Consensual Trends for Optimizing the Constitution of Middleware

Stéphane Spahni[*], Jean-Raoul Scherrer[*],
Dominique Sauquet[†], Pier-Angelo Sottile[‡]

Abstract: Middleware is now a commonly used expression and anyone building distributed applications is referring to “middleware services”. Nevertheless this notion lacks of sound theoretical foundation. This paper tries to clarify the relationship between the components of the distributed environment, and establishes some classification aiming at gaining a common understanding of the functionality and interdependency of the existing modules of distributed environments.

Keywords: Middleware, Hospital Information System, OSI

[*] Division d’Informatique Médicale, Hôpital Cantonal Universitaire de Genève, rue Micheli-du-Crest 24, 1211 Genève 4, Switzerland. E-mail: stephane.spahni@dim.hcuge.ch
[†] Hôpital Broussais, 96, rue Didot, 75014 Paris, France. E-mail: sauquet@hbroussais.fr
[‡] GESI / Gestione Sistemi per l’Informatica SRL, Via Rodi 32, 00195 Roma, Italy. E-mail: pas@gesi.it

1 Introduction

As stated by P. A. Bernstein in ACM communication [1], the computing facilities of large enterprises are evolving into a utility. Each application terminal is a desktop appliance that connects to the utility. It can be a PC terminal, or a Workstation. Basically the utility itself is an enterprise-wide network of information services including applications and databases.

Servers on the local area network support files and file-based applications such as electronic mail, board minutes, document preparation, and printing. Servers typically support access to DBS (Database System) or transaction processing applications. Some servers are gateways to services and can also offer electronic document interchange between institutions or equivalent.

Today's hospitals or “healthcare organizations”, like most large institutions, have a wide variety of heterogeneous systems running on different operating systems (OS) and relying on different network architectures. This of course makes integration difficult. Moreover an application available on one local area server may not always be available on other servers. These are only some of the limitations of such systems.

One solution would be to support standard programming interfaces to a variety of server types. Indeed, today's applications may use databases, communications, presentations and other services. Hence, it is now possible to deal not only with widely supported tools but also with standard tools.

Another way vendors solve heterogeneity problems is by supporting standard protocols. An inter-operation occur when a program of a given system is able to access data and/or programs of another systems. A necessary condition for making inter-operation possible is that the two systems are using the same protocol (i.e. same message formats and message sequences).

Besides, the systems are supporting the protocols. In large healthcare enterprises, application developers also have to deal with heterogeneity. “Middleware services” help solve users heterogeneity and distribution problems. By “middleware services” we mean information utilities including standard programming interfaces and services. The name “middleware” comes from the fact that they sit above the OS and networking software and below specific applications.

For many new applications, middleware components are becoming more important than the underlying OS and networking services on which the application services formerly depend. As an example, new applications often depend more on a SQL DB (Database Manager) rather than on an OS's record oriented file system and more on an RPC (Remote Procedure Call) mechanism rather than on transport-level messaging.

A middleware service is thus a general purpose service that sits between “platforms” (subsequently called “bitways”) and applications. A “platform” is a set of low level services and processing elements defined by a processor architecture and an OS's API (Application program Interface). Hence a middleware service is defined by the APIs and protocols it supports such as the different implementations of RPC, e.g. “Sun Microsystems RPC”, or “DCE RPC” (DCE-Distributed Computing Environment). Therefore it implements functionality of general interest for complex distributed systems, and in particular for healthcare distributed systems.

However the present situation is unclear! Indeed there are as many different ways of doing things as there are implementations of distributed information systems using varied pathways, e.g. as indicated in Figure 1. Having in mind that Healthcare information system applications should become far more portable from one site to another, it becomes desirable to consensually restrict the commendable pathways to only the most appropriate ones deduced from the critical analysis of the present situation.

2 The concept of the “Middleware” approach

2.1 The design concept

Middleware is quite a fuzzy notion, often used without any reference to a sound theoretical foundation. This is mostly due to the lack of scientific publications in technical journals. However, when building structured applications, distributed ones in particular, it is difficult not to deal with this middleware approach. Although the literature about middleware is relatively scarce [1-3], several common characteristics can be identified as follows:

- The middleware is at the application service level. It implements services for high level applications. The common functionality is grouped into a “layer” called middleware, enabling the developers to concentrate on the functionality specific to the application.
- The middleware is strongly connected to distributed systems. It addresses essentially complex applications spread over many systems, i.e. used in a distributed environment.
- The middleware runs on different platforms. The middleware needs to be available on heterogeneous platforms in order to support independence between the environment and the applications. The most commonly used environment for distributed applications is based on client/server approach, where the servers are typically Unix or Windows NT™ servers and clients PCs, Macintoshes or even Unix workstations in the 3-tier architecture [4].
- The middleware takes advantage of bitways or DBMS services. It relies on lower level bitways services for gaining access to the network, and possibly uses DBMS services for getting access to database systems.

Therefore the middleware aims at reducing the impact of problems related to the development of applications within heterogeneous environments, offering high level standardized services hiding most of this heterogeneity. Indeed the concept itself of middleware has its origin in the development of distributed systems, where the applications run on top of a “cloud of systems” and are therefore clients of good and easy-to-use information exchange facilities. Considering the relation between the middleware and the level of coupling of systems, or the type of parallelism between the components of the distributed system, it can be observed that middleware is primarily related to loosely coupled systems. Massive parallelism does not require middleware, although it can use it: in tightly coupled systems, the communication between the processors is made through low level interfaces (e.g. communication bus, shared memory, etc.), exchanging a lot of data at a very high throughput. In such massively parallel systems, the key words for the relationships between processors are simplicity and efficiency, and not the level of functionality.

On the other hand, the sharing of tasks in a distributed system implies the exchange of generally more complex data and the use of advanced communication and synchronization mechanisms. Therefore there is a real need for having, at the application’s disposal, high level interfaces for implementing the required communication between processes, like those proposed in the upper layers of the Open Systems Interconnection Reference Model (OSI RM).

2.2 How standardized functionality is implemented into the “middleware”

For instance, if portability, interoperability and wide availability keywords are related to standards, then it appears that middleware is quite often subject to standardization. In general, two categories of standards are identified: de facto standards and expressed standards. De facto standards are mainly issued by software companies while expressed standards are issued by standardization bodies like the International organization for standardization (ISO), the American National Standard Institute (ANSI), the International Telecommunications Union (ITU), and others. The main difference between de facto and expressed standards comes from the way they are promoted. Expressed standards are defined by groups of people, and when a consensus is met, the standard is published and implementations are then expected to be achieved! De facto standards are based on already implemented products, widely accepted and with already a certain level of robustness and of performance. When the product becomes widely enough available and used, it then becomes a standard!

Middleware standards can be found in both categories (e.g. the Open Systems Interconnection stack of protocols is built upon expressed standards while the Internet stack of protocols is mostly built upon de facto standards). Most of them are de facto standards as it will be explained in the next chapter. But there is quite a significant amount of work being done for example in the medical field for establishing expressed standards.

3 Trends in the development of middleware since its design stage

Middleware is being developed since several years. Some common characteristics can already be identified for establishing a classification of the various trends in the developments. While most of the emerging classification work is oriented towards the type of communication between the clients and the servers (RPC, messages, using an Object Request Broker - ORB, etc.) [3], we prefer to base our classification on the level of abstraction of the services provided by the middleware, considering this information more relevant to our work than the mechanism itself used for the distribution.

In this context, three classes or generations can already be defined:

First generation: Elementary Middleware (MW level I)

The first generation of middleware dates from the mid-eighties, when distributed systems really became “the” way for replacing ageing mainframes. Basic tools were developed for enabling applications to delegate part of their work to other systems. For example, the “Remote Procedure Call” (RPC) mechanism is a quite rudimentary way of delegating tasks. It implies that the remote machine knows some predefined functions and is able to process requests to these functions with parameter values received from the network, the results being sent back the same way. The problem was raised when dealing with the need for transferring data or objects between a client machine and a host database. A first approach included the use of FTP (File Transfer Program) [5-6] promoted by the US Department of Defense. Then, and almost in parallel, two different pathways were opened based upon the principle of remote access services from a host to a client local database: (1) the NFS line (Network File System of Sun Microsystems [7]) exclusively devoted to files, followed by AFS (Andrew File System from Carnegie Mellon University) not exclusively devoted to files, and finally DCE/DFS (Distributed File System, based onto AFS) part of the DCE package [8]; (2) the Sun Microsystems RPC line, followed by ROSE (Remote Operation Service from the OSI world [9]) - now superseded by the OSI RPC* [10], and DCE RPC.

Several implementations of RPC are currently used. The main ones in the Internet world are Sun Microsystems’ proprietary RPC and OSF’s Distributed Computing Environment RPC (which also is proprietary in some way). In the OSI world there is the OSI RPC, which is based on concepts and services of DCE. It is worthwhile to note that although their respective functionality is very similar - and the names even sometimes identical - these environments are unable to transparently inter-operate with each other, as each defines its own formalism, encoding method (eXternal Data Representation for Sun, Network Data Representation (NDR) for OSI RPC, etc.) and protocol. Nevertheless they all cope with the four characteristics defined previously for belonging to the middle-

* The Remote Operation Service Element - ROSE was initially standardized and specifies the “Remote Operations” application service element (the standard is based on the structuring scheme valid in 1989, and was never updated). Although not defined as such, implementations made it very close to a remote procedure call service (e.g. M. T. Rose and others met a certain level of achievement with it with ISODE [11]).

ware layer. They all are at the application service level, hence facilitating the development of distributed systems. Each being available on many different platforms, they can be considered portable.

Extension to the RPCs rapidly became available in order to govern transparent access to distributed databases. “Transparency” has to be taken here with two separate meanings: either concerning the location of the database when networked databases are being accessed, or concerning the real database manager (e.g. Ingres, Oracle, DB2, etc.), “folklore” into which we are reluctant to enter since it is against the concept of openness! The SQL language is an example of a middleware-level interface enabling this independence against product-specific query languages.

A database-oriented middleware is still an elementary middleware, since accessing a database over the network is more or less at the same abstraction level as sending a SQL request to a remote procedure call server.

Second generation: Middleware as a support of distribution (MW level II)

With the development of more and more complex and varied distributed systems came a growing demand for extended services handling the distribution (dynamic or not) of tasks, for localizing resources over the network, or for managing the sharing of common resources. This resulted in the development of *middleware environments* such as DCE (Distributed Computing Environment), PVM (Parallel Virtual Machine), or TUXEDO®.

Although the complexity and the services of these environments differ with each other, they all offer far more advanced and varied services compared to those offered by the middleware of the first generation. For example, in addition to basic RPC, DCE offers time synchronization between systems, resources sharing and a directory service.

This distinction between level I and level II middleware does not imply a higher level within the hierarchy of protocol stacks, but an enhancement of the extension and complexity as well as of the number of provided services. For instance, RPC is not only a component of DCE but is driven by higher level services from DCE. This is why in Figure 1 RPC belongs to Middleware I while DCE belongs to Middleware II!

The growing use of object-oriented approach and the willingness to exchange real objects between processes were the principal motivation for enhancing this second generation of middleware. This led to expressed standards on object technology (e.g. CORBA, the Common Object Request Broker Architecture [12]) and to object oriented environments like HELIOS [13], Microsoft® OLE (Object Linking and Embedding), ORBIX-TUXEDO® [14] or GemStone [15] to mention only some of the main ones.

The second generation further enhances the portability of applications over different platforms. It also introduces serious enhancements to the first generation by providing standardized services and tools for supporting the distribution of tasks itself (i.e. the developer now has at his disposal integrated frameworks for building distributed applications without having to bother with basic distribution mechanisms and tools). However, the reusability of an application in another environment is still not easy to achieve as specific services depending on the type of organizational environment are still not standardized. Healthcare information environment is a good example of one type of such organizational environments. This concern leads to the latest generation of middleware, tailored to specific domains of activity.

Third generation: Domain-specific Middleware (MW level III)

The middleware has to provide generic services to applications, which does not mean that the applications have to be generic! We can therefore classify under the name “third generation of middleware” what is being developed now: middleware tailored to specific environments. The goal of this type of middleware is to offer generic services in one specific domain of activity, grouping the domain-specific tasks and information common to all applications into the middleware layer. Through the clear identification and isolation of high level but domain-specific tasks, a higher level of portability can be reached and applications can be easily ported from one organization to another.

Developments currently made in the field of healthcare are good instantiations of such middleware services. Several research projects are exploring desired features of Health Information System’s Middleware (HIS middleware) in order to facilitate the exchange of data between hospitals and to make possible the reuse of the same applications at several sites. For example, one can mention the work of the Health Level Seven group (HL-7) - accredited by the ANSI to write the standard of the same name [16], or the SYNAPSES project described afterwards.

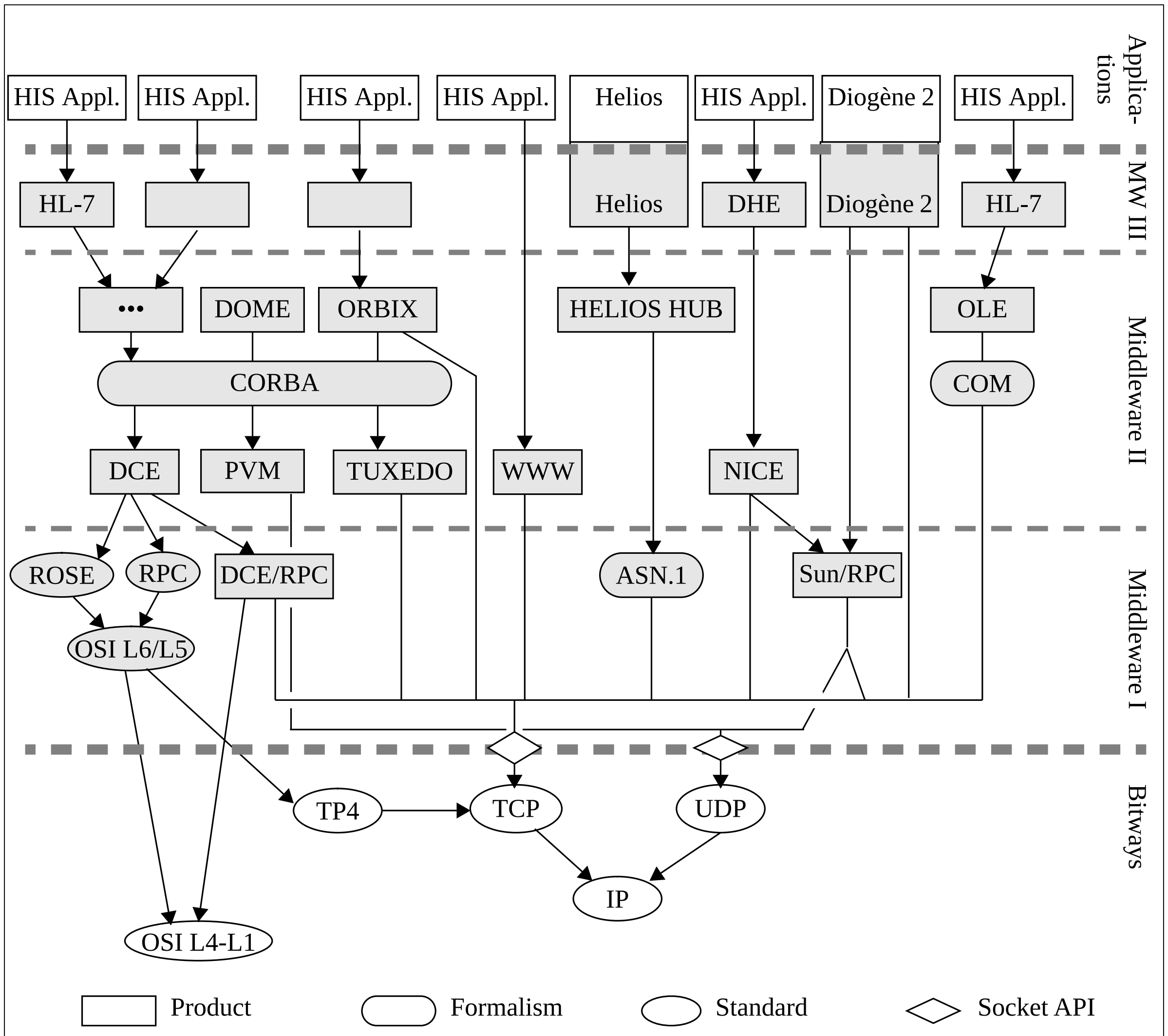
It can be noted that the difference between the second and the third generation can be easily compared to the difference between "objects" and "specialized or business objects": while the former type is generic, the latter is customized to meet the needs of a specific domain of activity.

Considering that one major goal of the middleware is to provide transparent access to information and databases, one well known service has not yet been mentioned in the examples illustrating our classification: the World Wide Web infrastructure. Taking into account the servers infrastructure only (the clients or browsers are just user interfaces), we can identify the following set of basic characteristics:

- The network access to the servers is based upon basic bitways services (TCP/IP).
- There are advanced services like proxies and caches that enhance the performances, hide other protocols (ftp, gopher, etc.) or enable the access across a firewall.
- It is possible to define an "intelligent" behavior at the level of a proxy server in order to filter requests.

These basic characteristics make clear that WWW servers have their place in our classification. Given their advanced services offering far more than a simple access to information on a specific host, we consider the World Wide Web servers as belonging to the second generation of middleware. From the client point of view, there is no need to know exactly where each piece of information is located, or where to find it: it can follow links (URLs - Uniform Resource Locators) without having to realize where they point to. Moreover, thanks to caching servers the information may not come physically from where it thinks it is coming from!

To summarize the discussion, the figure below (Figure 1) presents a refinement of the 3 layers model (bitways, middleware, application) commonly used to describe the tools and protocols for building an application. The middleware layer is divided into 3 sub-layers, reflecting the classification presented above. One can see that most of the (n+1) generation middleware is based on services from the (n) generation. Figure 1 is not intended with the idea of being exhaustive regarding the products listed, but rather with giving a graphical representation of the various abstraction levels that compose the protocol stack of a HIS (Healthcare Information System / Hospital Information System) application.



COM: Component Object Model	IP: Internet Protocol
CORBA: Common Object Request Broker Architecture	NICE: Network Independent Communication Environment
DHE: Distributed Healthcare Environment	OSI L4-L1: OSI transport to physical layers
Diogène 2: Distributed HIS from Geneva's hospitals	TCP: Transport Control Protocol
DOME: Distributed Object Management Environment	TP4: Transport Protocol type 4
HELIOS HUB: Helios Unification Bus	UDP: User Datagram Protocol
HL-7: Health Level Seven	WWW: World Wide Web servers

Figure 1:Classification of some middleware modules

With respect to the classification presented by R.J. Cypser in [17], and without wishing to go into what will be discussed in the following chapter, we can already establish a link between the "Bitways" layer and the Open Systems Interconnection's layers 3c to 4 (or 1 to 4) and between the "Middleware I/II/III" layers and the OSI's layers 5 to 7.

4 Middleware and the OSI model

As presented in the previous chapter, there are many middleware environments, with different levels of functionality. In order to make possible structured descriptions of the applications and the services being used, some formalization of the structure and of the relationships between these environments becomes necessary. To help establishing this structure we refer to the Open Systems Interconnection Reference Model (OSI RM) defined by the International Organization for Standardization (ISO). This model defines 7 hierarchically structured layers, from 1 to 7. The layers can be divided in two groups: the lower layers group, which is responsible for the transmission and which includes layers 1 to 3, and the upper layers group which contains layers 5 to 7 (i.e. session to application layers). The transport layer, or layer 4, provides the necessary glue between the network-oriented lower layers and the application-oriented upper layers. This layer is generally considered as part of the lower layers, the resulting four layers forming the network environment.

It is important to note that despite its name, the "application" layer is related to the *services* for applications, and that it does not contain any application! Applications can be considered as forming the 8th layer, not addressed by the OSI RM.

4.1 The OSI upper layers

Since its initial definition, the OSI reference model went through a certain number of transformations (e.g. [18]). Among the latest are those regarding the *efficiency* of the stack. It has long been recognized that a strict splitting of the layers leads generally to inefficient implementations. Moreover, structural problems were identified, and solutions like merging layers 6 and 7 were proposed (e.g. Ph.D. thesis of S. Spahni [19]). It is therefore now clearly recommended not to implement the layers 5 to 7 as separate layers [20]. Efficient protocol stacks are also being standardized: for example, the presentation layer can be almost completely bypassed (encoding being specified at the level of the service to the applications). In that case, the association control protocol is greatly simplified ("fast byte", "fast associate"). Although this leads to a lower overall functionality, it is largely counterbalanced by the advantages of these proposals and recommendations. The description that follows will therefore make use of the following enhancements, standardized in [21] or proposed in [19]:

- The presentation services are considered as application services, thus allowing an easier and better composition of the bricks forming the services to the applications;
- Enhanced functionality is introduced, such as data encryption and data compression which can be seen as services to applications instead of being one (almost physical) way of transmitting the data at a low abstraction level.

Prior describing in depth the conceptual elements on which the mapping will be based, a few acronyms related to the OSI world have to be defined:

- Application Layer Structure (ALS) is the set of rules defining the structure of the OSI application layer; the present version [22] was often referred to as "eXtended Application Layer Structure" (XALS) before its approval;
- Application Service Elements (ASE) are bricks used to establish the set of services used by an application; we distinguish between Specific ASE (SASE, related to a specific application) and Common ASE (CASE, generic bricks like ACSE (Association Control Service Element) or RPC (Remote Procedure Call - OSI version or e.g. Sun Microsystems' one));
- Application Service Objects (ASO) are composed by a set of one or more application service elements (ASE) and/or application service objects (ASO) and one control function (CF);

- the relationships between ASOs are defined through a control function (CF), which defines how the services are handled by the ASO (see Figure 2).

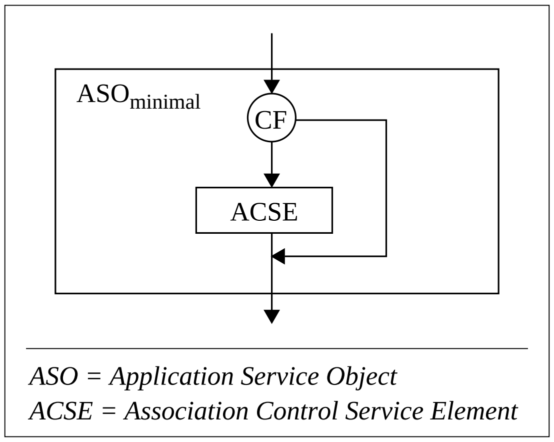

ASO$_{minimal}$
CF
ACSE
ASO = Application Service Object
ACSE = Association Control Service Element

Figure 2:Usage of a Control function (CF)

4.2 3 layers model versus 7 layers OSI model

The figure below establishes a link between the layers used in the two stacks of protocols. As mentioned earlier, the OSI transport layer has been placed into the lower layers, although it may have been placed on the border line between the lower and the upper layers.

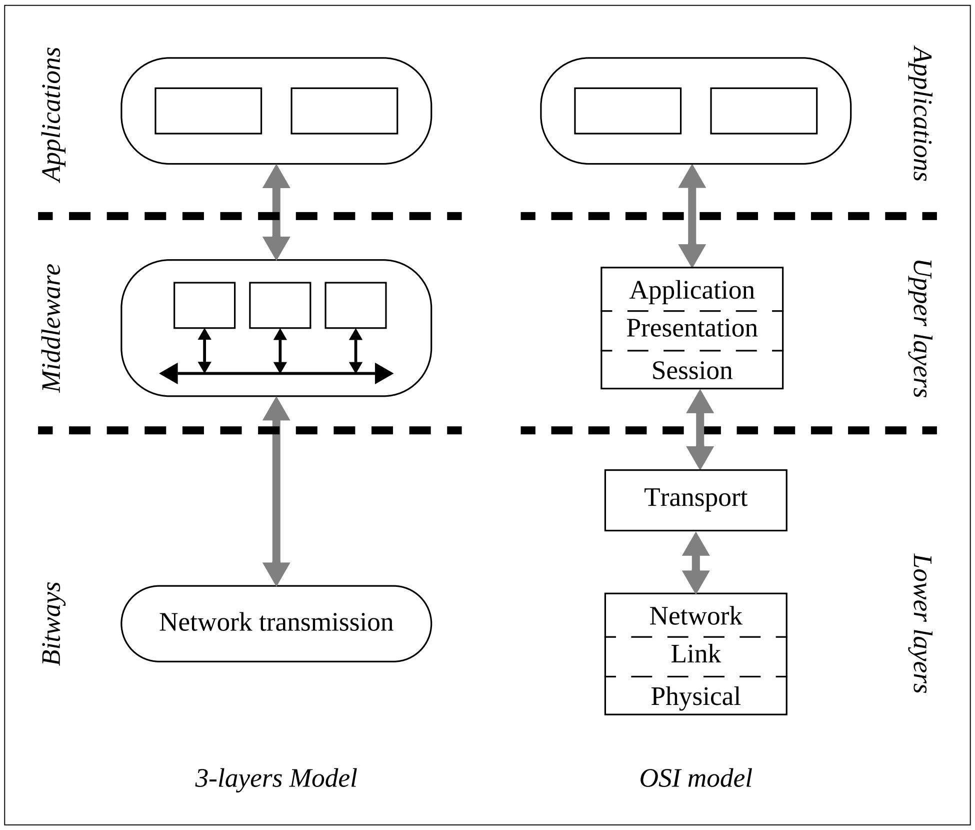


Figure 3:Equivalence between Middleware and OSI layers

The equivalence between most of the layers is obvious. Applications based on the middleware are above the OSI upper layers, and end-to-end transmission of bits and bytes is performed by the OSI lower layers.

The parallel between the structure of the middleware and the OSI upper layers can be further expressed by taking into account the specific structure of the OSI application layer as presented in the figures below, where the three middleware layers are described using Application Service Objects:

- The environment-oriented middleware is the more complex one (Figure 4). It provides six services through four ASOs: identification of patients and resources, management of the data and access to external services like display and spooling. The first two ASOs are complex ones obtained by composition of several basic application service elements while the latter two are simple ones.
- The middleware of the second generation, implementing services for the management of distributed processes, contains three ASOs (Figure 5): NICE (Network Independent Computing Environment), the HELIOS HUB environment and the ORBIX/TUXEDO object oriented environment.
- The middleware of the first generation is represented in Figure 6, where three complex ASOs are proposed. In this example, the first complex ASO is obtained by the composition of the Sun Microsystems' Remote Procedure Calls and the XDR (eXternal Data Representation) ASOs. The second one is a secured RPC, where the data is encrypted and the third complex ASO uses an OSI stack with the new OSI RPC, the association control and the packed encoding rules.

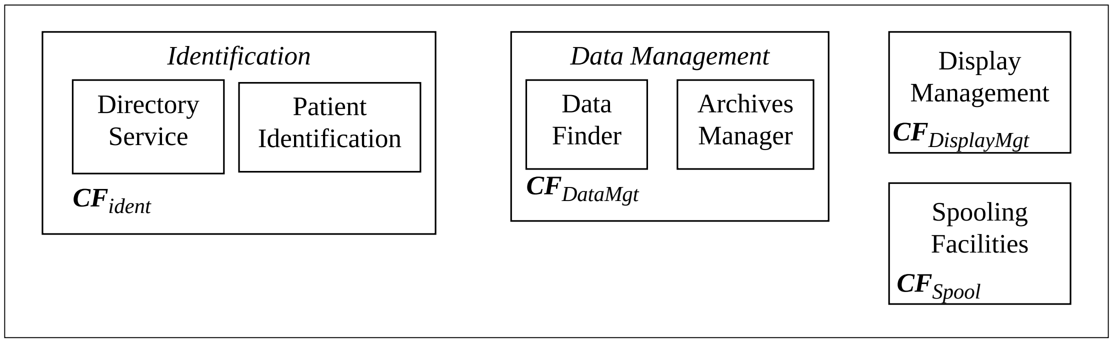


Figure 4:HIS/Level-III Middleware ASOs

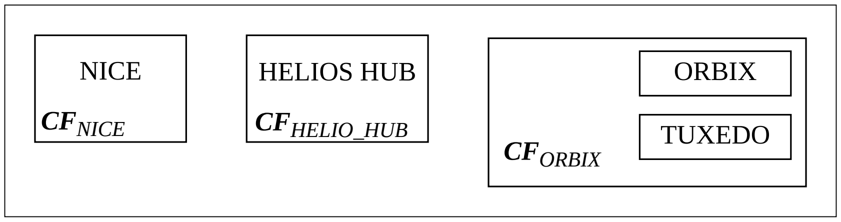


Figure 5:Level-II Middleware ASOs

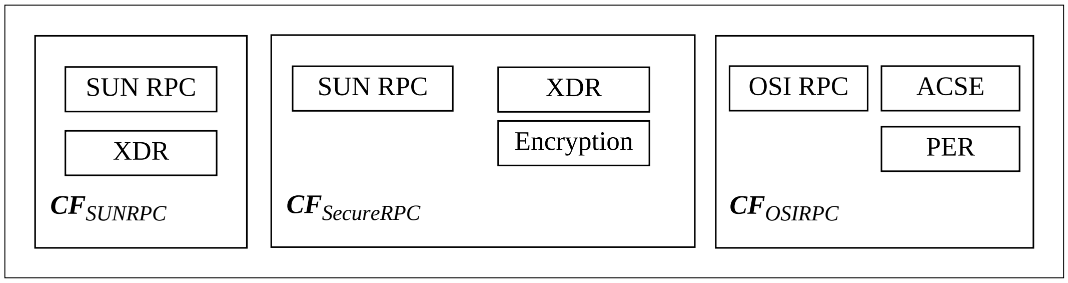


Figure 6:Level-I Middleware ASOs

Application entities (AEs) based on these ASOs can be built as shown in Figure 7, where two examples are presented:

- An application entity that could be used for getting a patient identifier or for localizing a specific resource through the directory (Figure 7a).
- An application entity that could be used for accessing data concerning a specific patient (Figure 7b).

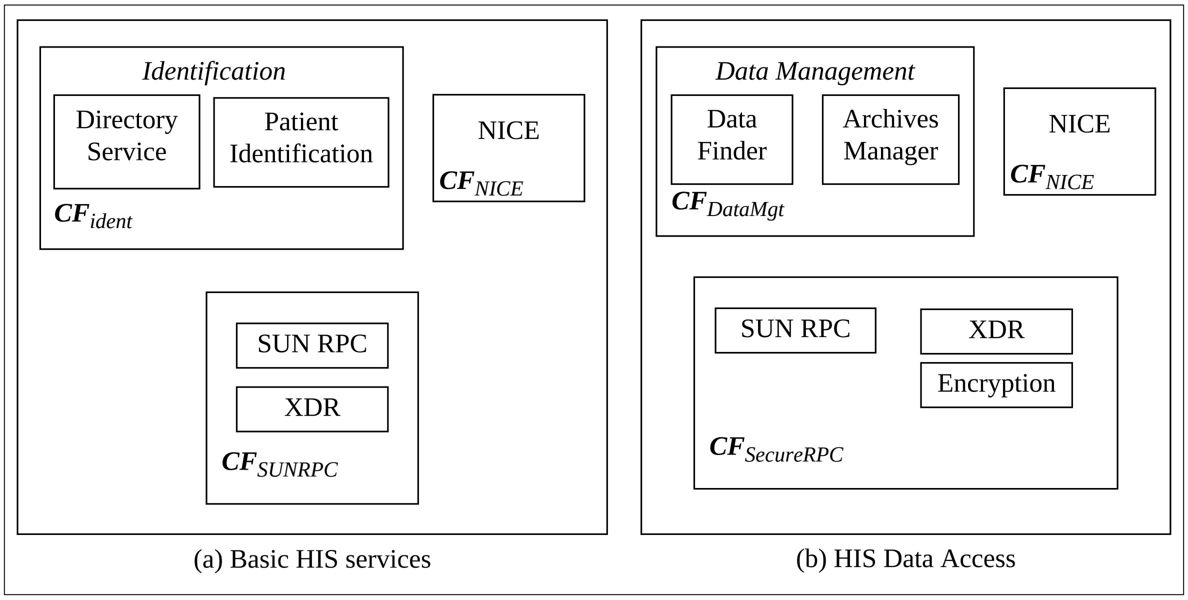


Figure 7:Examples of Application Entities

While these two Application Entities provide a functionality equivalent to the OSI layers 5 to 7, they are based on the 3-layers middleware architecture and implement valued-added services for distributed hospital information systems.

4.3 Summary

Although such a description may look like introducing several new layers between OSI layer 7 and the applications - or additional sub-layers in OSI layer 7 -, it has to be seen as a way of *describing* the functionality of the “services to applications” box for gaining a better understanding of the relationships between the components. The time when each layer was implemented independently from the other is now long past - performance and efficiency criteria took over strict layering.

Such a formalism does not only help in the description of the structure of an application. It can also be used for analyzing the links between the modules and for determining where and how parallelism could enhance the performances of the middleware stack. This is particularly important when deploying distributed applications on a large scale:

- The development of large distributed applications requires a huge amount of effort. A better understanding of what other developments have already produced, i.e. how far it can be reused in other environments, can lead to a better integration of existing specialized modules developed with portability in mind, and therefore reduces the implementation work. The formalization into ASOs of these specialized modules is a way of allowing this necessary understanding as well as a first step towards the establishment of a library of HIS-related middleware services.
- Ensuring good response time from the main servers may require the use of parallelism for implementing the communication stack. An example of what can be realized is given in [19]. A paralleled OSI stack including enhanced encoding functions (compression and encryption of data) has been realized using several transputers as a support of parallelism. Such a paralleled implementation is only possible when the “boxes” used have their internal behavior and relationships well defined and identified.

5 A case study: The SYNAPSES Project or building HIS Middleware

Hospital Information Systems are particularly concerned by the problem of reuse of software. This is emphasized by the migration towards complex distributed systems, where many applications are sharing and exchanging data. Several European projects are therefore developing strategies and standards in order to develop common distributed architectures and in order to make the reuse of software components possible. In particular, the European project SYNAPSES aims at promoting the middleware approach for HIS applications and at establishing standard interfaces for accessing HIS middleware services - typically level III middleware. According to the standardization terminology, this amounts to developing formulated standards and immediately realizing prototypes assessing the adequacy of the standards.

The SYNAPSES project does not reinvent the wheel. It rather tries to capitalize on existing or ongoing work wherever possible. Figure 8 is an illustration of this strategy, where the components of a HIS system are represented according to the structure being standardized by the Project Team 1-013 of the CEN[*] Technical Committee 251 [23]. This new standard defines three layers of abstraction: applications, common components and bitways, respectively represented in Figure 8 by applications and services, middleware and bitways.

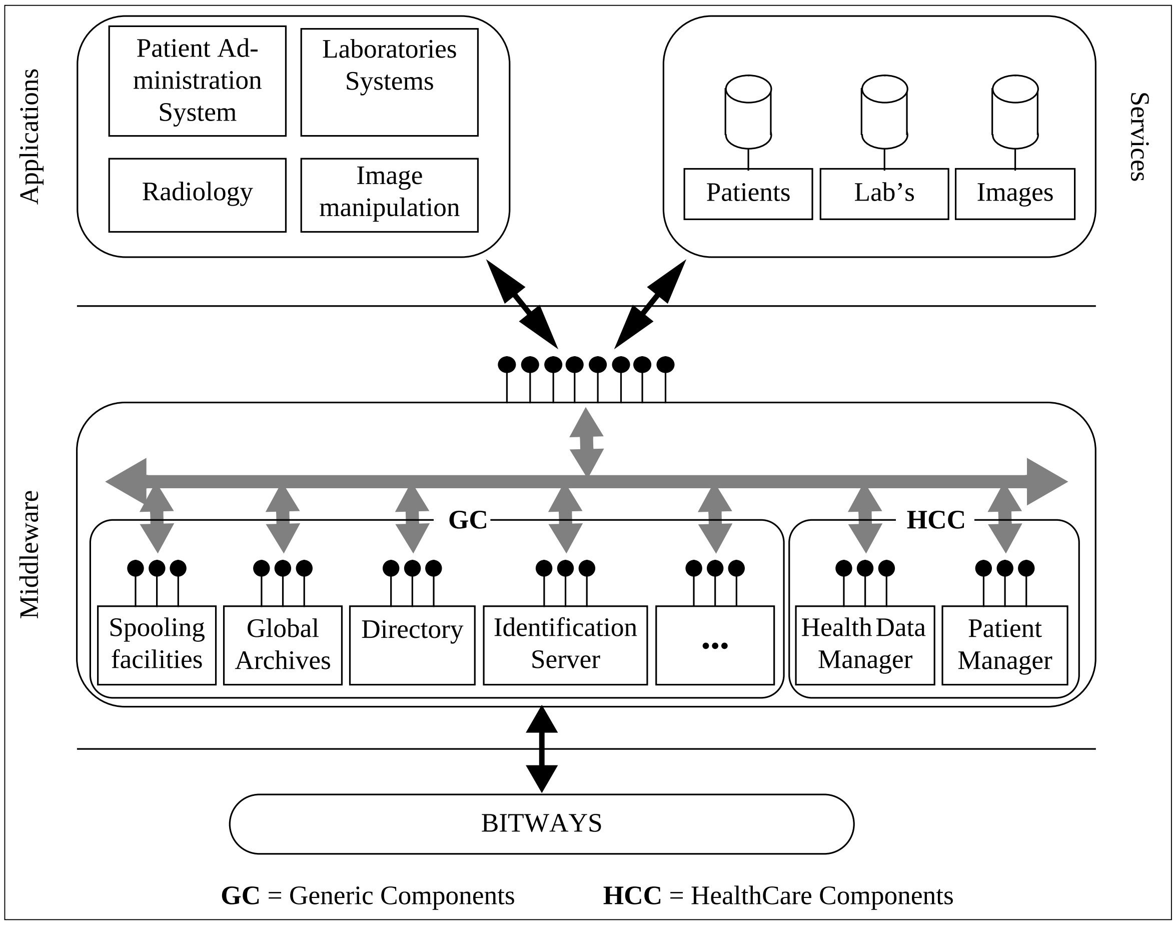


Figure 8:HIS architecture according to CEN/TC251/PT1-013

With respect to Figure 8, one goal of the SYNAPSES project is to define the internal architecture, the functionality and the interfaces of the "Middleware" elements specific to the Healthcare record architecture ("HCC"

* Comité Européen de Normalisation - European committee for standardization.

boxes in the above figure). Fully standardizing such middleware services implies the definition of two interfaces, as explained in Figure 9: one towards the application(s) and one towards the server(s).

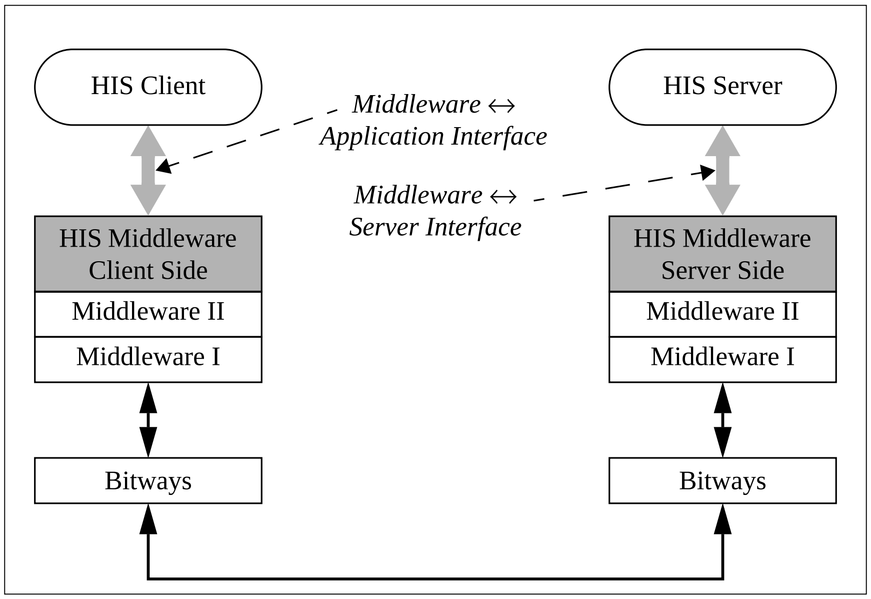

HIS Client
HIS Server
Middleware ↔ Application Interface
Middleware ↔ Server Interface
HIS Middleware Client Side
HIS Middleware Server Side
Middleware II
Middleware I
Bitways

Figure 9:SYNAPSES Interfaces

The interface between the HIS client and the HIS Middleware Services (i.e. the “HIS Middleware Interface”) makes possible the development of portable *applications*, while the interface between the HIS Server and the HIS Middleware Services is concerned with the portability of the middleware itself (it is also possibly concerned with the portability of the HIS server, but in the usual environments where such standards are deployed, changing the HIS server is of no concern).

The main result of the project will be a high level of portability of applications and of the middleware itself, level reached through the standardization of these two interfaces and through the definition of a common Federated Health Care Record expressing the available data using object-oriented technology.

6 Why using Middleware and API functionality: Advantages and ease of use

Middleware-based approach for developing software environments can just be seen as a way of decomposing and classifying functions. Fortunately middleware is not simply another way of describing the same things, middleware also offers strong advantages apart from the aesthetic ones:

- The most important element of a computing environment is the applications. The focus of the developments should be essentially user-oriented, and this is especially true in healthcare centers where the users are regularly changing. Using standard middleware components enables the developers to concentrate on what the users need, i.e. intuitive and well designed user interfaces.
- The sharing of applications is becoming crucial for most organizations, especially when they cannot afford any more to develop all applications by themselves, and when they do not want to be too specific to a given environment. By the clear definition of APIs to middleware services, a higher portability of both applications and servers can be reached, enabling thus their reuse by other organizations.
- In environments such as the one presented in SYNAPSES, the sharing of data is a crucial activity in order to reach a certain level of cost-effectiveness in healthcare. However accessing data from another system still means too often using a specific protocol for retrieving the data. Looking at sites like the University Hospital of Geneva, where more than 50 heterogeneous databases coexist, developing specific tools for enabling each application to access many of these 50 different database managers is unfeasible. The only realistic solution

is to use a middleware based approach, concentrating all the specific elements into the application services layer, i.e. the middleware layer, and providing the applications with a uniform view of all databases. The heterogeneity of the distributed system is then hidden behind the middleware layer.

Although one can think of middleware as yet another set of layers between the applications and the real world, it has to be noted that the functionality implemented by the three identified middleware layers has to be provided anyway. As now clearly mentioned in the standard describing the OSI upper layers structure, there is no reason for implementing physically separated middleware layers. However the clear identification of what is middleware and what is not allows to independently develop the common components from the application and the bitways layers. This is an important step towards the establishment of a library of middleware modules.

The increasing consensus on the importance of the middleware layers proves that it is more than just the present way of thinking. There is really a consensual trend for the development of complex distributed systems around middleware functionality, and especially around specialized middleware. Structuring the middleware into three levels of functionality seems to us to be the best fit regarding the present trends in the development of complex distributed systems. There is no reason for having either more or less logical middleware layers. But while the pathways across the middleware levels I and II are now clearly identified and standardized, it is not yet the case with the third level. There is a risk of getting n^2 solutions, ending with the need for middleware for interconnecting middleware! In order to avoid this, it is vital to promote agreed or de facto standards so that they are adopted by the widest possible audience. This is the real challenge facing the development of the specialized middleware.

Standardizing middleware implies two tasks: identifying standard functionality to be provided by middleware modules and standardizing the APIs to these modules. While the former is commonly admitted, we are far from a consensus for the latter. The API itself, or the service objects in an object-oriented world, is nevertheless as important as the services themselves. Without a common protocol for accessing the services, a true portability cannot be achieved! This has been recognized already in many domains, such as databases systems where not only the basic concepts of tables, indexes, and so on are common to all vendors but also the way to retrieve the data through the standardized SQL language. Although it is not a real API, this common query language can be considered as a precursor of an API to the databases. A true de facto standard API is the Internet socket library: the procedures and their parameters are the same on most if not all platforms providing this functionality, and are even reused and extended in several Internet Protocol version 6 informational documents (e.g. [24]).

Unfortunately most expressed standards are not defining the APIs, generally considered as “implementation dependent stuff”. This gap often lead to incompatible implementations of the same standard. While the gap can be filled by the definition of what is called in Europe a “Profile” complementing the standard and places constraints on implementations, this is an additional work delaying the production of interoperable implementations.

7 Conclusion

Most large organizations view the evolution of their information technology infrastructure as evolutionary, not revolutionary. This translates into the requirement to support heterogeneous configurations in which proprietary legacy systems, supporting operational applications, interact with new applications.

Distributed systems support the current business trends and provide direct answers to most of the needs of the market and the users. They provide direct support for decentralized business units, making use of their own local processing, while being given access to company-wide information systems. Federation of systems complies with flat organizational structures (lean management). They promise to maximize the use of networked resources, services and databases, leading to a minimization of costs.

Distributed systems provide an environment where innovative off-the-shelf applications can be developed and distributed, provided appropriate measures are taken to reduce their cost of entry and to broaden their potential market. Such applications should be seen independently of the physical structure of the system. This need is fulfilled by the transparency of distributed systems. Users finally perceive open systems as an opportunity to obtain equivalent products from several suppliers.

These issues are nonspecific to medicine and healthcare! Federation is seen to be the key structuring principle to combine components of a system. Finally, worldwide interconnection is likely to have a considerable impact on the development of various types of distributed information systems. «Openness» and «Interoperability» are the key issues.

The client-server approach is even more broadly illustrated by the extensive use of Internet / WWW / Java, and is more and more open to remote access services rather than to systematically importing copied records of scattered remote databases. As an example the ExPASy WWW server set up at the University Hospital of Geneva and the Medical Biochemistry Department of Geneva University [25], has links from SWISS-PROT database to remote databases like EMBL (nucleotide sequences), PDB (three dimensional structures), Medline (bibliographic references from the National Library of Medicine), PROSITE (protein sites and patterns), OMIM (On-line Man Inherited Mendelian disease database of Johns Hopkins) and quite a few others.

Besides, regarding the new client-server distributed information systems, there are more and more application services on top of the APIs of the basic Application Service Elements. These new services, such as for instance the patient identification server and the archives server constitute a kind of upper middleware which on one hand make domain-specific applications more easily portable from one environment to another and on the other hand allow the interoperability with other heterogeneous sub-systems.

Hiding heterogeneity by using specialized middleware is now a widely agreed trend, but there is a serious lack of methods and tools for describing the various middleware components and their relationships. The importance of the present work is indeed in the formalization of a generic way to describe middleware elements along with their mutual relationships. Thanks to the approach adopted here, the progressive construction of HIS from the same common elements is made easier and the comparison of systems becomes possible. This will be highly valuable for avoiding the n^2 approach in interconnecting heterogeneous distributed systems. It is clear that other formalization methods could have been adopted for this purpose, but we consider that the new version of the OSI Application Layer Structure is now designed well enough for such a work.

8 Acknowledgments

This study was supported by the SYNAPSES project that is funded by the European Union from the Telematics in Healthcare program (contract number HC 1046). The Swiss part of the project is fully supported by the Swiss government (Federal office for Education and Science: contract number OFES 95.0286).

9 References

[1] Bernstein PA. Middleware: A Model for Distributed System Services. Communications of the ACM, vol. 39, No. 2, February 1996, 86-98.

[2] King SS. Middleware! Data Communications, March 1992, 58-67.

[3] Korzeniowski P. Middleware: Achieving Open Systems for the Enterprise. Computer Technology Research Corp., Charleston, South Carolina, USA, 1997.

[4] Scherrer JR, Baud R, de Roulet D. Moving towards the future design of HIS: A view from the seventies to the end of the nineties, the DIOGENE paradigm. Hospital information systems: design and development characteristics; impact and future architecture. H.U. Prokosch, J. Dudeck ed., Elsevier, 1995:347:375.

[5] RFC 542 NIC 17759: File Transfer Protocol, August 1973.

[6] Scherrer JR, Revillard C, Borst F, Berthoud M, Lovis C. Medical Office Automation Integrated into the Distributed Architecture of a Hospital Information System. Methods of Information in Medicine, vol. 33, 1994:174-179.

[7] Sun Microsystems, Inc. NFS Administration Guide. Mountain View, CA, USA, 1995.

[8] Open Software Foundation. OSF DCE Administration Guide - Core Components. PTR Prentice Hall Inc., 1993.

[9] ISO/IEC 9072-1:1989 Information processing systems -- Text communication -- Remote Operations -- Part 1: Model, notation and service definition (Provisionally retained edition)

[10] ISO/IEC 11578:1996 Information technology -- Open Systems Interconnection -- Remote Procedure Call (RPC)

[11] ROSE MT. The ISO Development Environment: User's Manual. The Wollongong Group, 6.0 Edition, January 1990.

[12] PTC/96-03-04: CORBA 2.0 Specification. Object Management Group Inc., July 1995.

[13] The HELIOS Software Engineering Environment. Computer methods and programs in biomedicine supplement, Elsevier, December 1994, vol. 45, suppl. pp. S1-S152.

[14] http://www.iona.com/Orbix/index.html

[15] http://www.gemstone.com/

[16] Rishel W. Pragmatic Considerations in the Design of the HL7 Protocol. Proceedings of the thirteenth annual Symposium on Computer Applications in Medical Care, November 5-8, 1989, Washington DC, 687-691.

[17] Cypser RJ. Evolution of an open communications architecture. IBM SYSTEMS JOURNAL, Vol. 31, No 2, 1992:161-188.

[18] Day J. The (Un)Revised OSI Reference Model. ACM SIGCOMM Volume 25, Number 5, October 1995, 39-55.

[19] Spahni S. Parallélisation des couches supérieures du modèle OSI: stratégies et mise en oeuvre. Editions systèmes et information, thèse de doctorat no 2588, Genève 1993.

[20] ITU - Telecommunication Standardization Sector, Temporary Document 5013, Study Group 7, Question(s) 22/7. Draft rec. X.osieff (as per SC 21 N 10451). Geneva, 10-21 March 1997.

[21] ISO/IEC JTC 1/SC 21. Open Systems Interconnection, Data Management and Open Distributed Processing. Revised text of ITU-T X.226/Amd 1 | ISO/IEC 8823-1/DAM 1, CO Presentation Protocol / Amd 1: Efficiency Enhancements. Geneva, 17-19 March 1997.

[22] ISO/IEC 9545:1994 Information technology -- Open Systems Interconnection -- Application Layer structure

[23] Healthcare Information System Architecture. CEN TC/251 PT 1-013.

[24] Gilligan, et. al. Basic Socket Interface Extensions for IPv6. Network Working Group, Request for Comments: 2133, Category: Informational, April 1997.

[25] Appel RD, Bairoch A, Hochstrasser DF.Trends in Biochemical Sciences (TIBS), 1994, 19:258-60

[26] Coulouris F, Dollimore J. DISTRIBUTED SYSTEMS: Concepts and Design. International Computer Science Series, 1988.

[27] Von Bochmann G. Concepts for Distributed Systems Design. Springer-Verlag, 1983.