Conceptual Frameworks for Web Information Systems Development

Alberto Silva*, Miguel Mira da Silva+, José Delgado'

Abstract

The main objective of this paper is to survey, in a schematic and generic way, the major models and approaches concerning the construction and execution of Web information systems. Basically we found three different approaches: server-centric, client-centric and distributed infrastructure-centric applications. We discuss the major strengths and weaknesses of the current Web technology, in this context, and we depict what models and technology of the distributed and dynamic information systems should exist in a 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, or information system (IS), eventually accessible to a world wide scale.

Nowadays, 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 commercial organizations. Due to this pressure situation, it *is not* the aim of this paper to describe specific tools, environments, or products concerning WIS development because they would be outdated quickly by new generations of homologous and competitive products.

The main objective of this paper *is* to describe, in a schematic and generic way, the *main models or approaches concerning the WIS construction*. Consequently, we discuss the major strengths and weaknesses of the current Web technology and we depict what models and technology of the distributed and dynamic WIS of the future should exist.

This paper is a result of our real-world experience designing and developing large distributed applications, as well as our academic and research interests. 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 [MdSA96, MdS96], 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 IS whose activity consists mainly of one or more processes running on the server machine(s). Section 3 presents the client-centric WIS approach, meaning the class of IS whose activity is executed mainly on client machine(s). In section 4 we describe a new approach of WIS that is based on distributed infrastructures and where the activity is effectively divided between client and server machines. In section 5 we discuss what we guess will be the IS of the future, namely in the Internet and Intranet contexts. Lastly, section 6 summarizes the

* Alberto.Silva@inesc.pt, INESC / Tecnical University of Lisbon

+ mms@dmat.uevora.pt, University of Évora

' Jose.Delgado@inesc.pt, INESC / Tecnical University of Lisbon

presented approaches and points what we intended to in a near future.

2 Server-Centric WIS

2.1 CGI

The first WIS were based on the CGI (*Common Gateway Interface*) [Rob96] which consists in a generic and simple specification. CGI defines a communication interface between a Web server and any specific process that would run in the same computer architecture. It is a flexible and extensible way to add functionality to Web servers, such as data bases access, data and protocol conversions, etc.

After the construction of the first real WIS based on the CGI, some variants happened, either for simplicity reasons (Server Side Includes mechanism) or due to performance stress (Web server proprietary APIs). Nevertheless, all server-centric IS share the following common issues:

- Because Web servers are stateless and the HTTP protocol is connectionless, WIS are called "short-life" processes and should be responsible for the maintenance of successive user interactions. There are some known techniques of keeping and simulating the state between different interactions (accesses) in a common session, 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 documents generators (which may be more trivial or more complex). This means that the main function of any WIS is to parse a set of information sent by the Web client component, and produce at run-time a convenient response, which is in general in HTML format [SBD95].

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

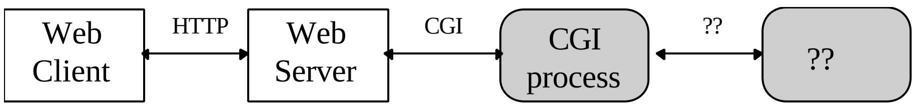

Figure 1: WIS computational model based on the CGI approach.

2.2 SSI

The Server Side Includes (SSI) mechanism was originally introduced on the NCSA's Web server [NCSA95a]. Figure 2 depicts its generic functionality.

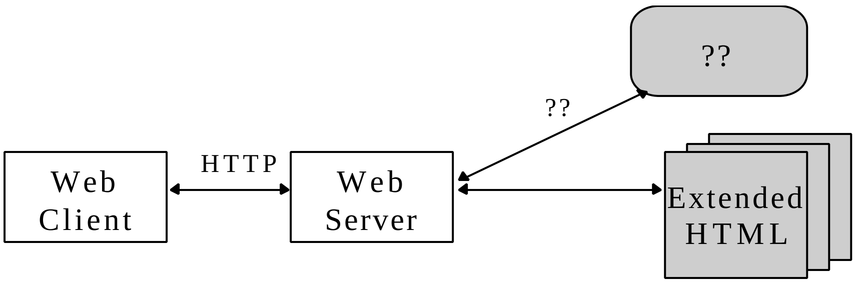

Figure 2: WIS computational model based on the SSI approach.

The application is basically composed by a given set of HTML files with some extended elements (*tags*). When the Web server receives a request for some extended HTML document, it parses the corresponding file and solves the semantic associated to every extended element in order to convert this file into a new one with just “standard” HTML elements. To solve all 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.

SSI was originally conceived and adopted by the NCSA Web server. However, it has inspired some equivalent mechanisms but more flexible and skillful, such as Microsoft’s Internet Database Connector [Mic96a], or WebQuest’ SSI++ [Ques96]). Nevertheless, its main goal is to enable the development of simple WIS in an easy (even without knowing any programming language) and fast way.

2.3 Proprietary Web Server API

In this approach WIS use Web servers application programming interface. This means that they are loaded in the same memory space of the Web server. The load memory operation is executed at the server boot time or by demand each time they would be invoked by any client request. This approach doesn’t avoid the “short-life cycle” presented on the server-centric WIS, but eliminates, for each access, a new creation process with its corresponding memory loading. It is just required, from the operating system point of view, the support to dynamic linking executable modules. Figure 3 depicts the computational model corresponding to this approach.

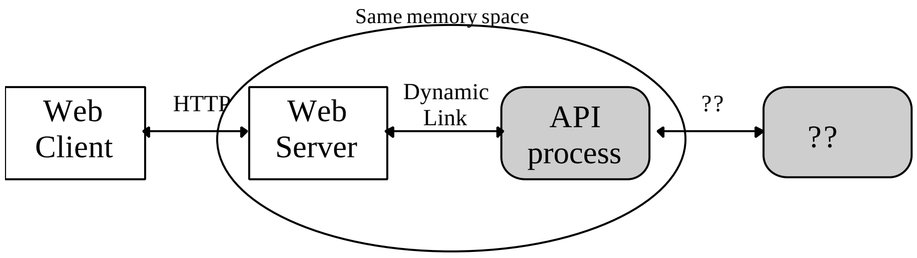

Figure 3: WIS computational model based on the proprietary Web server API approach.

Currently, it is usual that every Web server provides its own proprietary API, for example: Netscape’s NSAPI (*Netscape API*), Microsoft and Process Software’s ISAPI (*Internet Server API*), Oracle’s WRBAPI (*Web Request Broker API*), etc.

2.4 Comparative Analysis

In this section we summarize and present comparatively CGI, SSI and API. Obviously, the opinions presented on Table 1 (as well as Table 2) are our own responsibility and mean a global and generic classification.

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 following criteria are analyzed: runtime performance; scalability (the ability to support a increased number of request and connections); safety or robustness (if a malfunction occurs in the application and that doesn't imply the server crash); application flexibility and versatility; easy development and technical skills required; application management and maintenance; and open (vendor-neutral) or proprietary solution. The aim of this analysis is to enable a deep reflection of the strengths and weaknesses of the different mechanisms.

Applications based on CGI have a low level performance due to their inherent deficient management of processes and memory. They may support scalability via replication of CGI processes on different machines (This implies the existence of some special load balancing algorithms in the code of this CGI application activated in the moment of the HTML documents generation - in order to generate the URL corresponding to the least loaded machine). Since applications are external to Web servers this approach presents a high safety level, meaning that if the process crashes, it doesn't imply the corresponding Web server crash. Since these applications can be developed independently of the Web server, using any conventional language (e.g., C, C++, Perl, Basic, Java) and development environment (e.g., VB, Delphi, WebObject, PowerBuilder), they could be very flexible and even complex. The development and test cycle is moderated - depending directly on the capabilities present in the chosen language/development environment and the existence, or not, of specialized software libraries supporting the CGI particularities: state maintenance; HTML code generation from DBMS; etc. Technical skill requirements, application management and maintenance is medium/high - depending directly on the same characteristics previously referred to. Due to its simplicity and consequent support and adoption by the generality of Web servers, CGI is the *de facto* standard in the development of WIS.

SSI is basically a substitution mechanism, where each extended HTML element is converted at "request-time" to a corresponding set of standard HTML elements, either by the result of a system call (e.g., *time()*), or the result of a SQL query. So, SSI presents a medium/high performance. Its main drawback consists in the limited capabilities concerning the development of flexible and complex applications. Due to these characteristics scalability is not well supported. If the resolution of some extended HTML element crash that implies the crash of the corresponding Web server. Usually each vendor defines its own set of extended elements, consequently it becomes hard to develop an independent-vendor application. Nevertheless, its corresponding development is easy, fast and it is easy to change and maintain the application.

Lastly, *applications based on property Web server API* can be flexible and complex, and present high/medium level performance. Due to this flexibility, it is possible to develop the same kind of algorithms as for the CGI approach, in order to support some scalability degree. Because applications are executed in the some Web server's memory space, this implies the Web server crash when a crash in the application occurs (However, there are already some technological solutions to this problem. For instance Oracle's Web server prevents that problem by providing separated process memory spaces). Lastly, they require high level technical skills which imply a high cost development and high risk management and maintenance.

3 Client-Centric WIS

After the construction of the first real-world server-centric WIS, their main limitations became apparent: difficult to support complex or long transactions; low level global performance; and also a low level end-user interactivity. Client-centric WIS approach looks for (partial) solutions of these limitations. This approach may be divided into two distinct groups. The former, consists on applications based on *previous installed code*. The latter group represents applications based on *mobile code*. The distinction between these two groups is not always clear, rather it is subjected to discussion and controversy such as the cases of Microsoft's Active-X [Mic96b] or Colusa Software's Omniware [LSW95] technologies.

3.1 Previous Installed Code

In order to provide a better cohesion and integration between Web client and specific applications, some protocol proposals were done originally: CCI (Common Client Interface) [NCSA95b] and CCI++ (Constituent Component Interface ++) [AM95]. Netscape, based on goals and capabilities present on CCI++ technology, provides an open API that enables third-parties to develop external and independent applications that can run tightly into his Web client.

These applications are designated by *plug-ins* [Net95a] and are dynamic link executable modules. They are previously installed in the client computer, and are responsible for the handling of one or more multimedia documents (MIME types). They are invoked, loaded in memory and run dynamically by, and in, the computational context of the Web client whenever it receives a corresponding MIME document.

Figure 4 emphasis (gray color) the following relevant components: the application previously installed; the MIME documents; and some application, executed on the server side, responsible by the MIME documents generation (e.g., file server, specific CGI application).

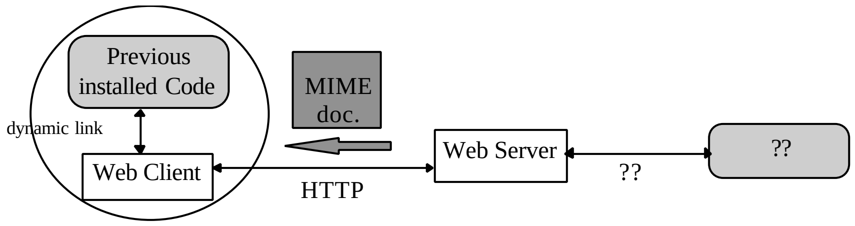

Figure 4: WIS computational model based on previous installed code approach.

In general, applications that adopt this approach are relatively generic, complex and produced by specialized software houses. Usual examples are viewers, players and editors of know multimedia information formats, such as Acrobate, VRML, Director, MPEG2, etc. Other examples are spreadsheets editors and proprietary virtual machines such as Microsoft's Java Virtual Machine or Colusa Software's Omniware Virtual Machine (both of them for the Netscape Web client).

3.2 Mobile Code

In spite of the plug-in approach allowing a reasonable cohesion and integration between Web client and external applications, this approach presents some limitations, such as application versatility and portability. Additionally, it requires a previous and (human) installation, which implies a human effort concerning the version maintenance task.

A more general development approach is based on mobile code [Con95,BTV96]. Mobile code is a program (built in any programming language) that is stored in some computer but which can be transportable, through a set of network(s), to another one in order to be executed there. Though mobile code concept doesn't imply any specific language, its implementation is usual on interpreted (scripting) languages due to its portability and flexibility characteristics, in an obvious loss of performance.

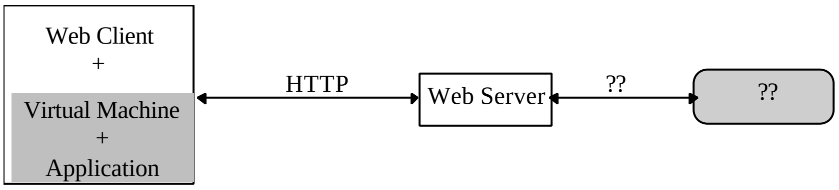

Figure 5: WIS computational model based on mobile code approach.

The specific application code (vid. Figure 5) is managed in the Web server machine -'??' block -, and is transferred, on demand, to the Web client machine where it is executed - "virtual machine" block". The client provides a machine responsible for the safe interpretation/execution of the received code. In some cases it is a virtual machine, in others, it is a real machine.

We found basically two distinct WIS approaches based on mobile code: mobile code embedding in HTML documents; and mobile code independent of HTML documents.

In the *mobile code embedding in HTML documents approach*, the Web client, beyond its regular capabilities of parsing and rendering HTML elements, has also to have capabilities of interpreting and executing (text) embedding code. This approach is adopted by the majority of scripting languages, such as JavaScript [Net95b], VBScript [Mic96c], Tcl [Ous94], Obliq [Car95], etc. This approach has at least two main goals. First, to allow the construction of small and simple WIS, with reasonable end-user interactivity level. Second, to allow the convenient integration of HTML documents with Java applets, plug-ins, etc., so, acting as a glue technology.

In the *mobile code independent of HTML documents approach*, there is a virtual machine that executes the code referenced in some HTML document, but which is located in a independent file (managed and saved on the server machine). Consequently, the code can be referenced several times inside the same HTML document, and among several documents. The code is transferred and executed in text or binary format (vid. Figure 6). This is the approach adopted by Java applets [AG96] and Active-X controls. There are also other research experiences, following this model, based on languages referred to above, such as Tcl [TB96] and Obliq [BN96].

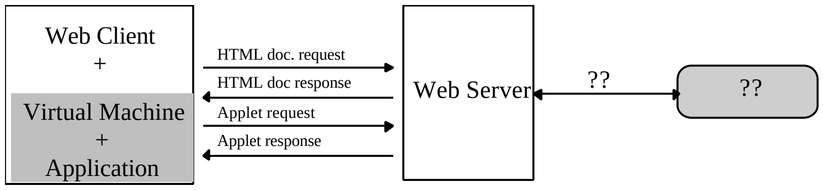

Figure 6: Mobile code, HTML document independent.

3.3 Comparative Analysis

In this section we summarize and present comparatively the main client-centric WIS models. Beyond the criteria analyzed on section 2.4, the security issue is additionally analyzed.

Analysis criteria	Prev. Inst. Code	M.C. embed. in HTML	M.C. indep. of HTML
Application performance	☺	☹	😐
Application safety	☹	😐	😐
Application flexibility	☺	☹	😐
Easy development	☹	☺	☹
Security	☺	😐	😐
Application maintenance	☹	😐	☺
Vendor-neutral	☹	😐	☺

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

Applications based on *code previously installed* and configured present high level performance due to at least two reasons. First, the application code is already in the client machine (eventually it is already loaded in memory) when it is invoked. Second, the code is compiled to the corresponding client architecture. So, the code is directly executed by a real machine. Because the application shares the same memory space of Web client, implies some safety problems. This approach doesn't raise any security problem - the application is either reliable or it is not. On the other hand, it requires intense human labor with version maintenance and management; and requires high level technical knowledge in order to develop complex and specialized applications. Lastly, this approach is based on proprietary Web client API, so there is a close dependence with some specific vendor.

Applications based on *mobile code embedded in HTML documents* present, depending on their complexity, a low/medium level performance due to two reasons. First, the fact that the code is transferred from server to client machine. Second, the fact that the code is interpreted in a little efficient (text-based) format. The original languages on which they are based should be simple to interpret and short enough. Furthermore, they should restrict themselves due to security reasons. For example, the prohibition to access the (client) file system is normal, or to establish connections with server processes in different source machines. However, there aren't any good solutions, supported by virtual machines, for the resource (memory and CPU) exhaustion syndrome. Consequently, these applications should be restrictive enough and are not very flexible. On the other hand, their version maintenance and management is easily done. (we don't give them the highest mark because small changes in an application may eventually require several changes in several HTML documents).

Applications based on *mobile code independent of HTML documents* present, depending on the application and the adopted technology, a medium/low level performance basically because the code should be transferred and interpreted. Unlike the previous model, the code is usually interpreted in a more efficient format - virtual machine byte code. There are other solutions to improve the performance of these applications based on Java applets: just-in-time compilation, or even interpretation by the real machine (Network Computer approach). 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, security issues are raised for which basic solutions and problems were focused above (relating to mobile code embedded in HTML documents). Consequently, these applications present a medium level in flexibility and versatility. Lastly, in spite of several announcement about integrated and visual development environment, this approach requires a high level of technical skills. Not only of the language itself, but also and mainly, due to several libraries of classes, standard and non standard, that are going to emerge, and that the programmer must dominate. Lastly, due to Java success and its largely support by the majority of vendors, applications based on it are (in general) vendor-neutral.

We put here a special note about the Active-X technology because, from our point of view and in relation to the shown models, it is a hybrid approach. From the portability and flexibility perspective, Active-X controls are similar to plug-ins. On the other hand, from the mobility (remote transfer) and automatic configuration perspective, they are equivalent to Java applets. Consequently, applications based on Active-X controls present high level performance (the code was previously compiled to a specific architecture), requiring the maintenance and management of all controls used for each platform and support operating system; and raising several security problems.

4 WIS Based on Distributed Infrastructures

The client-centric WIS model allows the reduction or (partial) solution of the problems related to performance and end-user interactivity present on server-centric WIS approach. However, it doesn't give an adequate answer to complex transactions deficiently supported by the HTTP protocol. Because the HTTP protocol is based on the "request-response" communication pattern , it doesn't adapt well to some class of services, namely those involving live multimedia interactions (e.g., real-time video or audio applications) or data bases systems requiring complex, long and eventually distributed transactions.

Consequently, some *combined solutions* should be the convenient answer. In this approach (vid. Figure 7), the Web client and Web server are just used in the initial interaction, only to establish the connection and eventually to transfer the corresponding (mobile code) application. Next, the client application establishes a connection with the associated proprietary server. So, in that moment, the application lost the effective contact with the Web (the Web client is used, but just due to its virtual machine capabilities).

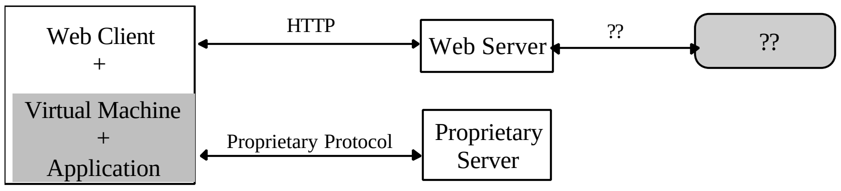

Figure 7: WIS computational model based on combined approaches.

In order to avoid the integration and interoperability difficulty raised by this approach - where specialized clients and servers agree, in a bilateral way, a proprietary communication protocol - a new approach should be conceived based on generic and distributed communication infrastructures.

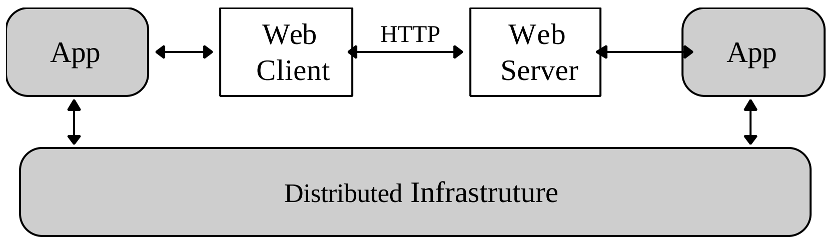

Figure 8: WIS computational model supported by a common infrastructure.

The basic idea showed in Figure 8 is that the Web technology (as it is known currently) is just used following two vectors. First, it is used as a service and resource uniform access standard mechanism. Second, as a consistent human machine interaction mechanism (e.g., HTML or Java AWT [Yu96] based). The real application activity is divided effectively between client and server applications. All of them communicating transparently through a communication infrastructure.

4.1 Based on CORBA

According to its aims and technical state, *Common Object Request Broker Architecture* (CORBA) [Spi96] seems to be a likely solution to the presented model in Figure 8. Figure 9 depicts the CORBA approach based on technology emerging, namely associated with Java systems.

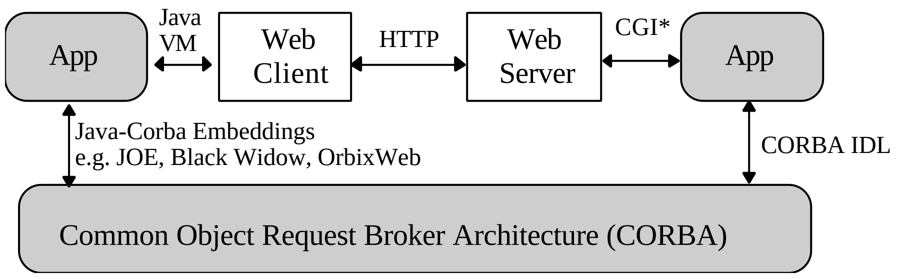
App
Java VM
Web Client
HTTP
Web Server
CGI*
App
Java-Corba Embeddings
e.g. JOE, Black Widow, OrbixWeb
CORBA IDL
Common Object Request Broker Architecture (CORBA)

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

From the client side, the application is a Java applet and is executed in the context of the Web client virtual machine. However, this applet should embed CORBA IDL to invoke (transparently) other services provided by one or more applications accessible in the CORBA space. JavaSoft's (a subsidiary of SunSoft) JOE, PostModern Computing's Black Widow, or Iona's OrbixWeb are some commercial products that are (or would be) supporting this approach.

From the server side, there may exist an application with CORBA IDL embedding code (namely based on Internet Inter-ORB Protocol (IIOP)). Consequently, this application may invoke or provide specific services to other disparate and independent applications.

4.2 Based on Java

Despite CORBA goals and its adequacy to support interoperability among a set of distributed applications in the Web context, some people doubt that approach. The main questions raised are the following. If Java is already supporting interoperability because it provides virtual (or even real, such as on Network Computer architectures) computational environments to a large set of disparate platforms and operating systems, because it presents an interpreted (with easy code mobility) language, and it is network aware, why use CORBA instead of Java? And why use CORBA IDL embedding in Java applets?

Concerning the first question, it may be argued that CORBA technology is more complete, mature and robust than the equivalent based on Java. Furthermore, CORBA IDL is theoretically language independent, i.e., it may be adopted by different programming languages, such as C, C++, Smalltalk, Java, etc.

Relatively to the second issue and supposing that Java based communication infrastructures had an enormous adhesion and success, there would be always need to support legacy applications based on CORBA, so there would be a good meaning to exist CORBA IDL in the Java system.

Figure 10 depicts the Java based infrastructure approach. From the client side, the application consists in some Java applets, which are executed by the Web client virtual machine. Beyond their basic functionality such as end-user interaction, they could invoke services provided by other Java applications (or Java embedding applications) supported and managed by distributed infrastructures.

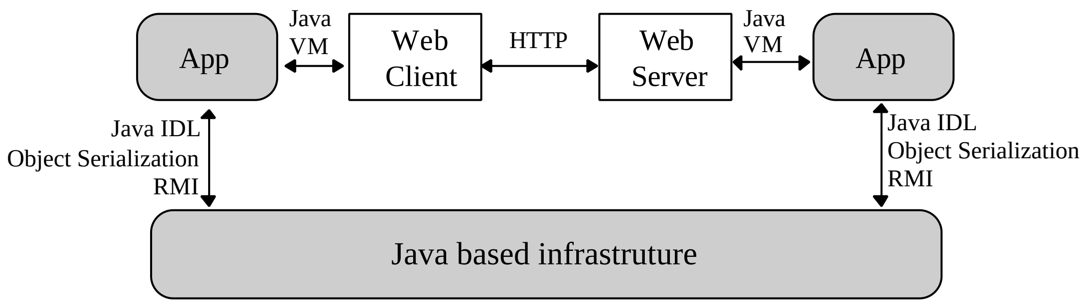

Figure 10: WIS computational model supported by Java based infrastructure.

From the server side point of view, if the Web server was itself a Java application, then the process management and process communication between the Web server and server-centric application ought to be more coherent, efficient and integrated. Currently the best announced technologies are Java IDL, Object Serialization and Remote Method Invocation (RMI) [Jav96a].

This novel approach, due to and based on Java inherent characteristics, would enable and support an emerging computational paradigm which are yet misunderstanding called mobile objects systems, multi-agents systems, coordination systems, etc. PageSpace [CTV96], Mole [SBH96], and Aglet [IBM96] are some initial research projects involved.

5 Discussion and Expectations

Nowadays, there have been an enormous effort and activity all over the world related to the Web information systems development. There are already several WIS running in the context of some Internet community. Application areas are disparate such as: collaborative work; public information kiosks; digital libraries; organization’s document systems; etc. Though there have already been several real and operational WIS, their limitations are easily visible. It is important to note that the WIS technical and experimental state of art is still in a childhood phase.

Server-centric approach had as its main advantage the fact of has allowed the Web paradigm (originally as a distributed hypermedia system) to become an “umbrella” system to a disparate number of data repositories and IS. Nevertheless, WIS based on this approach present some technical drawbacks: difficult to support complex transactions; low level performance; and a poor end-user interactivity. Client-centric aimed to attenuate or eliminate these two last previous limitations.

Though these two combined approaches allow WIS construction with a reasonable quality level, they raise new problems in part due to the existence of disparate technology promoted by different vendors in a highly competitive environment. Some of these questions are: Would it be possible achieve a reasonable level of interoperability among different WIS build with different technologies? Would it be possible WIS development in a open and vendor independent environment? What about user-interface standards?

We think that the previous depicted problems as well as the approaches and technologies presented in this paper, will be associated with two basic visions: Intranets and the Internet of the future.

The former - Intranets - will be a natural evolution of the current state of art of the relatively mature client-server technology [Lew95] together with the server-centric and client-centric WIS approaches. These combined efforts will provide better organization’s IS or even federated IS involved a limited set of organizations. This vision is already being achieved by the major IT vendors. For example Java Beans [Jav96b], with the key industry leaders commitment, should be a first real step in that direction.

The latter, the vision about the Internet of the future (which should be directly involved with the *Information Society* and *National Information Infrastructures* concepts) will change again the way people interact with technology. Instead of using basic (and primitives) tools such as FTP, News, Archie clients, the end-user would use one or more specialized software applications - *agents* - who on his behalf, in order to achieve some required tasks. Consequently, the Internet should become an *open agent space*, where disparate agents could live, negotiate, sell and buy goods, mediate or broker other agents, solve collaboratively complex tasks, etc.

From the current Internet state, that we call a static space with manual (human based) main utilization, we foresee the future Internet as a dynamic and complex space with more automatic utilization and interaction among a disparate and unknown number of specialized agents.

From the organizations point of view, they will provide, beyond their e-mail addresses and Web sites information, specific agents with well defined goals and behavior. These agents should be responsible for promoting and distributing other (user specific) agents; selling services; buying money; managing its private network load balancing; etc.

From the end users point of view, they will have (as they have today a Web home page and a e-mail mailbox) one or more agents, each of them specialized and customized in a great range of potential activities: information retrieval; smart e-mail reader; bank account management; buy and sell products; public administration bureaucracies; etc.

This vision requires several new challenges and paradigms, from a wide range of technologies such as: accounting mechanisms, agent language representation and communication, security mechanisms, etc.

6 Summary and Future Work

We presented in a schematic and generic way the basic technological approaches involved with Web information system development. Two complementary approaches were identified. First, IS whose activity is mainly executed on server machines (server-centric). Second, IS whose activity is mainly executed on client machines (client-centric). The right integration of these two approaches in a open and flexible way will be (in the next months) on major industrial challenge. Furthermore, we identified a third approach based on distributed communication infrastructures. This model is currently in a intense research phase and raises several new problems in the Internet context. It is tackled by two distinct research communities. The distributed artificial intelligent community who has introduced the intelligent agent concept [GK95,Nwa96]. On the other hand, the distributed system community, that has introduced the distributed object infrastructures concept [Sie96,Lew95].

We are involved in the conception and development of wide scale distributed IS that support many (hundreds to hundreds of thousands) end-users in a non cohesive an heterogeneous environment. Some applications of this class will be the electronic commerce, virtual organizations and electronic public administrations of the XXI century. In that way, we are currently researching and discussing mobile code frameworks [SMdSD97, MdSS97] in order to understand and propose a better support to this class of novel applications.

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`

[CTV96] P. Ciancarini, R. Tolksdorf, F. Vitali. PageSpace: An architecture to coordinate distribute applications on the web. *Computer Networks and ISDN Systems*. 28, 941-952, 1996.

[DSCOB95] J. Delgado, N. Santos, P. Caetano, L. Osório, J. Barros. Extensão das capacidades de um cliente WWW com um sistema telemático multimédia. *Comunicações da 1ª Conferência Nacional de WWW*, Braga, Julho 1995.

[GK95] M. Genesereth, S. Ketchpel. *Software Agents*. Standford University, 1995.
`http://logic.standford.edu/sharing/papers/agents.ps`

[IBM96] IBM Tokyo Research Laboratory. The Aglets Workbench: Programming Mobile Agents in Java, 1996.
`http://www.ibm.co.jp/trl/projects/aglets/about.html`

[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`

[Lew95] T. Lewis. *Client/Server Yellow Pages*. Prentice Hall, 1995.

[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`

[MdSA96] M. Mira da Silva, M. Atkinson. Combining mobile agents with persistent systems: Oportunities and challanges. In Baumann et al. [BTV96].

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

[MdSS97] M. Mira da Silva, A. Silva. Insisting on Persistent Mobile Agent Systems with an Example Application Area. Submitted to the Mobile Agent Workshop 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`

[Nwa96] H. Nwana. Software Agents: an Overview. *Knowledge Engineering Review,* 11(3), 1-40, Sept. 1996.

[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`

[SBH96] M. Strasser, J. Baumann, F. Hohl. Mole - a Java based mobile agent system. In Baumann et al. [BTV96].

[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: A Framework for Building Agent Programming Systems. Submitted to the Fourth International Conference on Intelligence in Services and Networks, 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`