A Survey of Web Information Systems

Alberto Silva
Alberto.Silva@inesc.pt
IST/INESC

M. Mira da Silva
mms@dmat.uevora.pt
University of Evora

José Delgado
Jose.Delgado@inesc.pt
IST/INESC

Abstract:
The main objective of this paper is to survey, in a schematic and generic way, the major models and approaches concerning the design, construction and execution of Web information systems. We identify three different approaches – server-centric, client-centric and distributed applications – and discuss the major strengths and weaknesses of each approach in a Web context. Finally, we present our view of those models and technologies that are needed for building the kind of distributed, dynamic information systems that will exist in the near future.

1. Introduction

The Web grew fast from a distributed hypermedia system, with basic navigation and information retrieval capabilities, to what we call an "umbrella system" of several and distinct applications (also called information systems or simply IS) eventually accessible at a world wide scale.
These days, Web Information System (WIS) technology involves many efforts, interests and investments all over the world, spread by the main research and academic centers as well as the major IT companies. Due to this highly dynamic situation, it *is not* the aim of this paper to describe specific tools, environments, or products concerning WIS development because they would be quickly outdated by new generations of (yet more competitive and complex) products.
The main objective of this paper *is* to describe, in a schematic (see schema in annex) and generic way, the *main models or approaches concerning WIS design, construction and operation*. Consequently, we discuss first the major strengths and weaknesses of the current Web technology and will also depict what models and technology of the distributed and dynamic WIS should exist in the future.
The paper is a result of our real-world experience on designing and developing large distributed applications, as well as our research work. More recently, the first author has pursued research on developing distributed information systems on the Internet [SBD95,SAD96,SMdSD97], the second on higher-order distribution mechanisms for persistent programming languages [MdS96,MdSS97], and the third on visual development environments and intelligent building systems [DSCOB95].
The paper is organized in six sections. The next section presents the server-centric WIS approach (i.e., the class of WIS in which activity is concentrated on the server machine) and section 3 presents the client-centric WIS approach. In section 4 we describe a new approach to WIS based on distributed infrastructures, where the activity is effectively divided between client and server machines. Finally, section 5 summarizes and compares all approaches, including the future models for WIS.

2. Server-Centric WIS

We define as server-centric WIS those information systems based on the Web that concentrate their activity on the server side. In this survey we will describe the common gateway interface, server side includes and proprietary server interfaces like ISAPI. The section ends with a comparison between these variants of the basic server-centric WIS model.

2.1 Common Gateway interface (CGI)

The first server-centric WIS were based on the CGI (*Common Gateway Interface*) [Rob96]. This interface consists in a generic and simple specification that defines a communication interface between a Web server and any specific process that runs in the same computer architecture.
CGI is a flexible and extensible way to add functionality to a Web server, such as: data base access, data translations, protocol conversions, and so on. CGI programs are typically written in a well-know language (such as C or C++) or in a simple, powerful script language (such as Perl).
After the construction of the first real WIS based on the CGI, some variants happened, either for simplicity reasons (Server Side Includes mechanism, see section 2.2) or due to performance stress (Web server proprietary APIs, see section 2.3). Nevertheless, all server-centric WIS share the following common issues:

- WIS are called "short-life" processes because Web servers are stateless and the HTTP protocol is connectionless. As such, a CGI process is be responsible for maintaining state (persistent data) between successive user interactions. There are some known techniques for simulating state between different interactions (accesses) in the same session, for example: using: URL information (e.g., *QueryString* and *Path Info* components); visible or hidden form's fields; cookies; etc.

- The majority of these applications are HTML document generators, which may vary between trivial or very complex. This means that the main function of any WIS is to parse a set of data sent by the Web client component (the browser) and produce at run-time a convenient response, which is typically in HTML format [SBD95].

Figure 1 presents the computational model of WIS based on the CGI approach. Specific components of the application are presented with emphasis (gray color): the CGI process and the purposely non identified "??" component. This "??" block is usually a more specialized process with which the CGI process would have to communicate, for example, a file server, a data base management system, a geographic information system, or any other more specific process. The other blocks – Web client and server – are generic, just used as support components (consequently, in this paper, we don't give them any special relevance).

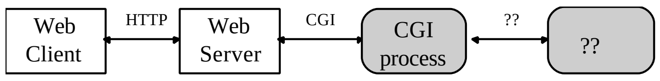


Figure 1: WIS computational model based on the CGI approach.

2.2 Server Side Includes (SSI)

The Server Side Includes (SSI) mechanism was originally introduced in the NCSA's Web server [NCSA95a] to facilitate the inclusion of simple extensions to the Web server.

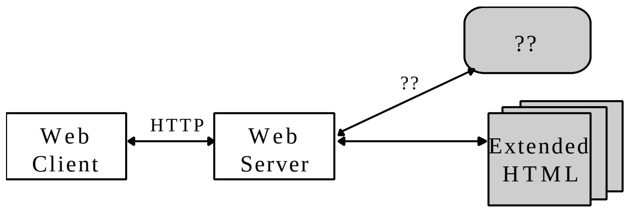


Figure 2: WIS computational model based on the SSI approach.

Figure 2 depicts its generic functionality. The application is basically composed by a given set of HTML files with some extended (non standard) elements called *tags.* When the Web server receives a request for some extended HTML document, it parses the corresponding file and translates every tag into a set of "standard" HTML elements. To translate the extended elements, the Web server may eventually need to invoke external processes (such as CGI programs, DBMS, etc.) which are represented by the "??" block with a "??" interface.

SSI was originally conceived for the NCSA Web server. However, it has inspired some equivalent mechanisms, although more flexible and skillful, such as Microsoft's Internet Database Connector [Mic96a] (designed for accessing databases via ODBC) and WebQuest's SSI++ [Ques96]. Nevertheless, its main goal was just to enable the development of simple WIS in an easy and fast way – even without the need for any programming language.

2.3 Proprietary APIs

In this approach, WIS use an application programming interface (API) proprietary to a specific Web server such as those from Netscape or Microsoft. Unlike both CGI and SSI, this permits to write and load new programs in the same memory space of the Web server itself. Sun's Servlets (Java components executed on the server-side) technology may be also classified in these approach.

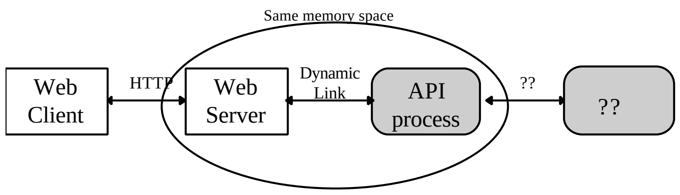


Figure 3: WIS computational model based on the proprietary Web server API approach.

Figure 3 depicts the computational model corresponding to this approach. The load (also called binding or linking) operation is executed once at server boot time or on demand each time they are invoked by a client request. This approach doesn't solve the "short-life cycle" problem presented by all server-centric WIS, but eliminates, for each access, a new process creation. The only requirement is the support to dynamic linking of executable modules by the operating system, an almost standard feature in modern OS like

Windows 95.
Currently, almost all well-known Web servers provide their own proprietary API, for example: Netscape's NSAPI (*Netscape API*), Microsoft's ISAPI (*Internet Server API*), and Oracle's WRBAPI (*Web Request Broker API*). However, since it is impossible to support a large number of APIs, most software development tools such as Borland's Delphi choose to support only one or two of these, typically Microsoft's ISAPI (when based on Windows) and/or NSAPI (since Netscape is more popular in the commercial UNIX world).

2.4 Comparative Analysis

In this section we summarize and compare the three approaches: CGI, SSI and API. Obviously, the classification presented in Table 1 represent only high-level classification and are not intended to be used as a means to decide which approach is the best.

Analysis criteria	CGI	SSI	API
Application performance	☹	😐	☺
Application scalability	😐	☹	😐
Application safety	☺	☹	☹
Application flexibility	☺	☹	☺
Easy development	😐	☺	☹
Application maintenance	😐	☺	☹
Vendor-neutral	☺	😐	☹

Table 1: **Comparative analysis of server-centric WIS approaches.**

The aim of this analysis is to enable a deep reflection of the strengths and weaknesses of the different approaches, and as a result the following criteria are analyzed: runtime absolute performance; scalability (the ability to support a large number of client requests); safety (the consequences to the Web server if a malfunction occurs in the application); flexibility and versatility; easy development and technical skills required; application management and maintenance; and open (vendor-neutral) against proprietary solution.

WIS based on CGI

These have low-level performance due to their deficient management of processes and memory. However, they can support some degree of scalability via replication of CGI processes on different machines. (This is achieved by a special load balancing algorithm in the code of each CGI application.)

This approach presents a high level of safety because applications are external to Web servers, meaning that if the process crashes it doesn't imply a corresponding Web server crash.

These CGI applications are very flexible and even complex because they can be developed independently of the Web server using any conventional language (C, C++, Perl, Basic, Java) and development environment (Visual Basic, Delphi, PowerBuilder).

The development and test cycle is moderated. It all depends on the capabilities of the chosen language and development environment and also depends on the existence (or not) of specialized software libraries supporting the CGI particularities. These libraries typically give support for: state maintenance; HTML code generation, access to a DBMS; etc. Technical skill requirements, application management and maintenance is medium or high depending on the very same characteristics.

Due to its simplicity and support, it was adopted by the generality of Web servers. CGI then became, and still is, the standard *de facto* for building WIS.

WIS based on SSI.

This approach is basically a substitution mechanism, in which tags (non-standard HTML elements) are converted at "request-time" to a corresponding set of standard HTML elements. Examples include: the result of a system call (e.g., current time) or the result of a SQL query.

SSI presents a medium/high performance because the server has to parse the page and substitute all tags on it every time that page is requested. As a result, scalability is not well supported.

Its main drawback consists of the limited capabilities concerning the development of flexible and complex applications, which are impossible without a computationally complete programming language like C.

If the application crashes when translating one of these tags, that implies the crash of the corresponding Web server. Also, it is difficult to develop an independent-vendor application because each vendor defines its own set of tags. Nevertheless, its corresponding development is easy and fast for very simple applications.

WIS based on Proprietary APIs

Applications based on this approach are flexible and can be fairly complex, and they also present high-level performance. Due to this flexibility, it is possible to develop the same kind of algorithms as those of the CGI approach in order to support some degree of scalability.

However, because these applications are now executed in the same address space as that of the Web server, this implies the Web server will crash whenever a crash occurs in the application. (This can be prevented by providing separate address spaces, like Oracle's Web server, but then performance and scalability cannot be the same.) Also, these applications require higher technical skills than other approaches and this implies a higher cost and risk of development, management and maintenance.

3. Client-Centric WIS

After the deployment and generalized use of the first real-world server-centric WIS, their main limitations became apparent: difficult to support complex and/or long transactions; low performance; and poor end-user interaction. This new approach, based on client-centric WIS, appeared as a solution to these limitations. (Although, as we will see, they could not solve all of them and also generated new problems.)
Client-centric WIS can be divided into two distinct groups: applications based on *previously installed code*; and applications based on *mobile code*.

3.1 Previously Installed Code

In order to extend the Web client with new facilities, it is necessary to provide an API so that it can communicate with the application providing them. CCI (Common Client Interface) [NCSA95b] and CCI++ (Constituent Component Interface ++) [AM95] are examples of APIs that were proposed to provide a better cohesion and integration between the Web client and specific applications.
Based on CCI++, Netscape provides a proprietary API to enable third-party developers to write external applications that run tightly connected to its Web client (Netscape Navigator).
These applications are designated in market parlance by *plug-ins* [Net95a] and are just dynamic link executable modules (DLLs in Windows) that follow the Netscape proposed API. These plug-ins are previously installed in the client computer, and are typically responsible for handling one or more multimedia document formats (MIME types). The process works like this: whenever Netscape receives a non-standard MIME document (identified as such in the document header) the corresponding plug-in – that was previously installed and registered by the Netscape user – is dynamically loaded in the same computational context of the Web client. (This decision is responsible for some crashes if the plug-in is not robust enough.)
Figure 4 puts an emphasis (gray color) on the following components: the plug-in (an application previously installed); the MIME document(s); and some other application executed on the server machine that is responsible by delivering the MIME document (e.g., file server, video producer).

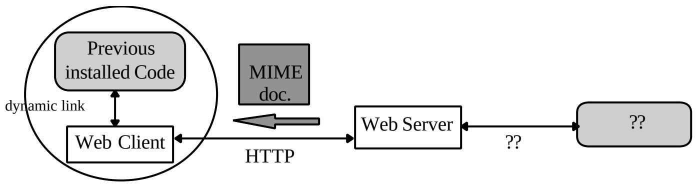


Figure 4: WIS computational model based on previous installed code approach.

Applications that follow this approach are typically very general, complex and produced by specialized software houses. Examples include viewers (e.g., for Microsoft Word documents), players (real-time sound and video) and editors of well-known multimedia formats (e.g., Acrobat's PDF, VRML, MPEG2, and so on). Also included in the category are more complex applications such as spreadsheets editors and byte-code interpreters (also called virtual machines). For example, there are plug-ins for running Java byte-codes provided by Netscape, Microsoft and Colusa.

3.2 Mobile Code

The plug-in approach already supports a reasonable integration between a Web client and its external applications. However, this approach still presents some limitations, especially regarding flexibility and portability. In addition, plug-ins require a previous installation made by the user, further complicated by the need to update for newer software versions (from alpha to beta, from beta to final, from 1.0 to 2.0, and so on).
A more general WIS approach is based on mobile code [Con95,BTV96]. This approach is based on a program stored in some computer (typically the Web server) which can be transportable (or copied, moved, etc.) across the network to another computer in order to be executed remotely.
The mobile program is written in virtually any programming language, although some languages are more suitable than others to achieve this task. For example, interpreted (scripting) languages are inherently more portable then traditional (compiled) languages because an executable is based on the machine's native architecture and will not run in a different machine architecture. However, there is an obvious loss of performance if script languages are used directly without any compilation.

Figure 5 shows the architecture of the mobile code approach. The code for the specific application (mobile program) is stored on the Web server machine (the '??' block) and is transferred on-demand to the Web client machine where it is executed – typically by a "virtual machine" block. The client has to provide a virtual machine responsible for the safe interpretation and execution of the code received. (However, if the mobile program is native code, then the it will run directly on top of the operating system.)

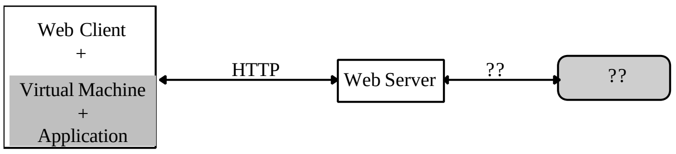


Figure 5: WIS computational model based on mobile code approach.

We found basically two distinct WIS approaches based on mobile code: mobile code embedded in HTML documents; and mobile code independent of (separated from) HTML documents.
In the first approach – *mobile code embedded in HTML documents approach* – the Web client, beyond its regular capabilities of parsing and rendering HTML elements, also needs capabilities for interpreting and executing embedded code (usually as source code). This approach has two main advantages: it allows the construction of small and simple WIS with reasonable end-user interaction, and supports the integration between HTML documents and other Web technologies (Java applets, plug-ins, etc.) what has been called "glue technology".
The majority of scripting languages in use today on the Web were either adapted to be used under this approach or originally designed for it. Examples include JavaScript [Net95b], VBScript [Mic96c], Tcl [Ous94], and Obliq [Car95].
In the second approach – *mobile code independent of HTML documents approach* – there is usually a virtual machine that executes the code referred in an HTML document but located in a separate file (usually stored on the same server machine, although not strictly necessary). As a consequence, the code can be referred several times inside the same HTML document or even in many documents. Figure 6 show how it all works.

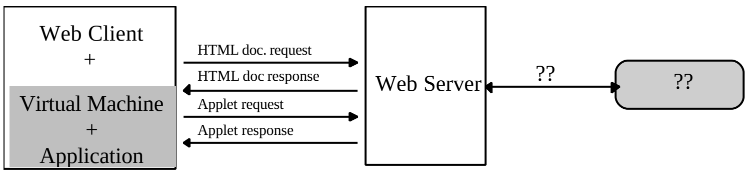


Figure 6: Mobile code, HTML document independent.

The mobile code is transferred and executed in text or binary format, and it should be noted that nothing prevents a program written in a script language to be used in a similar manner. The only apparent reason for this distinction is that script languages are represented as text and as such can be included inside an HTML document, while a general compiled program cannot.
Examples of this approach include Java applets [AG96] and Active-X controls. There are also many research prototypes based on this approach, such as Tcl [TB96] and Obliq [BN96].

3.3 Comparative Analysis

In this section we summarize and compare all client-centric WIS approaches described before. In addition to the criteria analyzed in section 2.4, security will also be analyzed here.

Analysis criteria	Previously Installed Code	Mobile Code Embedded in HTML	Mobile Code Independent of HTML
Application performance	☺	☹	😐
Application safety	☹	😐	😐
Application flexibility	☺	☹	😐
Easy development	☹	☺	☹
Safety/Security	☺	😐	😐
Application maintenance	☹	😐	☺
Vendor-neutral	☹	😐	☺

Table 2: Comparative analysis of client-centric WIS approaches.

WIS applications based on *previously installed code* present high-level performance mainly due to these reasons: the application code is already in the client machine when it is invoked and perhaps it is even loaded already in memory; and the code is compiled to the corresponding machine architecture and runs natively in the client computer. But since the application shares the memory space with the Web client, this approach also presents some safety problems, although it doesn't raise any security problem.

On the other hand, this approach requires a programmer with high-level technical knowledge to develop these specialized applications and also requires the user to manually install the program and maintain its versions. Finally, this approach is based on proprietary Web client APIs.

Mobile code embedded in HTML documents.

Applications based on present low to medium-level performance mainly due to two reasons. First, the code has to be transferred from the server to the client machine. Second, the code has to be interpreted directly from its text format and this is highly inefficient.

Furthermore, because these programs are downloaded from the Web server on-demand, their operations should be restricted for security reasons. (Just imagine an application that, when downloaded, removes all files in your hard disk.) For example, it is typical to restrict these applications from accessing the local file system or from establishing arbitrary network connections. However, there are still no good solutions for the resource (memory and CPU) exhaustion syndrome. Consequently, these applications should be over-restrictive and as a result are not very flexible.

On the other hand, their version maintenance and management is easily done. They did not receive the highest mark because small changes in an application may eventually require several changes in several HTML documents, but this can be solved by storing the code separately and simply referring to it from within the HTML document.

Mobile code independent of HTML documents.

Applications based on this approach also present, depending on the application and the adopted technology, a low to medium-level performance. This is because the code also has to be transferred (and, sometimes, also interpreted).

Unlike the previous model, the code usually runs natively or is interpreted in a more efficient format – virtual machine byte-code, something half-way between source code and native format. There are also many ways to improve the performance of these applications, for example, Java applets have been improved with so-called "just-in-time compilation" (dynamic compilation just before execution) and operating systems that support Java byte-code natively (and, more recently, even micro-processors designed exclusively for Java).

One of the most important strengths of this class of applications is its easy maintenance and management due to the fact that all code is stored centrally in just one place. However, the same security issues raised for the previous approach (mobile code embedded in HTML documents) also have the same solutions and problems as discussed above. Consequently, these applications present a medium level in flexibility and versatility.

Lastly, in spite of several announcements about development environments based on (Delphi-like) visual interfaces, this approach still requires a high level of technical skills. (The difficulties are raised not by the language itself, since Java is simple, but as a consequence of the many libraries that are needed for any Java program.) Lastly, due to Java success and its support by the majority of software companies, applications based on Java are in general vendor-neutral. However, the recent initiative by Sun and IBM – called "100% Java" – is perhaps a first sign that this neutrality is being put under serious challenge by Microsoft and others.

ActiveX controls

Before finishing the section, a special note should be written about the Active-X technology from Microsoft since it is so well-known. From our point of view, ActiveX can be considered an hybrid approach. From the mobility (remote transfer) and automatic configuration perspective, they are equivalent to Java applets: they are downloaded on-demand by the Web client when it founds a reference in a HTML document. On the other hand, from the portability and flexibility perspective, Active-X controls are similar to Netscape plug-ins: they are general applications that run natively and will probably be resident in the local machine (since previously downloaded controls are cached locally for efficiency reasons).

Consequently, applications based on Active-X controls present high-level performance and do not require elaborate maintenance or management. However, since controls run natively and have access to all computer resources, they present several security problems. Microsoft has partially solved this problem by digitally signing its own and other ActiveX controls. This gives some guaranties – the user knows who or which company wrote the control and can always choose not to execute it – but does not solve other security problems. (For example, these days many people are suspicious of less well-known companies that may include a Microsoft recently acquired company or a subsidiary.)

4. WIS Based on Distributed Infrastructures

The client-centric WIS approach solves some of the problems found in the server-centric approach, especially those related with performance and end-user interaction. However, it still does not give an adequate answer to the remaining issue: how to handle complex transactions between the end-user (at the client side), the application logic (now at the client side) and the database (at the server side).

This is because the HTTP protocol will never support efficient client-server communication – at least not until a new version of HTTP is widely accepted and deployed – due to its inherent connection-less mode. For the very same reason, HTTP cannot cope well with certain classes of services. These include applications involving real-time multimedia (e.g., real-time video or audio) and those accessing database systems for anything more complex than just reading a few simple records. Examples of this last class of applications are those based on long (eventually nested and/or distributed) transactions and those that manage large sets of very complex data, such as geographical or temporal information. For these kind of applications, the simple HTTP protocol and its HTML associated format are simply not enough!

As a consequence, a new approach was proposed to handle the kind of complex applications not well supported by the previous approaches described above. In this new approach, depicted in Figure 7, the Web client and the Web server are just used for the initial interaction, i.e. to establish the connection and/or to transfer the corresponding (mobile code) application. After that, the client (non-Web) application establishes another (non-HTTP) connection with its proprietary (non-Web) server. This means the application itself is not actually based on the Web since it does not need the Web infra-structure anymore.

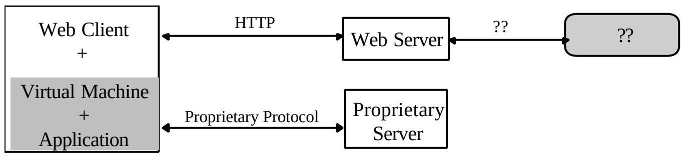


Figure 7: WIS computational model based on combined approaches.

The main advantage of this approach is obvious: once free from the standard Web shortcomings, the application and its server can now take full advantage of well-known distributed systems technology like message passing, RPC and other proprietary communication protocols. However, this approach creates difficulties regarding the integration and interoperability between different applications that follow this approach. In particular, each application will be incompatible with any other application!

As a result, a number of compromises were found based on current Web technology and a (more or less) standard distributed infrastructure. Figure 8 shows the basic idea in which the Web is used as a worldwide mechanism for finding and accessing a given application. In addition, the Web client is still used to provide a consistent human-machine interaction based on HTML and/or Java windowing toolkit [Yu96].

In this distributed approach, the application is effectively divided between a client and a server sub-application. The only difference to a standard Web application is that applications based on this new approach now communicate via the distributed infrastructure (instead of using HTTP and HTML) for efficiency reasons. Below we briefly describe the two main technologies that are used in these environments, respectively based on CORBA and Java.

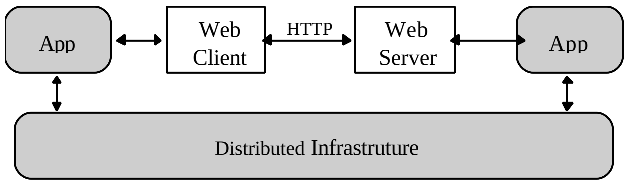


Figure 8: WIS computational model supported by a common infrastructure.

4.1 Based on CORBA

The *Common Object Request Broker Architecture* (CORBA) [Spi96] seems to be a likely solution to the approach depicted in Figure 8. CORBA is a standard model for what an RPC-like communication mechanism should offer at the application level, and has been implemented by an (increasing) number of software vendors. The latest CORBA 2.0 specification even proposes a standard communication between different CORBA implementations so that all CORBA products will eventually inter-operate between them.

A solution based on CORBA and the Web effectively integrates the two most popular distributed technologies in use today and seems a excellent candidate for developing distributed applications. Figure 9 depicts the CORBA approach to Web development, in which Java can be used at the client side as an add-on to the Web.

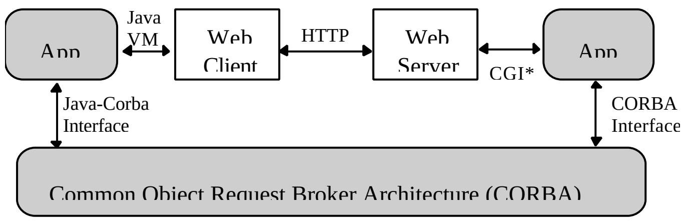


Figure 9: WIS computational model supported by CORBA based infrastructure.

It should be noted here that CORBA needs some programming capability at the client side – so Java or some other programming language will have to be used – but that is only natural if we assume this approach was proposed precisely because some programming was needed at the client! As an alternative, JavaScript or VBscript could be used if the Web client itself supports CORBA, as Netscape said will happen with the next version of Navigator (Microsoft will propose something similar for sure).
So let us assume the application uses Java at the client side. The client application is a Java applet being executed in the context of the Web client virtual machine. In order to access its server-side application via CORBA, the applet either includes CORBA itself or relies on CORBA support from the Web browser as we discussed in the previous paragraph. At the time this paper was written, there was no commercially available Web browser supporting CORBA. There are, however, many Java-based CORBA products such as JavaSoft's JOE, PostModern Computing's Black Widow, or Iona's OrbixWeb.
At the server side, the application provides a set of services accessible via its CORBA interface. There are many CORBA-compliant products available commercially today, so in principle any one of these products could be used. However, if the application is designed to support services to client applications using other CORBA products, than it should be based on the standard IIOP protocol for interoperability between different CORBA implementations.

4.2 Based on Java

CORBA was designed to support communication between applications by taking advantage of the best technology that existed in the early 90s. It has been widely implemented and used in a large number of industrial, mission-critical distributed applications. As a result, CORBA is praised by its virtues in the distributed systems community, with entire conferences on the latest CORBA products. However, there is an interesting argument posed by the Java community regarding the CORBA approach, as follows.

- Java represents a newer (read better) technology and was designed, like CORBA, for building distributed applications, so why use CORBA if we now have Java?
- In addition, Java is already supported by most Web clients and it has built-in support for inter-process communication, so why support CORBA in addition to Java?

The CORBA people argue that CORBA is a proved technology. According to them, Java can be an interesting programming language (like Tcl, Smalltalk, or Obliq) but "real programs" are written in "real languages" (like C or perhaps C++). Also, there is nothing in Java like the world-wide distributed applications built with CORBA. Furthermore, the CORBA interface is (at least, in theory) language independent and so CORBA can be used by any programming language, including Java. For example, an existing server application written in C can be used by a Web client application written in Java provided both use CORBA.
Here we are interested on approaches for building Web applications from a technological point of view. In this paper it does not matter if Java or CORBA will eventually succeed (or maybe both) since the issue here is how to build Web applications based on a distributed infrastructure using Java and compare the result with CORBA.

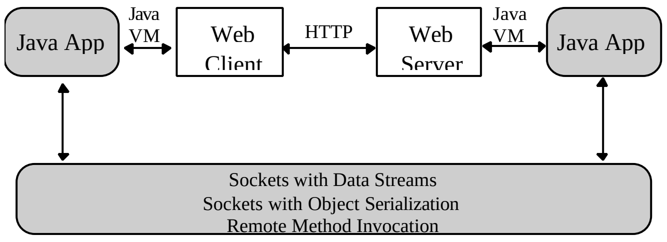


Figure 10: WIS computational model supported by Java based infrastructure.

Figure 10 depicts the distributed infrastructure approach based on Java. At the client side, the application consists of a Java applet that is executed by the Web client virtual machine. Beyond its basic functionality such as end-user interaction, it can invoke services provided by another Java application at the server side. The communication between the client and the server applications is supported and managed by a distributed infrastructure based on Java.

The most popular approaches for communication between two Java processes is by means of raw sockets or the Remote Method Invocation (RMI) protocol [Jav96a]. RMI is a proprietary (just for Java) CORBA-like RPC mechanism for calling methods on remote objects. The arguments are passed by copy using Object Serialization, a mechanism that can be used independently, e.g. for storing objects on files or copying them via sockets to another process.

If the Web server is itself written in Java – such as the Java Web Server from SunSoft and many others – then programmers have enormous opportunities for taking advantage of increased efficiency, integration and cooperation between the Web server and the sever-side application. For example, SunSoft is promoting this integration with its idea of servlets, Java programs that can be used to extend the basic Web server functionality. Further integration can be achieved if the Web client is also written in Java, such as HotJava from SunSoft.

5. Summary

In this paper we presented the basic technological approaches for designing and building Web information systems. Two complementary approaches were identified: WIS whose activity is mainly executed at the server side (Server-centric WIS); and WIS whose activity is mainly executed at the client side (Client-centric WIS). Furthermore, we identified a third approach, that integrates the previous two, based on a distributed infrastructure.

All these approaches are based on existing technology, sometimes widely used in other contexts, and as a result there has been an enormous development activity all over the world building Web information systems. There are already a number of mission-critical WIS, especially in restricted contexts – often inside an organization or a dispersed but well defined community. Examples include: collaborative work intra- and inter-organizations; public information kiosks; support for the sales people; and document storage and retrieval, including digital libraries.

However, some limitations are now becoming visible. For example, server-centric WIS typically offer a *poor interaction with the end-user* based on forms; *have difficulties supporting complex transactions* like reliable debit-credit operations; and *present a low-level performance* due to its centralized architecture and an inefficient HTTP protocol.

Client-centric WIS were developed to eliminate these problems, but also raised new issues regarding downloading time, version maintenance, security and safety.

Although the new distributed approach already permits to build WIS applications with a quality similar to traditional client/server applications, it raises new problems regarding the large number of proprietary products being sold by many vendors in a highly competitive market.

Some of the questions that are now being posed include: Is it possible to build high-quality WIS using only standard technologies and products like HTML and Java? If not, is it possible to achieve a reasonable level of interoperability amongst WIS build with different technologies? We hope that Java is the answer to all these questions, but surely only the market knows what will happen.

References

[AG96] K. Arnold, J. Gosling. The Java Series - The Java Programming Language. Addison-Wesley Publishing, 1996.

[AM95] C. Ang, D. Martin. *Constituent Component Interface ++*. Centre for Knowledge Management, University of California, 1995.
`http://www.eolas.com/eolas/ccippapi.htm`

[BN96] M. Brown, M. Najork. Distributed Active Objects. Fifth International WWW Conference, Paris, *Computer Networks and ISDN Systems*. 28 (7-11), 1996.

[BTV96] J. Baumann, C. Tschudin, J. Vitek, editors. *Proceedings of the 2nd ECOOP Workshop on Mobile Object Systems (Linz, Austria).* Dpunkt, 1996.

[Car95] L. Cardelli. Obliq: a Language with distributed scope. *Digital White Paper*, Digital Equipment Corporation, Systems Research Center, 1995.
`http://www.research.digital.com/SRC/Obliq/Obliq.ps`

[Con95] D. Connoly. *Mobile Code - Issues on the Development of Hypermedia Applications*. W3C, 1995.
`http://www.w3.org/pub/WWW/MobileCode`

[DSCOB95] J. Delgado, N. Santos, P. Caetano, L. Osório, J. Barros. Extensão das capacidades de um cliente WWW com um sistema telemático multimédia. In *Proceedings of the 1st WWW Portuguese Conference*, Braga, Portugal, 1995.

[Jav96a] JavaSoft, a Sun Microsystems Business. *Java IDL*. 1996
`http://splash.javasoft.com/JavaIDL-alpha2.0/`

[Jav96b] JavaSoft, a Sun Microsystems Business. *Java Beans: A Component Architecture for Java.* 1996
`http://splash.javasoft.com/beans/WhitePaper.html`

[LSW95] S. Lucco, O. Sharp, R. Wahbe. Omniware: a Universal Substrate for Web Programming. Fourth International WWW Conference, Boston, In *WWW Journal*. W3C, Dec. 1995.
`http://www.w3.org/pub/Conferences/WWW4/Papers/165/`

[Mic96a] Microsoft Corporation. *IDC - Internet Database Connector.*
`http://www.microsoft.com/`

[Mic96b] Microsoft Corporation. *What is Active-X?* July 1996.
`http://www.microsoft.com/activex/actx-gen/awhatis.htm`

[Mic96c] Microsoft Corporation. *Visual Basic Scripting Edition.* 1996.
`http://www.microsoft.com/vbscript/vbsmain.htm`

[NCSA95a] NCSA, University of Illinois. *Server Side Includes (SSI)*. 1995.
`http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html`

[NCSA95b] NCSA, University of Illinois. *Common Client Interface Protocol Specification*. 1995.
`http://yahoo.ncsa.uiuc.edu/mosaic/cci.spec.html`

[MdS96] M. Mira da Silva. *Models of Higher-order, Type-safe, Distributed Computation over Autonomous Persistent Object Stores*. PhD Thesis, University of Glasgow, 1996.

[MdSS97] M. Mira da Silva, A. Silva. Insisting on Persistent Mobile Agent Systems with an Example Application Area. In *Proceedings of the Mobile Agent Workshop, Berlin, Germany, 1997.*

[Net95a] Netscape Communications. *Netscape Plug-ins*. 1995.
`http://home.netscape.com/eng/mozzila/2.0/handbook/plugins`

[Net95b] Netscape Communications. *JavaScript Authoring Guide*. 1995.
`http://home.netscape.com/comprod/products/navigator/version_2.0/script/script-info/index.html`

[Ous94] J. Ousterhout. *Tcl and Tk Toolkit*. Addison-Wesley Publishing, 1994.

[Ques96] Questar Microsystems Inc. *Advanced HTTP Server Side Functionality with SSI+*. 1996.

[Rob96] D. Robinson, *The WWW Common Gateway Interface Version 1.1*. Internet Draft, 1996.
`http://www.ast.cam.ac.uk/~drtr/cgi-spec.html`

[SAD96] A. Silva, G. Andrade, J. Delgado. A multimedia database supporting a generic computer based quality management system. In *Proceedings of the 9th ERCIM Database Research Group Workshop*, Darmstadt, Germany,1996.
`http://bruxelas.inesc.pt/~alb/papers/edrg95.ps.gz`

[SBD95] A. Silva, J. Borbinha, J. Delgado. Organizational management system in an heterogeneous environment - A WWW case study. In *Proceedings of the IFIP working conference on information systems development for decentralized organizations*, Trondheim, Norway, 1995.
`http://bruxelas.inesc.pt/~alb/papers/ofw3.ps.gz`

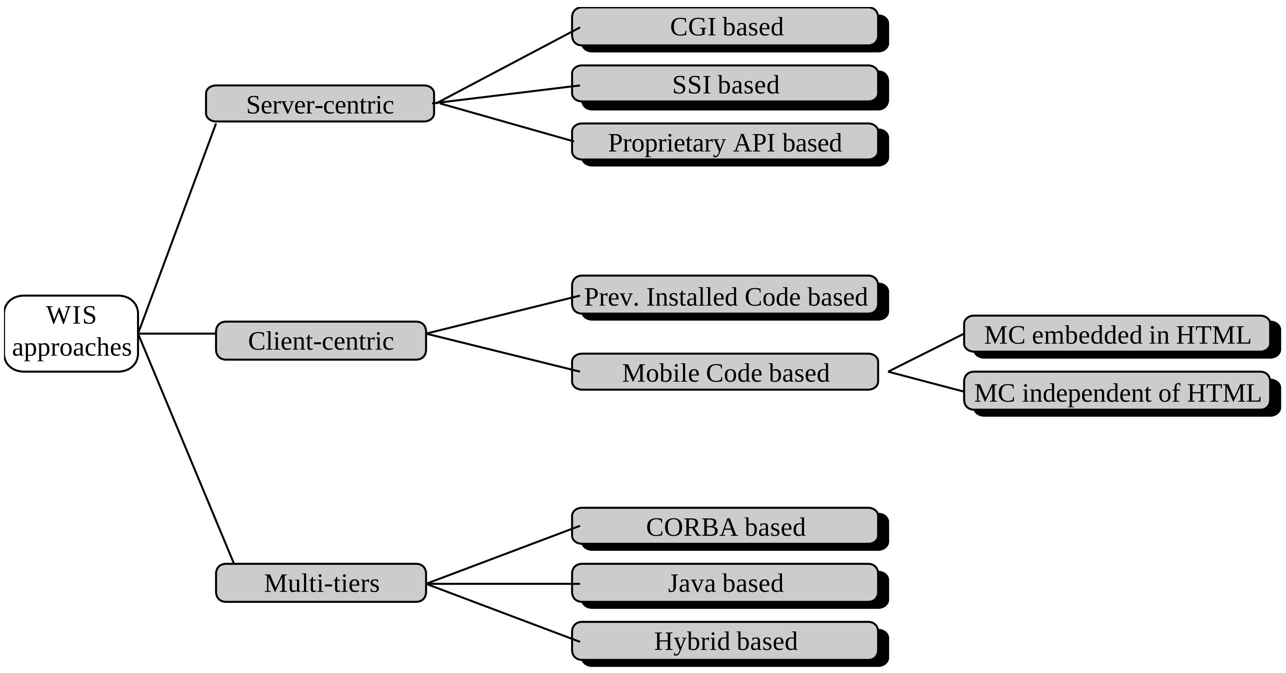
[Sie96] J. Spiegel. CORBA Fundamentals and Programming. Wiley, 1996.
`See also http://www.omg.org/corfun/corfunhp.htm`

[SMdSD97] A. Silva, M. Mira da Silva, J. Delgado. AgentSpace: Motivation and Requirements for the AgentSpace: A Framework for Developing Agent Programming Systems. *Proceedings of the Fourth International Conference on Intelligence in Services and Networks* (IS&N'97). Cernobbio, Italy, 1997.

[TB96] P. Thistlewaite, S. Ball. Active Forms. Fifth International WWW Conference, Paris, *Computer Networks and ISDN Systems*. 28 (7-11), 1996.

[Yu96] N. Yu. *AWT Tutorial*. University of Alberta, 1996.
`http://ugweb.cs.ualberta.ca/~nelson/javaAWT.Tutorial.html`

Annex: Schematic hierarchy of the identified WIS approaches.


WIS approaches
Server-centric
CGI based
SSI based
Proprietary API based
Client-centric
Prev. Installed Code based
Mobile Code based
MC embedded in HTML
MC independent of HTML
Multi-tiers
CORBA based
Java based
Hybrid based