy->httpThreadController->condVariable->Wait();

      if(_HTTPproxy->httpThreadController->GetNumEntries() > 0)
          _HTTPproxy->httpThreadController->PopEntry(&sockId, udpBuf);
      else
          sockId = -1;

if(sockId != -1)
{
#ifdef THREAD_DEBUG
  char tmpStr[32];

  sprintf(tmpStr, "Processing Request (%d) [Id=%d]", sockId, _thId);
  _HTTPproxy->LogMsg(tmpStr);
#endif
  _HTTPproxy->ProcessRequest(sockId, udpBuf);
#ifdef THREAD_DEBUG
  sprintf(tmpStr, "Completed processing (%d) [Id=%d]", sockId, _thId);
  _HTTPproxy->LogMsg(tmpStr);
#endif
}
#ifdef THREAD_DEBUG
    else
     _HTTPproxy->LogMsg("Nothing to do");
#endif
    }

  _HTTPproxy->activePersonality.CleanUpThread();
  _HTTPproxy->LogMsg("Leaving Thread...");

  THREAD_RET_VALUE
}
#endif

THREAD_RET_TYPE ServeRPCRequest(void* thrArgs)
{
  RPCThreadArgs *args=(RPCThreadArgs*)thrArgs;

  IncrementNumComponents();

  args->socks[args->sockId].blocked = 1; /* Nobody else can use it */
  _HTTPproxy->ProcessRPCRequest(args->socks[args->sockId].sockId);
  args->socks[args->sockId].blocked = 0; /* Unlocked */

  DecrementNumComponents();

  THREAD_RET_VALUE
}

THREAD_RET_TYPE CheckToolDirectory(void* notUsed)
{
  for(;_HTTPproxy->GetToolsUpdateState();)
    {
      DoSleep(SLEEP_TIME_TOOLS_UPDATE);

      // _HTTPproxy->LogMsg("About to update tools...");

#ifndef STHREAD
      (_HTTPproxy->GetUpdateMutex())->access(); /* Block tools update process */

if(numRunningComponents == 0)
{
  // _HTTPproxy->LogMsg("Updating tools...");
  _HTTPproxy->UpdateTools();
}

/* Enable tools update process */
(_HTTPproxy->GetUpdateMutex())->release();
#else
      _HTTPproxy->UpdateTools();
#endif
      /* LogMsg("End update..."); */
    }

  THREAD_RET_VALUE
}


Proxy::Proxy()
{
  int i, basePort;
  char tmpStr[256];
  char* _basePort = getenv("BASE_PORT");

   printf("Welcome to %s v.%s [%s%s personality]\nWritten by Luca Deri (lde@zu-
rich.ibm.com)\n"
 "Copyright IBM 1995-96, All Rights Reserved.\n\n",
 applName, shortVersStr, personalityName, threadSupport);

  logFile = stdout;
  httpLogFile = fopen("http.log", "a");

  hostAddresses = new HashTable(256, FREE_DATA_AND_KEY_MEMORY);
  callUpdateTools = 1; /* Tools have to be updated */
  numHttpThreads  = 8; /* # threads used to serve HTTP requests */

  LogMsg("Initialization. Wait please...");

  _HTTPproxy = this;
  numRunningComponents = 0;
  eventQueue = new EventQueue;

  /* Reset Event Handler List */
  for(i=0; i<=MAX_HANDLED_EVENT; i++)
    evtHdlr[i].theEvtHandler = (EvtHandlerProc)NULL;

  tmpStr[0] = '\0';
  gethostname(tmpStr, 256);
  localHostName = _strdup(tmpStr);

  if(_basePort != NULL)
    {
      basePort = atoi(_basePort);
      if(basePort < 1024)
{
  LogMsg("The base port you've selected is < 1024. The default port "
         "is used instead.");
  basePort = RECORDING_PORT;
}
      else
{
  sprintf(tmpStr, "The base port is %d [HTTP port is %d]", basePort, basePort+2);
  LogMsg(tmpStr);
}
    }
  else
    basePort = RECORDING_PORT;

  /* Socket used for the communications with the remote peers */
  i=basePort+2 /* HTTP Listen port */;
  if((httpSocket = OpenTcpSocket(&i)) < 0)
    {
      LogMsg("Please check whether there's a Proxy already running");
      LogMsg("Fatal error. Shutting down...");
      exit(-1);
    }

  /* Socket used for RPC-like communications */
  rpcSocketPort=basePort+1 /* RPC_PORT */;
  if((rpcSocket = OpenTcpSocket(&rpcSocketPort)) < 0)
    {
      LogMsg("Please check whether there's a Proxy already running");
      LogMsg("Fatal error. Shutting down...");
      exit(-1);
    }

   /* Socket used for the communications with the remote peers */
  sendTCPSocketPort=0, sendTCPSocket=0;
  if((sendTCPSocket = OpenTcpSocket(&sendTCPSocketPort)) < 0)
    {
      LogMsg("Please check whether there's a Proxy already running");
      LogMsg("Fatal error. Shutting down...");
      exit(-1);
    }

 /* Socket used for the communications with the YP server and IT */
  sendUDPSocketPort=basePort /* RECORDING_PORT */;
  if((sendUDPSocket = OpenUDPSocket(&sendUDPSocketPort)) < 0)
    {
      LogMsg("Please check whether there's a Proxy already running");
      LogMsg("Fatal error. Shutting down...");
      exit(-1);
    }
  InstallBasicEvtHandlers();

  if(LoadTools() != 0)
    {
      LogMsg("Problems loading external tools.");
      return;
    }

  ReadPasswordFile();

  if(getenv("DROPLET_RELOAD") != NULL)
    reloadDroplets = 1;
  else
    reloadDroplets = 0;

#ifndef STHREAD
  /* NOTE:
     In OS/2 the DLLs cannot be modified while they are in use
     and therefore only new tools can be added. Due to this a
     thread can load such tools and do not care about anything
     else. */

 if(reloadDroplets)
  activePersonality.CreateThread(CheckToolDirectory, NULL, &toolUpdateThreadId);
#endif

  char *httpdPort = getenv("HTTP_SRVR_PORT");

  if(httpdPort != NULL)
    httpDaemonPort = atoi(httpdPort);
  else
    httpDaemonPort = 80; /* Default Port */

#ifdef LEAKS
  InitProc();
#endif
  LogMsg("Proxy ready to serve requests.");

#ifndef STHREAD
  httpThreadController = new ThreadController();

  for(i=0; i<numHttpThreads; i++) /* 8 = max num threads active for HTTP */
    activePersonality.CreateThread(CreateHTTPServer, NULL, NULL);
#endif
}


Proxy::~Proxy()
{
  static int calledDestructor=0;
  int i;

  if(calledDestructor == 0)
    {
      LogMsg("Called Proxy::~Proxy");
      calledDestructor = 1;

      fclose(httpLogFile);

      delete localHostName;

#ifndef STHREAD
      delete httpThreadController;
```

```
#endif

#ifdef LEAKS
      TermProc();
#endif

      UnloadTools();

#ifndef STHREAD
      activePersonality.KillThread(toolUpdateThreadId);
#endif

      CloseSocket(httpSocket);
      CloseSocket(sendUDPSocket);
      CloseSocket(sendTCPSocket);
      CloseSocket(rpcSocket);

      for(i=0; i<MAX_RPC_SOCKETS; i++)
       if(rpcSockets[i].sockId != -1)
         CloseSocket(rpcSockets[i].sockId);

      LogMsg("Communication sockets closed successfully.");

      for(i=0; i<numProtectedURLs; i++)
       delete protectedURLs[i];

      delete hostAddresses;
      delete passwds;

      if(eventQueue != NULL)  delete eventQueue;
      // exit(0);
    }
}


void Proxy::Listen()
{
  int i, newSockFd, maxFd, rc;
  SOCK_LEN_TYPE length;
#ifdef STHREAD
  long expireTime;
  struct timeval wait_time;
#endif
  sockaddr_in from;
  struct fd_set sockMask;
  char udpBuf[256];
  struct sockaddr source;

  length = sizeof(from);

  for(;;)
    {
      newSockFd = -1;
      maxFd=0;
      FD_ZERO(&sockMask);

      FD_SET(httpSocket, &sockMask);
      FD_SET(sendUDPSocket, &sockMask);
      FD_SET(rpcSocket, &sockMask);

      for(i=0; i<MAX_RPC_SOCKETS; i++)
      if((rpcSockets[i].sockId != -1) && (rpcSockets[i].blocked == 0))
      {
        FD_SET(rpcSockets[i].sockId, &sockMask);
       if(rpcSockets[i].sockId > maxFd) maxFd = rpcSockets[i].sockId;
     }

      if(httpSocket > maxFd) maxFd = httpSocket;
      if(sendUDPSocket > maxFd) maxFd = sendUDPSocket;
      if(rpcSocket > maxFd) maxFd = rpcSocket;

      maxFd++; /* Necessary for select */

#ifdef STHREAD
      expireTime = eventQueue->GetNextEventExpireTime();

      if((reloadDroplets != 0) || (expireTime != -2 /* No Events */))
{
  if(expireTime == -1)
    {
      /* Process the event */
      ProcessQueuedEvent();
      continue;
    }
  else if(expireTime != -2)
    {
      expireTime = expireTime-time(NULL);
      if(expireTime < 0)
     {
     /* Process the event */
     ProcessQueuedEvent();
    continue;
    }
  else if(expireTime > SLEEP_TIME_TOOLS_UPDATE)
    expireTime = SLEEP_TIME_TOOLS_UPDATE;
    }
  else
    expireTime = SLEEP_TIME_TOOLS_UPDATE;

  wait_time.tv_sec = expireTime, wait_time.tv_usec = 0;
  rc = select(maxFd, &sockMask, 0, 0, &wait_time);
}
else
  rc = select(maxFd, &sockMask, 0, 0, NULL /* Infinite timeout */);
#else
  rc = select(maxFd, &sockMask, 0, 0, NULL /* Infinite timeout */);
#endif

#ifndef STHREAD
/* These instructions are needed to block the current request until
   the UpdateTools() is in progress */
(_HTTPproxy->GetUpdateMutex())->access(); /* Block tools update process */
(_HTTPproxy->GetUpdateMutex())->release();
#endif

if(rc > 0)
  {
  if(FD_ISSET(httpSocket, &sockMask))
    {
      newSockFd = accept(httpSocket, (sockaddr*)&from, &length);

      if(newSockFd < 0)
       continue; // Accept failed
      else
       udpBuf[0] = '\0';

#ifdef SOCK_OPTS
      int buffer = 8192; /* 8Kb is the new socket buffer */
      (void)setsockopt(newSockFd, SOL_SOCKET, SO_SNDBUF, (char *)&buffer, sizeof(buffer));

#endif
    }
  else if(FD_ISSET(sendUDPSocket, &sockMask))
    {
      //UD* Something has arrived on the UDP socket */
      newSockFd = sendUDPSocket;

      length = sizeof(struct sockaddr);
      if((rc = recvfrom(sendUDPSocket, udpBuf, 256, 0, &source, &length)) < 0)
{
  perror("recvfrom");
  udpBuf[0] = '\0';
}
      else
udpBuf[rc] = '\0';
    }
  else if(FD_ISSET(rpcSocket, &sockMask)) /* RPC-like request */
    {
      RPCThreadArgs args;

      newSockFd = accept(rpcSocket, (sockaddr*)&from, &length);

      // DO NOT CLOSE THE SOCKET
      /* NOTE: for sure there is space because the max number of accepted
         connection is set to 8 (<MAX_RPC_SOCKETS) by the listen() call
         contained in Proxy::OpenTcpSocket */
      for(i=0; i<MAX_RPC_SOCKETS; i++)
      if(rpcSockets[i].sockId == -1)
      {
       rpcSockets[i].sockId = newSockFd;
       break;
     }

      args.socks = rpcSockets;
      args.sockId = i;

      activePersonality.CreateThread(ServeRPCRequest, (void*)&args, NULL);
      newSockFd = -1; /* Do not ask for additional processing */
    }
  else
    {
      /* Check Open RPC-Sockets */
      for(i=0; i<MAX_RPC_SOCKETS; i++)
     if((rpcSockets[i].sockId != -1)
           && (FD_ISSET(rpcSockets[i].sockId, &sockMask)))
     if(i != MAX_RPC_SOCKETS) /* Something has been found */
     {
      /* Check whether we're received data or whether the remote peer
         has closed the connection */
         rc = recv(rpcSockets[i].sockId, udpBuf, 1, MSG_PEEK /* Do not
               remove data from the socket */);
   if(rc > 0)
   {
   RPCThreadArgs args;
   /* Please note that 'newSockFd = -1' */
   args.socks = rpcSockets;
   args.sockId = i;

   activePersonality.CreateThread(ServeRPCRequest, (void*)&args, NULL);
 }
  else
{
  /* The peer has closed the connection */
  /* printf("Closing socket %d\n", rpcSockets[i].sockId); */
  CloseSocket(rpcSockets[i].sockId);
  rpcSockets[i].sockId = -1;
}
    }
  else
    LogMsg("Program check: received data from an unknown socket");
    }

  if(newSockFd != -1)
    {
      /* In this case the accept has really received a request. It might
         be happened that the alarm time has elapsed and therefore a call
         to UpdateTools has been made */

#ifndef STHREAD
      httpThreadController->PushEntry(newSockFd, udpBuf);
#ifdef THREAD_DEBUG
      LogMsg("Posted new Request");
#endif
      httpThreadController->condVariable->Signal();
#else
      ProcessRequest(newSockFd, udpBuf);
#endif
    }
}
#ifdef STHREAD
else
if(rc == 0) /* Timeout */
  {
    /****************************************/
    // (void)_HTTPproxy->CallLocalService("ReceiveNotifications", NULL, NULL);
      /****************************************/
    expireTime = eventQueue->GetNextEventExpireTime();
    if((expireTime != -2) && ((expireTime-time(NULL)) < 0)) /* Event in queue */
      ProcessQueuedEvent();
    else
      _HTTPproxy->UpdateTools();
  }
#endif
else
  perror("select failed");
    }
}


void Proxy::ProcessQueuedEvent()
{
  while(eventQueue->NumEventsAvailable() > 0)
    {
      EventInfo* theEvent;
      int eventType;

      theEvent = eventQueue->GetEvent();
      eventType = theEvent->GetEventType();

      if(!(((eventType<MIN_SYSTEM_EVENT) || (eventType>MAX_HANDLED_EVENT))
   || (evtHdlr[eventType].theEvtHandler == NULL)))
{
  /* The event handler MUST delete the event if necessary */
  evtHdlr[eventType].theEvtHandler(theEvent);
  /* LogMsg("Event processed"); */
  break; /* An event has been processed */
}
      else
LogMsg("Event discarded");
    }
}

int readline(int sockId, char* buffer, int bufferSize)
{
  int len, rc;

  for(len=0;;len++)
    {
      rc = recv(sockId, &buffer[len], 1, 0);
      if((rc != 1) || buffer[len] == '\n')
       break;
    }

  return(len);
}

int sendstring(int sockId, char* buffer, int bufferSize)
{
  int rc, len=0;

  if((bufferSize == 0) || (buffer == NULL))
    return(0);

  while(len < bufferSize)
    {
      rc = send(sockId, &buffer[len], bufferSize-len, 0);
      /* printf("rc=%d\n", rc); */

      if(rc <= 0)
        break; /* Error (peer has disconnected ?) */
      else
       len += rc;
    }

  return(len);
}

static void ReadGarbage(int sockId, char* pw)
{
  char aChar, lastChar, preLastChar, lineStr[256];
  int rc, idxChar=0;

  preLastChar = '\r';
  lastChar = '\n';

 for(;;)
    {
      rc = recv(sockId, &aChar, 1, 0);

      if(rc != 1)
      {
       idxChar=0;
       break; /* Empty line */
      }
      else
     {
      /* printf("%c", aChar); */
  if((aChar == '\n') && (lastChar == '\r') && (preLastChar == '\n'))
    {
      idxChar=0;
      break;
    }
  else
    {
      if(aChar == '\n')
     {
     lineStr[idxChar-1] = '\0';
     idxChar=0;
     if(strncmp(lineStr, "Authorization: Basic ", 21) == 0)
     strcpy(pw, &lineStr[21]);
     }
     else
     {
      if(idxChar<256)
       lineStr[idxChar++] = aChar;
     }

      preLastChar = lastChar;
      lastChar = aChar;
    }
   }
 }
}

int Proxy::ClientAuthorized(char* password,
    char* dropletName, char* urlTrailer)
{
  char outBuffer[64], *user, *pw, url[384];
  int i, rc, len, access=1 /* Access granted */;

  i = DecodeString(password, (unsigned char*)outBuffer, 64);

  if(i == 0)
    {
      user = "", pw = "";
      outBuffer[0] = '\0';
    }
  else
    {
      outBuffer[i] = '\0';

      for(i=0; i<64; i++)
if(outBuffer[i] == ':')
  {
    outBuffer[i] = '\0';
    user = outBuffer;
    break;
  }

      pw = &outBuffer[i+1];
    }

  // printf("%s-%s\n", user, pw);
  strcpy(url, dropletName);
  strcat(url, "/");
  i = strlen(url);
  strncpy(&url[i], urlTrailer, 384-i);

  // printf("%s\n", url);

  for(i=0; i<numProtectedURLs; i++)
```

```
    {
      len = strlen(protectedURLs[i]->header);
      rc = 1;

      if(strncmp(protectedURLs[i]->header, url, len) == 0)
{
  if(protectedURLs[i]->trailer[0] != '\0')
    {
      len = strlen(protectedURLs[i]->trailer)-len;
      if(strstr(&url[len], protectedURLs[i]->trailer) == NULL)
continue;
    }

  if(rc == 1)
    {
      if(strcmp(protectedURLs[i]->user, user) != 0)
{
  access = 0; /* Access Denied: in case there is not another rule
       that matches, the default is access denied */
  continue;
}
      else
{
  char* _pw = (char*)passwds->RetrieveEntry(user);

  if((_pw == NULL) || (strcmp(pw, _pw) == 0))
    return(1);
  else
    return(0);
   }
  }
 }
}
  return(access);
}

void RemoveJunk(char* str)
{
  int len = strlen(str), i, idx;
  char *out;

  out = new char[len+12 /* Just to be safe */];

  for(i=0, idx=0; i<len; i++)
    {
      switch(str[i])
{
case ' ':
case '\t':
  if((idx>0)
     && (out[idx-1] != ' ')
     && (out[idx-1] != '\t'))
  out[idx++] = str[i];
  break;
default:
  out[idx++] = str[i];
  break;
}
    }

  out[idx] = '\0';
  strcpy(str, out);
  delete out;
}
```

```
int Proxy::ReadURLConfigurationFile(char* name)
{
  FILE *fd;
  char tmpStr[512];
  int i, starIdx, nameIdx, line=0;
  SecurityEntry *se;

  if((fd = fopen(name, "r")) == NULL)
    {
      sprintf(tmpStr, "Unable to read file '%s'", name);
      LogMsg(tmpStr);
      return -1;
    }

  while(fgets(tmpStr, 512, fd))
    {
      nameIdx=-1, starIdx=-1;
      line++;
      tmpStr[strlen(tmpStr)-1] = '\0'; /* Remove final '\n' */

      switch(tmpStr[0])
{
case '\n':
case '\0':
case '#':
  break;

default:
  RemoveJunk(tmpStr); /* Remove uneeded spaces/tabs */

  for(i=0; tmpStr[i] != '\0'; i++)
    switch(tmpStr[i]) {
      {
      case '\t':
      case ' ':
        if(nameIdx == -1)
        {
          nameIdx=i;
          tmpStr[i] ='\0';
        }
        break;
      case '*':
        if(starIdx != -1) /* Double star */
        {
           sprintf(tmpStr, "Invalid line %d (file: %s) [too many stars]",
               line, name);
           LogMsg(tmpStr);
        }
        else
        {
          starIdx = i;
          tmpStr[i] ='\0';
       }
        break;
      }
    }
}

if((nameIdx != -1) && (starIdx == -1))
{
  sprintf(tmpStr, "Invalid line %d (file: %s) [missing star]", line, name);
  LogMsg(tmpStr);
}
else if((nameIdx != -1) && (tmpStr[0] == '\0'))
{
  sprintf(tmpStr, "Invalid line %d (file: %s) [star at the beginning of a line]",
line, name);
  LogMsg(tmpStr);
}
else if(nameIdx != -1)
{
  se = new SecurityEntry;
  se->header= _strdup(tmpStr);
  se->trailer = _strdup(&tmpStr[starIdx+1]);
  se->user = _strdup(&tmpStr[nameIdx+1]);
  protectedURLs[numProtectedURLs++] = se;
}
  }

  fclose(fd);
  return 0;
}


int Proxy::ReadPasswordConfigurationFile(char* name)
{
  FILE *fd;
  char tmpStr[512];
  int i, nameIdx;

  if((fd = fopen(name, "r")) == NULL)
    {
      sprintf(tmpStr, "Unable to read file '%s'", name);
      LogMsg(tmpStr);
      return -1;
    }

  while(fgets(tmpStr, 512, fd))
    {
      nameIdx=-1;
      tmpStr[strlen(tmpStr)-1] = '\0'; /* Remove final '\n' */

      switch(tmpStr[0])
{
case '\0':
case '\n':
case '#':
  break;

default:
  RemoveJunk(tmpStr); /* Remove uneeded spaces/tabs */

  for(i=0; tmpStr[i] != '\0'; i++)
    switch(tmpStr[i]) {
      {
      case '\t':
      case ' ':
if(nameIdx == -1)
  {
    nameIdx=i;
    tmpStr[i] ='\0';
  }
break;
      }
    }
}
```

```
   if(nameIdx != -1)
    passwds->AddEntry(_strdup(tmpStr), _strdup(&tmpStr[nameIdx+1]));
  }

 fclose(fd);
 return 0;
}

void Proxy::ReadPasswordFile() {

 numProtectedURLs = 0;
 passwds = new HashTable(256, FREE_DATA_AND_KEY_MEMORY);

 if((ReadPasswordConfigurationFile("passwd") == -1)
   || (ReadURLConfigurationFile("urlProtection") == -1))
  {
   LogMsg("WARNING: Security disabled.");
   return;
  }
}


void Proxy::ProcessRequest(int newSockFd, char* inData)
{
#define PAD  10
 char *queryString, *httpStringRepository,
 *reqString, tmpStr[256], password[64];
 int  operation;
 ToolInfo* toolInfo;
 struct fd_set mask;
 struct timeval wait_time;
 struct sockaddr_in peerName;
 SOCK_LEN_TYPE peerNameLen=sizeof(peerName);
 struct stat statInfo;

 IncrementNumComponents();

 // printf("--> Proxy::ProcessRequest\n");

 if(newSockFd == sendUDPSocket)
  {
   /* UDP socket connection */
   char *toolName=NULL;

   if(isdigit(inData[0]))
 {
  reqString = &inData[4];
  queryString = NULL;
  inData[3] = '\0';
  operation = atoi(inData);

  if(operation == 40) /* PROXY_YP_SERVER=40 - ProxyGlobals.h */
   toolName = "/HTTP/YP_Server";
 }
   else
{
 /* This might be an old stack which send the message that we're going
   ot use for the CMIP discovery. We assume that it will be able to
   handle the message */

 toolName = "/HTTP/YP_Server";
}
```

```
      if(toolName == NULL)
{
  sprintf(tmpStr, "Received unknown UDP message [%d-%s]", operation, reqString);
  LogMsg(tmpStr);
  DecrementNumComponents();
  return;
}
      else
toolInfo = GetDroplet(toolName); /* Get the requested tool */

      if(toolInfo == NULL)
{
  DecrementNumComponents();
  return;
}
    }
  else /* HTTP Request */
    {
      int numBytes=0, rc, len, i, numSlash=0;
      char lastChar, *name, logEntry[96];

      httpStringRepository = new char[4096];

      while(numBytes < 4095)
{
  rc = recv(newSockFd, &httpStringRepository[numBytes], 1, 0);

  if((rc < 0) || (httpStringRepository[numBytes] == '\n'))
      break;

  numBytes++;
}

      //printf("--> HTTP Request [1] (%d)\n", numBytes);

      if((rc == -1) || (numBytes < 4)
          || (strncmp(httpStringRepository, "GET ", 4) != 0))
             /* Client disconnected or wrong msg */
{
  httpStringRepository[numBytes] = '\0'; /* Just to be safe */

  if(rc == -1)
    {
#ifdef PRINT_INFORMATIVE_MESSAGES
      _HTTPproxy->LogMsg("Client disconnected while sending request");
      /* No error message is sent since the client disconnected already */
#else
      ; /* Nothing to do */
#endif
    }
  else
    {
#ifdef PRINT_INFORMATIVE_MESSAGES
      char s[96];

      if(numBytes > 40)
{
  httpStringRepository[38] = '.';
  httpStringRepository[39] = '.';
  httpStringRepository[40] = '\0';
}

     sprintf(s, "Unknown HTTP req. '%s' (client abort ?)", httpStringRepository);
      _HTTPproxy->LogMsg(s);
#endif
      SendString(newSockFd, "HTTP/1.0 500 Unknown request (client abort?)\n"); /
* Send error response */
    }

  delete httpStringRepository;
  CloseSocket(newSockFd);
  DecrementNumComponents();
  return;
}

memset(&httpStringRepository[numBytes+1], '\0', 8); /* Make sure there's no gar-
bage at the end */
//printf("--> HTTP Request [2]\n");

if(getpeername(newSockFd, (sockaddr*)&peerName, &peerNameLen) == 0)
{
#ifdef ENABLE_LOG
    struct hostent *peerAddr;
  char* addr;
  int len;

      sprintf(logEntry, "%ld", peerName.sin_addr.s_addr);

  addr = (char*)hostAddresses->RetrieveEntry(logEntry);

  if(addr == NULL)
  {
  peerAddr = gethostbyaddr((char*)&peerName.sin_addr.s_addr, 4, AF_INET);
 if(peerAddr != NULL)
 {
   hostAddresses->AddEntry(_strdup(logEntry), _strdup(peerAddr->h_name));
   sprintf(logEntry, "[%s] ", peerAddr->h_name);
 }
 else
   logEntry[0] = '\0';
  }
  else
     sprintf(logEntry, "[%s] ", addr);
#else
  logEntry[0] = '\0';
#endif

  len = strlen(httpStringRepository);
  if((len>=11) && (strncmp(&httpStringRepository[len-11], " HTTP/1.0", 9) == 0))
    {
      httpStringRepository[numBytes-9] = '\0';
      if(len>30)
      {
  char c = httpStringRepository[25];
  httpStringRepository[25] = '\0';
  strcat(logEntry, &httpStringRepository[4]);
  httpStringRepository[25] = c;
  strcat(logEntry, "...");
}
else
  strcat(logEntry, &httpStringRepository[4]);

      httpStringRepository[numBytes-9] = '\0';
      strcat(httpStringRepository, "HTTP/1.0\r\n");
      memset(&httpStringRepository[numBytes+1], '\0', 8); /* Make sure there's
             no garbage at the end */
    }
  else
```

```
    {
      if(len>30)
      {
       char c = httpStringRepository[25];
       httpStringRepository[25] = '\0';
       strcat(logEntry, &httpStringRepository[4]);
       httpStringRepository[25] = c;
       strcat(logEntry, "...");
    }
    else
     strcat(logEntry, &httpStringRepository[4]);

      httpStringRepository[len-1] = '\0'; /* Remove final '\n' */
    }
}
else
  logEntry[0] = '\0';

   WriteLogEntry(httpLogFile, logEntry); /* Write and entry into the log file */

     /* If the name is of type "/zzz/bbb cccc" then the toolname is "/zzz/bbb" */
      name = &httpStringRepository[4];

      if(name[0] == '/')
{
  len=strlen(name);

  for(i=1; i<len; i++)
    if((name[i] == '/'))
      {
        numSlash++;
       if(numSlash == 2)
        break;
      }
    else if((name[i] == '?') || (name[i] == ' '))
      {
       numSlash = 2;
       break;
      }

  if(numSlash == 2)
    {
      lastChar = name[i];
      name[i] = '\0';
    }
  }
  else
{
  i = 5;
  lastChar = name[i];
}

#ifndef LEAKS
      toolInfo = toolMgr.GetToolByName(name); /* Get the requested tool */

      if((toolInfo == NULL)
 || (toolInfo->jumpInfo->toolProc == NULL)
 || (toolMgr.LockTool(toolInfo) == NULL))
{
  FILE *fd;
  int idx, insIdx, stop;

  name[i] = lastChar; /* Restore */
  strcpy(tmpStr, localDirString); /* It specifies the local directory */
 for(idx=1, insIdx=strlen(tmpStr), stop=0; stop == 0; idx++, insIdx++)
  {
   switch(httpStringRepository[idx+4])
{
case ' ':
case '\r':
case '\n':
 stop=1;
 tmpStr[insIdx] = '\0';
 break;
default:
 tmpStr[insIdx] = httpStringRepository[idx+4]; /* +4 discards 'GET ' */
 break;
 }
}

 // printf("File = '%s'\n", tmpStr);
 if(fileDividerChar != '/')
 {
  int dividerIdx;

  for(dividerIdx=0; tmpStr[dividerIdx] != '\0'; dividerIdx++)
   if(tmpStr[dividerIdx] == '/')
    tmpStr[dividerIdx] = fileDividerChar;
 }

 fd = fopen(tmpStr, "rb");

 if(fd != NULL)
  {
   if(strncmp(&httpStringRepository[idx+4], "HTTP/1.0", 8) == 0)
   {
    time_t theTime = time(NULL);
    char tmpBuffer[96];
    stat(tmpStr, &statInfo);

    ReadGarbage(newSockFd, password);
    SendString(newSockFd, "HTTP/1.0 200 Document follows\n");

    /* Thu, 25 Jul 1996 13:47:57 GMT */
   strftime(tmpBuffer, 95, "Date: %a, %d %b %Y %H:%M:%S %Z\n",
           localtime(&theTime));
 SendString(newSockFd, tmpBuffer);

 SendString(newSockFd, "Server: IBM Liaison\n");

 if(strncmp(&tmpStr[strlen(tmpStr)-4], ".gif", 4) == 0)
  SendString(newSockFd, "Content-type: image/gif\n");

 strftime(tmpBuffer, 95, "Last-modified: %a, %d %b %Y %H:%M:%S %Z\n",
  localtime(&statInfo.st_mtime));
 SendString(newSockFd, tmpBuffer);
 sprintf(tmpBuffer, "Content-length: %d\n\n", statInfo.st_size);
 SendString(newSockFd, tmpBuffer);
}
else
{
 //ReadGarbage(newSockFd, password);
 SendString(newSockFd, "HTTP/1.0 200 Document follows\n\n");
}

for(;;)
{
 numBytes = fread(tmpStr, sizeof(char), 255, fd);
 if(numBytes <= 0) break;
 rc = sendstring(newSockFd, tmpStr, numBytes);
 if(rc != numBytes)
   break;
}
   fclose(fd);
   }
 else
   {
    /* Tool NOT found: consult the HTTP daemon, if any, otherwise
       return an error */
    struct sockaddr_in addr;
    int err, len, listenSocket;

    listenSocket = socket(AF_INET, SOCK_STREAM, 0);
    bzero((char*)&addr, sizeof(addr));
    addr.sin_family      = AF_INET;
    addr.sin_addr.s_addr = INADDR_ANY;
    addr.sin_port        = htons(httpDaemonPort);
    err = connect(listenSocket, (struct sockaddr *)&addr, sizeof(addr));
    if(err != 0)
{
 SendString(newSockFd, "<HEAD><TITLE>404 Not Found</TITLE></HEAD>");
 SendString(newSockFd, "<BODY><H1>404 Not Found</H1>");
  SendString(newSockFd, "The requested URL can not be handled by this serv-
er.<P><hr>");
 sprintf(httpStringRepository, "<strong>IBM %s v.%s [%s personality]</strong>",
 applName, shortVersStr, personalityName);
 SendString(newSockFd, httpStringRepository);
 SendString(newSockFd, "<p>Copyright IBM 1995-96, All Right Reserved.\n");
 SendString(newSockFd, "</BODY>");
}
    else
{
 int contentLength=0;

 httpStringRepository[numBytes-1] = '\r';
 httpStringRepository[numBytes] = '\n';
 httpStringRepository[numBytes+1] = '\0';
 /* _HTTPproxy->LogMsg(httpStringRepository); */

 /* printf("Requested: %s\n", httpStringRepository); */
 send(listenSocket, httpStringRepository, strlen(httpStringRepository), 0);

#define ACCEPT_HTTP "Accept: image/gif, image/x-xbitmap, image/jpeg, image/
pjpeg, */*\r\n\r\n"
 send(listenSocket, ACCEPT_HTTP, strlen(ACCEPT_HTTP), 0);

 FD_ZERO (&mask);
 FD_SET (listenSocket, &mask);
 wait_time.tv_sec = 60, wait_time.tv_usec = 0;
 if (select(listenSocket+1, &mask, 0, 0, &wait_time) == 1)
   {
    for(len=1 /* Dummy value != 0 */;len > 0;)
{
 len = readline(listenSocket, httpStringRepository, 4096);

 /* printf("%d\n", len); */

 if(len == 0)
   {
    int bytesToRead;
    sendstring(newSockFd, "\n", 1);

    /* Now the document comes */
    while(contentLength > 0)
 {
 if(contentLength > 4096)
   bytesToRead = 4096;
 else
   bytesToRead = contentLength;

 len = recv(listenSocket, httpStringRepository, bytesToRead, 0);
 contentLength -= len;

 rc = sendstring(newSockFd, httpStringRepository, len);
 if(rc < 0) break; /* Partner disconnected ? */
 /* printf("- %d\n", len); */
 }
  break; /* End of the for loop */
 }
 else if(len > 0)
   {
    if(strncmp(httpStringRepository, "Content-length", 14) == 0)
{
 httpStringRepository[len] = '\0';
 contentLength = atoi(&httpStringRepository[16]);
 httpStringRepository[len] = '\n';
}

    len++;
    rc = sendstring(newSockFd, httpStringRepository, len);
    if(rc < 0) break; /* Partner disconnected ? */
   }
}
   }
 else
   {
    SendString(newSockFd, "<HEAD><TITLE>404 Not Found</TITLE></HEAD>\n");
    SendString(newSockFd, "<BODY><H1>404 Not Found</H1>\n");
     SendString(newSockFd, "The requested URL can not be handled by standard
HTTP server:\n"
 " [reason: timeout].<P><hr>\n");
     SendString(newSockFd, "<strong>IBM ZRL Proxy Server</strong>. Copyright IBM
1995-96, All Right Reserved.\n");
    SendString(newSockFd, "</BODY>\n");
   }
}

    CloseSocket(listenSocket);
   }

 CloseSocket(newSockFd);
 delete httpStringRepository;
 DecrementNumComponents();
 return;
}
    else
#endif
{
 int boundary=-1, j, readGarbage=0;
 char *tmpPtr = &name[i+1];

 len = strlen(tmpPtr);
 if((len>=11) && (strncmp(&tmpPtr[len-11], " HTTP/1.0", 9) == 0))
   {
    memset(&tmpPtr[len-11], '\0', 11); /* Make sure there's not garbage at the
end */
    readGarbage=1;
   }

 switch(lastChar)
   {
   case '?':
    reqString = "";
    queryString = &name[i+1];
    break;
   case ' ':
    reqString   = "";
    queryString = "";
    break;
   case '/':
    reqString = &name[i+1];

    for(i=0, j=strlen(tmpPtr);i<j; i++)
if((tmpPtr[i] == ' ') || (tmpPtr[i] == '\r'))
 break;
else
 if(tmpPtr[i] == '?')
   {
    tmpPtr[i] = '\0';
    boundary = i;
   }

    if(boundary != -1)
queryString = &tmpPtr[boundary+1];
    else
queryString = "";
    break;
   }

 /* Receive data on the socket */
 if(readGarbage)
   ReadGarbage(newSockFd, password);
}
   }

#ifndef LEAKS
 /* Verify authorization */
 if(ClientAuthorized(password, toolInfo->toolName, reqString))
   toolInfo->jumpInfo->toolProc(newSockFd, reqString, queryString,
 HTTP_LISTEN_PORT /* Port used by the remote HTTPD */);
 else
   {
     SendString(newSockFd, "HTTP/1.0 401 Unauthorized to access the document\n");
    SendString(newSockFd, "Server: IBM Liaison\n");
    SendString(newSockFd, "Content-Type: text/html\n\n");
             SendString(newSockFd,   "<HTML>\n<HEAD>\n<TITLE>Error</TITLE>\n</
HEAD>\n<BODY>\n"
 "<H1>Error 401</H1>\nUnauthorized to access the document\n</BODY>\n</HTML>\n");
   }

 toolMgr.UnlockTool(toolInfo); /* Used to decrement the usage counter */
#else
 ExternFunction(newSockFd, reqString, queryString, HTTP_LISTEN_PORT /* Port used
by the remote HTTPD */);
#endif

 if(newSockFd != sendUDPSocket)
   {
    delete httpStringRepository; /* Delete now: after the processing */
    CloseSocket(newSockFd);
   }
 else
   {
    if(reqString != NULL)   delete reqString;
    if(queryString != NULL) delete queryString;
   }

 DecrementNumComponents();

#undef PAD
}

Proxy* GetProxy() {
 return(_HTTPproxy);
}
```

Linux_Personality.cpp

```
#define _PROXY_TOOLS_C_ /* Do not remove it */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#define __STR31__
#include <string.h>
#include <signal.h>
#include <dlfcn.h> /* dlopen... */

#ifndef STHREAD
extern "C" {
#include <pthread.h>
}
#endif

#include "Proxy.h"
#include "ProxyTools.h"

#define VERBOSE

static JumpInfo* (*jumpProc)();

char applName[]         = "Liaison";
char personalityName[]  = "Linux (x86)";

char fileDividerChar  = '/';
char localDirString[] = "./"; /* It specifies the local directory */

extern "C" {
  void Terminate(int signalId);
  void IgnoreSignal(int signalId);
} // Prototype

class IdxEntry {
public:
  char* name;
  ino_t fileModDate;
  int   loadOnDemand;
  char* locServices;
  char* remServices;

  IdxEntry();
  ~IdxEntry();
```

```
};

IdxEntry::IdxEntry() {
  name = NULL;
  locServices = NULL;
  remServices = NULL;
}

IdxEntry::~IdxEntry() {
  if(name != NULL)    delete name;
  if(locServices != NULL) delete locServices;
  if(remServices != NULL) delete remServices;
}

extern "C" {
  void Terminate(int signalId)
    {
      char tmpStr[64];
      Proxy *HTTPproxy = GetProxy();

      sprintf(tmpStr, "Caught signal %d. Shutting down Proxy...", signalId);
      HTTPproxy->LogMsg(tmpStr);
      delete HTTPproxy;
      exit(0);
    }

  void IgnoreSignal(int signalId) {
    // printf("IgnoreSignal: %d\n", signalId);
    signal(signalId, IgnoreSignal);
  }

}

Personality::Personality()
{
  /* freopen("/dev/null", "r", stderr); */
  SetThreadAttributes(); /* For the main thread */
}

Personality::~Personality()
{
  ;
}

unsigned long Personality::GetDropletModificationDate(char* dllName)
{
  char path[96];
  struct stat statInfo;

  sprintf(path, "Droplets%c%s", fileDividerChar, dllName);

  if(stat(path, &statInfo) != 0)
    return(0); /* error */
  else
    return(statInfo.st_mtime);
}


int Personality::CreateToolIdx(char* dir)
{
  int numElems=0, createNewIdx=0;
  DIR* directoryPointer;
  struct dirent* dp;
  char tmpStr[512], tmpFile[255], indexPath[32], dropletName[48];
  time_t indexModificationTime;
  Proxy* HTTPproxy = GetProxy();

  indexModificationTime = GetDropletModificationDate("index");

  if(indexModificationTime > 0 /* The index file exists */)
    {
      sprintf(tmpStr, "%s/", dir);
      directoryPointer = opendir(tmpStr);

      if(directoryPointer == NULL)
return(-1);

      while((dp=readdir(directoryPointer)) != NULL)
{
  strcpy(tmpStr, dp->d_name);

  if((tmpStr[0] != '.') /* It could either be "." or ".." */
     && (strcmp(&tmpStr[strlen(tmpStr)-4], ".dll") == 0))
    {
      numElems++;

      if(GetDropletModificationDate(tmpStr) > indexModificationTime)
{
  createNewIdx = 1;
  break;
}
    }
}

      if(numElems == 0) createNewIdx = 1; /* Empty directory */
      closedir(directoryPointer);
    }
  else
    createNewIdx = 1;

  if(createNewIdx)
    {
      FILE *fd;
      void *moduleHdl, *myProc;
      JumpInfo* tmpJmp;
      int i;
      HashTable *ht;
      IdxEntry *idxEntry;

      /*** [1] Read the old index file first ***/
      ht = new HashTable(64, FREE_DATA_AND_KEY_MEMORY);
      sprintf(tmpStr, "Droplets%cindex", fileDividerChar);

      if((fd = fopen(tmpStr, "r")) != NULL)
{
  do
    fgets(tmpStr, 512, fd); /* Skip Comment */
  while(tmpStr[0] == '#');

  while(fgets(tmpStr, 512, fd))
    {
      idxEntry = new IdxEntry;

      tmpStr[strlen(tmpStr)-1] = '\0'; /* Remove final '\n' */
      strcpy(dropletName, tmpStr);

      fgets(tmpStr, 512, fd);
      tmpStr[strlen(tmpStr)-1] = '\0'; /* Remove final '\n' */
      idxEntry->name = strdup(tmpStr);

      fgets(tmpStr, 512, fd);
      sscanf(tmpStr, "%ul", &idxEntry->fileModDate);

      fgets(tmpStr, 512, fd);
      sscanf(tmpStr, "%d", &idxEntry->loadOnDemand);

      fgets(tmpStr, 512, fd);
      idxEntry->locServices = strdup(tmpStr);

      fgets(tmpStr, 512, fd);
      idxEntry->remServices = strdup(tmpStr);

      ht->AddEntry(strdup(dropletName), idxEntry);
    }

  fclose(fd);
}

      /*** [2] Scan the directory ***/
      time_t tp = time(NULL);

#ifdef VERBOSE
      HTTPproxy->LogMsg("- Creating new droplet index file");
#endif

      sprintf(tmpStr, "%s/", dir);
      directoryPointer = opendir(tmpStr);

      sprintf(indexPath, "Droplets%cindex", fileDividerChar);
      fd = fopen(indexPath, "w+");

      if((directoryPointer == NULL) || (fd == NULL))
{
  delete ht;
  return(-1);
}

      fprintf(fd, "###########################################\n");
      fprintf(fd, "# Droplet index file                       \n");
      fprintf(fd, "# ---------------------------------------- \n");
      fprintf(fd, "# Date: %s", ctime(&tp));
      fprintf(fd, "# Format:                                  \n");
      fprintf(fd, "# dropletFileName[str]   <cr>              \n");
      fprintf(fd, "# dropletName[str]       <cr>              \n");
      fprintf(fd, "# fileId[ulong]          <cr>              \n");
      fprintf(fd, "# loadOnDemand[0/1]      <cr>              \n");
      fprintf(fd, "# localServices[str*]    <cr>              \n");
      fprintf(fd, "# remoteServices[str*]   <cr>              \n");
      fprintf(fd, "#                                          \n");
      fprintf(fd, "###########################################\n");
      fprintf(fd, "\n"); /* Do not forget this */

      while((dp=readdir(directoryPointer)) != NULL)
{
  strcpy(tmpStr, dp->d_name);

  if((tmpStr[0] != '.') /* It could either be "." or ".." */
     && (strcmp(&tmpStr[strlen(tmpStr)-4], ".dll") == 0))
    {
      idxEntry = (IdxEntry*)ht->RetrieveEntry(tmpStr);
     if((idxEntry != NULL) && (idxEntry->fileModDate == GetDropletModificationDate(tmpStr)))
```

```
{
  /* The file is not modified hence it doesn't have to be loaded */

  fprintf(fd, "%s\n", tmpStr);                   /* dropletFileName */
  fprintf(fd, "%s\n", idxEntry->name);           /* dropletName     */
  fprintf(fd, "%lu\n", idxEntry->fileModDate);   /* modif. time     */
  fprintf(fd, "%d\n", idxEntry->loadOnDemand);   /* loadOnDemand    */
  fprintf(fd, "%s", idxEntry->locServices);      /* local services  */
  fprintf(fd, "%s", idxEntry->remServices);      /* remote services */
}
      else
{
  if((idxEntry != NULL) && (idxEntry->name != NULL))
   HTTPproxy->UnloadToolByName(idxEntry->name, 0); /* IMPORTANT: Unload the tool
first */

  if((LoadModule(tmpStr, &moduleHdl) == 0) /* All right */
     && (GetModuleEntryPoint(moduleHdl, &myProc) == 0))
    {
      jumpProc = (JumpInfo*(*)())myProc;
      tmpJmp = jumpProc();

      if(tmpJmp->version == TOOL_VERSION)
{
  fprintf(fd, "%s\n", tmpStr);            /* dropletFileName  */
  fprintf(fd, "%s\n", tmpJmp->toolName);  /* dropletName      */
  fprintf(fd, "%lu\n", GetDropletModificationDate(tmpStr));/* modification Time
*/

  if(tmpJmp->loadOnDemand == DONT_LOAD_ON_DEMAND)
    fprintf(fd, "1\n");
  else
    fprintf(fd, "0\n");

  if(tmpJmp->services)
    for(i=0; tmpJmp->services[i].servName != NULL; i++)
      {
if(i == 0)
  fprintf(fd, "%s", tmpJmp->services[i].servName);
else
  fprintf(fd, " %s", tmpJmp->services[i].servName);
      }
  fprintf(fd, "\n"); /* End of list of local services */

  if(tmpJmp->remServices)
    for(i=0; tmpJmp->remServices[i].servName != NULL; i++)
      {
if(i == 0)
  fprintf(fd, "%s", tmpJmp->remServices[i].servName);
else
  fprintf(fd, " %s", tmpJmp->remServices[i].servName);
      }
  fprintf(fd, "\n"); /* End of list of local services */
}
      else
{
  sprintf(tmpStr, "Unable to load '%s': bad version [%0.3hd]",
  tmpJmp->toolName, tmpJmp->version);
  HTTPproxy->LogMsg(tmpStr);
}

      UnloadModule(moduleHdl);
    }
}
```

```
  }
}

   fclose(fd);
   closedir(directoryPointer);
   delete ht;
#ifdef VERBOSE
   HTTPproxy->LogMsg("- index file created successfully");
#endif
   return(2); /* Newly created file */
  }
 else
  return(0); /* Nothing has been created */
}


int Personality::LoadModule(char* dllName, void** dllHandle)
{
 char *poldlibpath;
 int result;
 char tmpStr[64];
 Proxy* HTTPproxy = GetProxy();

 dlerror(); /* Reset error msg */

 sprintf(tmpStr, "%s", dllName);
 (*dllHandle) = dlopen(tmpStr, RTLD_GLOBAL); /* Load the library */

#ifdef VERBOSE
  sprintf(tmpStr, "Load [%p] %s", (*dllHandle), dllName);
  HTTPproxy->LogMsg(tmpStr);
#endif

 if((*dllHandle) == NULL)
  {
   char errMsg[128];

   sprintf(errMsg, "Unable to load droplet '%s' [%s]", tmpStr, dlerror());
   HTTPproxy->LogMsg(errMsg);

   result = 1;
  }
 else
  result = 0;

 return (result);
}

int Personality::UnloadModule(void* dll_handle)
{
 int rc;
 Proxy* HTTPproxy = GetProxy();
 char tmpStr[64];

 errno = 0;
 rc = dlclose(dll_handle);
 if(rc != 0)
  {
   sprintf(tmpStr, "Unload returned %d [errno=%d] %p", rc, errno, dll_handle);
   HTTPproxy->LogMsg(tmpStr);
  }
#ifdef VERBOSE
 else
  {
      sprintf(tmpStr, "Unload OK [%p]", dll_handle);
      HTTPproxy->LogMsg(tmpStr);
    }
#endif

  return(rc);
}

int Personality::GetModuleEntryPoint(void* dll_handle, void** pproc_addr)
{

  (*pproc_addr) = dlsym(dll_handle, "JumpProc__Fv");

  if(((char*)dlerror()) != 0)
    return(-1);
  else
    return(0);
}

void Personality::CreateThread(THREAD_RET_TYPE(*FuncPtr)(void *),
       void* thrArgs, long* _threadId)
{
  int rc=0;

#ifndef STHREAD
  static pthread_t sharedId;
  pthread_t *threadId;

  if(_threadId != NULL)
    {
      threadId = new pthread_t;
      (*_threadId) = (long)threadId;
    }
  else
    threadId = &sharedId;

  rc = pthread_create(threadId, NULL, FuncPtr, thrArgs);

#else /* STHREAD */
  FuncPtr(thrArgs);
#endif
}

void Personality::KillThread(long threadId) {
#ifndef STHREAD

  if(threadId == 0)
    return; /* Nothing to cancel */

  pthread_t *_threadId = (pthread_t*)threadId;

  int rc = pthread_kill(*_threadId, 15 /* SIGTERM */);

  delete _threadId;
#endif

}

void Personality::SetThreadAttributes()
{
  static short i=0; /* Under 4.x the signals will be set just once */

  if(i == 0)
  {
    signal(SIGCLD,  IgnoreSignal);   /* Avoid zombie processes            */
    signal(SIGPIPE, IgnoreSignal);   /* Don't die when a client is killed */
    //sigignore(SIGPIPE);
    signal(SIGTERM, Terminate);
    signal(SIGINT,  Terminate);
    signal(SIGQUIT, Terminate);
    i=1;
  }
}

void Personality::CancelCurrentThread() {
#ifndef STHREAD
  pthread_exit(NULL);
#endif
  printf("Thread canceled");
}

void Personality::CleanUpThread() {
  /* This function cleans up any memory that the thread allocated
     during its execution */
#ifndef STHREAD
  pthread_exit(NULL);
#endif
}

extern "C" {
  int soclose(int fd) {
    return(close(fd));
  }
}

/*******************************************************/

#ifndef STHREAD

MutexSemaphore::MutexSemaphore() {
  int rc;

  if ((mutex_sem = (void *) new pthread_mutex_t) == NULL) {
    GetProxy()->LogMsg("MutexSemaphore::MutexSemaphore: new failed");
    return;
  }

  rc = pthread_mutex_init((pthread_mutex_t *) mutex_sem, NULL);
}

MutexSemaphore::~MutexSemaphore() {
  pthread_mutex_unlock((pthread_mutex_t *) mutex_sem);
  pthread_mutex_destroy((pthread_mutex_t *) mutex_sem);
  delete (pthread_mutex_t *) mutex_sem;
}


int MutexSemaphore::access() {
  int rc = pthread_mutex_lock((pthread_mutex_t *) mutex_sem);
  if (rc)
    return(NOTOK);
  else
    return(OK);
}

int MutexSemaphore::release() {
  int rc = pthread_mutex_unlock((pthread_mutex_t *) mutex_sem);
  if (rc)
    return(NOTOK);
  else
    return(OK);
}

/*******************************************************/

CondVariable::CondVariable() {
  int rc;

  pthread_mutex_t  *_mutex;
  pthread_cond_t   *_condvar;

  _mutex = new pthread_mutex_t;
  _condvar = new pthread_cond_t;

  mutex = (void*)_mutex;
  condvar = (void*)_condvar;

  pred = 0;

  rc = pthread_mutex_init((pthread_mutex_t*)mutex, NULL);
  rc = pthread_cond_init((pthread_cond_t*)condvar, NULL);
}


CondVariable::~CondVariable() {
  pthread_mutex_destroy((pthread_mutex_t*)mutex);
  pthread_cond_destroy((pthread_cond_t*)condvar);
}

int CondVariable::Wait() {
  int rc;

  if((rc = pthread_mutex_lock((pthread_mutex_t*)mutex)) != 0)   /* lock mutex */
    return rc;

  pred = 1; /* set predicate to 1 */

  // if pred is 1: wait in a loop until it is 0
  // if pred is 0: proceed
  while(pred != 0)
   rc = pthread_cond_wait((pthread_cond_t*)condvar, (pthread_mutex_t*)mutex);

  rc = pthread_mutex_unlock((pthread_mutex_t*)mutex);   /* unlock mutex */
  return rc;
}


int CondVariable::Signal() {
  /* lock mutex */
  int rc = pthread_mutex_lock((pthread_mutex_t*)mutex);
  if(rc)
    {
      printf("CondVariable::Signal [1]: rc=%d\n", rc);
      return rc;
    }

  /* set pred to 0 */
  pred = 0;

  rc = pthread_cond_signal((pthread_cond_t*)condvar);
  if(rc) printf("CondVariable::Signal [2]: rc=%d\n", rc);

  /* unlock mutex */
  rc = pthread_mutex_unlock((pthread_mutex_t*)mutex);
  if(rc) printf("CondVariable::Signal [3]: rc=%d\n", rc);

  return rc;
}

#endif

void DoSleep(int secs) {
  sleep(secs);
}
```

Curriculum Vitæ

Surname:	Deri
First Name:	Luca
Date of Birth:	September 24th, 1968, Pisa, Italy.
E-mail:	lde@zurich.ibm.com, deri@iam.unibe.ch.
WWW:	http://www.zurich.ibm.com/~lde/ http://iamwww.unibe.ch/~deri/

Education

Nov. 1988 - Apr. 1992	University of Pisa, department of Computer Science: master degree on Computer Science.
1982 - 1988	Scientific Grammar School "U.Dini" in Pisa.

Work Experience

Since August 1993	Guest Scientist at the IBM Zurich Research Laboratory as member of the Network Management Group, Zurich, Switzerland.
Jan. - Aug. 1993	Research Fellow at the UCL (University College of London), London, United Kingdom.
June 1991 - Dec. 1992	Tecsiel S.p.A. an IRI-STET Company, Pisa, Italy.

Publications

1. L. Deri, *Yasmin: a Component-based Architecture for Software Applications*, IBM Research Report RZ 2899, Proceedings of STEP ‘97, London, July 1997.
2. L. Deri, *Rapid Network Management Application Development*, Proceedings of ECOOP '97 Workshop on Object Oriented Technology for Telecommunications Services Engineering, Jyväskylä, Finland, June 1997.
3. L. Deri and D. Manikis, *VRML: Adding 3D to Network Management*, Proceedings of IS&N ‘97, Como, Italy, May 1997.

4. L. Deri and B. Ban, *Static vs. Dynamic CMIP/SNMP Network Management Using CORBA*, Proceedings of IS&N ‘97, Como, Italy, May 1997.
5. F. Barillaud, L. Deri and M. Feridun, *Network Management using Internet Technologies*, Proceedings of INM ‘97, San Diego, May 1997.
6. J. Reilly, P. Niska, L. Deri and D. Gantenbein, *Enabling Mobile Network Managers*, Proceedings of the 6th Int. WWW Conference, Santa Clara, CA, April 1997.
7. L. Deri, *HTTP-based CMIP/SNMP Management*, Internet Draft, November 1996.
8. L. Deri, *Network Management for the 90*s, Proceeding of ECOOP '96 Workshop on Systems and Network Management, Linz, Austria, July 1996.
9. L. Deri, *Surfin' Network Management Resources Across the Web*, Proceedings of 2nd Int. IEEE Workshop on Systems and Network Management, Toronto, June 1996.
10. L. Deri and B. Ban, *Java Dynamic Class Loader*, IBM Research Report, December 1995.
11. B. Ban and L. Deri, *Object Factory Revised: a Design Pattern*, IBM Research Report, September 1995.
12. L. Deri, *Droplets: Breaking Monolithic Applications Apart*, IBM Research Report, September 1995.
13. L. Deri and A. Weder, *Webbin' CMIP*, Poster Proceedings of 3rd Int. WWW Conference, Darmstadt, Germany, April 1995.
14. L. Deri and E. Mattei, *An Object-Oriented Approach to the Implementation of OSI Management*, Computer Networks and ISDN Systems, Vol. 27, 1995.
15. S. Bhatti, L. Deri and G. Knight, *Secure Remote Management in the ESPRIT MIDAS Project*, Proceedings of ULPAA'94 (Upper Layer Protocols Architectures and Applications), Barcelona, Spain, June 1994.
16. L. Deri and P. Artico, *System and Network Management*, Proceedings of AICA ‘92, Genova, Italy, July 1992.
17. L. Deri, *Basi di Dati*, Booklet about database systems published at the university press of the University of Pisa, 1988.

Invited Talks

1. *Systems and Network Management*, Seminar held at Scuola Normale Superiore, Pisa, Italy, December 19-20, 1996.
2. *Modern Network Management*, CHOOSE-SI Meeting, Bern, December 12, 1996.
3. *Breaking Network Management Complexity Apart*, Rapid ‘96, Bern, September 1996.
4. *Surfin' Network Resources Across the Web*, Telecom PTT, Bern, February 1996.