MACROBUTTON HtmlDirect
<BODY ...>

Building Collaborative Environments with Use of Web Technologies.

Konrad Olszewski

Master thesis prepared under supervision of
prof. Czesław Jędrzejek

Syracuse September 1996

Acknowledgements

The authour would like to express thanks to professors Czesław Jedrzejek and Marek Podgórny of

Northeast Parallel Architectures Center, Syracuse NY, USA for their supervision and help, to Krzysztof Walczak, Gang Cheng and Janusz Buława for the contribution to the work presented in this document and many innovative ideas, to Łukasz Beca and Tomasz Jurga for their fruitful cooperation during development the system, to all the staff of NPAC who were extremely helpful during the research and writing of this document.
The research work presented in this document was funded by US Department of Defense grant, Rome Lab. Contract No. F30602-95-C-0273, PR No. C-5-2293/4.

Abstract

This document describes the design of a generic message passing and collaboratory system. As one of the key requirements for such system is its portability, it was designed to work in the World Wide Web environment and uses only highly versatile off the shelf components.
Newly introduced technologies, such as Java, JavaScript and plug-ins provide means to develop a completely new kind of applications that can use the Web browser not only as a display terminal, but effectively distribute the load between the client and the server. New solutions applied make in consequence possible to create complex systems using WWW as a distributed platform.
The designed system runs in Internet environment and provides means to integrate various kinds of applications into one system with unified management and message passing channels. It allows to incorporate into the system already existing stand-alone applications, as well as transported over the Internet Java applets.
The system will be used as a collaboration platform for a prototype Command and Control Center purposes, currently under development in Northeast Parallel Architectures Center, Syracuse , USA under a Department of Defense contract.

Streszczenie

Niniejszy dokument opisuje projekt generycznego systemu do pracy grupowej. Ponieważ jednym z głównych wymagań dla takiego systemu jest jego przenośność, został on zaprojektowany do działania w środowisku WWW przy użyciu charakterystycznych dla niego elementów i technologii.
Nowo wprowadzone technologie takie jak Java, JavaScript i plug-in dostarczają środki niezbędne do budowy nowego rodzaju aplikacji, wykorzystujących Web browser nie tylko jako wyświetlacz wiadomości, ale rozdzielających aplikacje na części działające tak po stronie serwera jak i klienta. Zastosowane nowe rozwiązania umożliwiają tworzenie złożonych systemów wykorzystujących WWW jako platformę komunikacyjną.
Zaprojektowany system działa w środowisku Internetu i dostarcza środków do integracji różnorodnych aplikacji w jeden system ze zintegrowanym zarządzaniem i metodami do przesyłania komunikatów między aplikacjami. System pozwala na dołączanie do niego już istniejących, pierwotnie samodzielnych aplikacji, jak również appletów napisanych w Javie transportowanych przez Internet.
System będzie użyty jako platforma do pracy grupowej w prototypowym systemie "Command and Control Center", aktualnie tworzonym w Northeast Parallel Architectures Center w Syracuse USA, finansowanym przez Departament Obrony Stanów Zjednoczonych.

Rèsumé

Ce document décrit la création d'un système génerique de transportation de messages et d'un système collaboratoire. Etant donné qu'une des principales demandés de ce système soit sa portabilité, ce système a été créé pour fonctionner dans le WWW et il utilise que des composants accessibles d'une versatilité élevée.
Des technologies novellement éntroduites, telle que Java, JavaScript et "plug-ins" offrent des moyens pour développer des nouvelles formes d'applications qui peuvent utiliser le "Web browser". Il sera utile non seulement en tant que terminus, mais aussi en tant que distributeur efficace de la charge entre le client et le serveur. L'application de nouvelles solutions renelent donc possible la creation des systèmes complexes utilisant le WWW comme plateforme de distribution.
Le système créé marche dans l'environnement Internet et offre des moyens d'intégration de différents types d'applications en un système de gèrement unifié et du système générique de transportation de messages.
Le système offre la possibilité d'incorporer des applications indépendantes dans un système déja existant, ainsi que de transporter les "applets" de Java.
Ce système sera utilisé en tant que plateforme de collaboration pour un prototype pour "Command and Control Center", en développement actuellement dans le NPAC, Syracuse, USA sous un contat du Ministère de la Défense.

Table of content

1. Introduction
2. WWW technologies
2.1 Java
2.1.1 Language description
2.1.2 JDBC
2.2 Netscape technologies
2.2.1 JavaScript
2.2.2 Plug-ins
2.2.3 Netscape LiveConnect
3. Java and WWW Environments
3.1 Habanero
3.2 WWW distributed system at Caltech
3.3 InSoft environment
3.4 Critical assessment of presented environments
4. System architecture
4.1 Dictionary
4.2 Overview MACROBUTTON HtmlDirect </P>
4.2.1MACROBUTTON HtmlDirect <P ALIGN="CENTER"> Interserver communication aspects
4.2.2 Interapplication communication
4.3 Architecture details
4.3.1 System components
4.4 Communication between applications
4.4.1 Partner mode
4.4.2 Master-slave mode
4.5 Network services
4.5.1 User authentication and security policy
4.5.2 Event logging in the system
4.6 Session management
4.6.1 Creating a session
4.6.2 Joining an existing session
4.6.3 Leaving a session
4.7 Event flow
4.7.1 Entering the system
4.7.2 Leaving the system

4.7.3 Launching local application
4.7.4 Joining a session with new application
4.7.5 Joining a session with existing application
4.7.6 Launching remote application
4.7.7 Switching to master mode
5. Communication protocol
5.1 Overview
5.2 Message structure
5.2.1 Application <-> Demon
5.2.2 Demon <-> Central Server
5.3MACROBUTTON HtmlDirect </P> Central Server - Local Demon protocol
5.4 Interfaces
5.4.1 Java interfaces
5.4.2 C interface
5.4.3 Example of interface use
5.5 Database scheme
6. Requirements for the project
6.1 Existing applications
6.1.1 Chat
6.1.2 Whiteboard
6.1.3 Visible Human
6.1.4 GIS - Geographic Information System
6.1.5 Videoconferencing system
6.1.6 Search system
6.2 Military scenario and simulation
7. Conclusions and future development
8. Bibliography

1.Introduction

During the ~~The~~ recent years one could observe an enormous growth of networks and technologies related to them. Especially Internet contributed to this area, introducing means for global computer communication. Computer using networks may now form one gigantic virtual machine used to get access to the information stored around the world. Many different methods of information exchange have been used for years, including electronic mail, gopher and file transport, to mention a few. However it is the World Wide Web technology that introduced a real revolution into computer world, not only in the field of technology, but also in the human perception. Its popularity is caused by the rich information contents, easy and intuitive interface and good availability. The hypertext concept it uses allows to present information that consists of various types of data (text, images, links to other documents). Because of its attractive contents and intuitive method of use World Wide Web gained immense acceptance. Over the years it has proven its stability and robustness. Due to its popularity, new technologies are being developed very rapidly, providing users with new capabilities such as viewing and processing new various kinds of data, e.g. sound and video.
Because of its popularity WWW started to be perceived as a common platform not only for information storage and exchange, but also for the computing in general sense. Common users wanted to migrate with their software into this new, attractive network environment. WWW provides means for effective information sharing and transportation. It would be tempting to build the software entirely based on this platform, enabling truly collaborative work. However, until recently, WWW could not provide effective tools to satisfy these demands.
Despite its acceptance in the modern computer world, WWW was very little but a convenient method of displaying information. Building collaborative frameworks was not possible, because there was no mean to provide a user with an interactive way of processing information available on the WWW. The only method of introducing some interactivity was the CGI (Common Gate Interface) mechanism. This technology however was insufficient for growing demand of users. At this stage of WWW development it was impossible to create tools for cooperative work based on this technology. This situation has changed when the new technologies like Java and plug-ins were introduced.
In the year 1995 Sun Microsystems introduced Java. Java is an object-oriented, platform-independent,

general purpose programming environment. Apart of its other features, it introduces the notion of applets. Applets are programs that can be included in HTML page, much like an image can be included. When one uses a Java-compatible browser to view a page that contains Java applet, the applet code is transferred to client's system and executed by the browser. Java gained enormous popularity among the WWW user and together with its lightweight script partner JavaScript is now broadly used to bring interactivity into Web environment.

Netscape Inc., the company shipping the most popular Internet browser Netscape Navigator, introduced successfully their concept of plug-ins. Plug-ins are software add-ons coming in the form of shared libraries that can be integrated with the browser to enhance its capabilities, e.g., handle new data types, enable new connections, allow interactivity with the user.

With all these technologies mentioned above it became possible to create interactive multiuser environments based on World Wide Web. Several attempts have been done. Netscape Inc. after acquiring InSoft Inc., shipped plug-ins enabling videoconferencing. NCSA laboratories developed Java based collaboratory system called Habanero. Another Java environment is created in California Institute of Technology. All these projects are relatively new and in present version have certain limitations. Netscape technologies are platform dependent and not versatile. Habanero system is quite promising but currently does not cooperate with Internet browser environment, so the whole code must be present locally on each client's host. It is also constrained to Java applications only. CIT system demonstrates use of Java technologies only for certain types of applications, being more a distributed computing than collabratory system.

The goal of this work was to develop a generic collaborative environment based on the World Wide Web technologies. It should provide the following new features:

- enable to integrate standalone applications providing them with unified interface to communicate with their instances on different machines
- allow to execute and control application from Internet browser environment providing easy and intuitive interface to collaborative applications
- provide means for session control (user authentication, starting and ending sessions, tracing participants, changing user privileges)
- integrate existing applications written in any programming language, assuming socket communication as only necessary integration prerequirement
- provide a possibility to download certain applications across the network through the use of Java applet technology, allowing part of the system to be distributed across the Internet
- provide database logging mechanism, so all user activities may be stored in the persistent form and retraced if necessary

This document contains the design of the environment basing on the assumptions listed above. The result of this work is the complete description of the system . The working environment based on this design is currently built in Northeast Parallel Architectures Center at Syracuse University as a part of the CIV (Collaborative and Interactive Visualisation) project.

The remainder of this document is organized as follows.

Section 2 covers several World Wide Web technologies. First the WWW is described as general phenomenon. Then Java language is described with stress on its role in WWW. Also JDBC - Java based interface to databases is presented. Paragraph 2.3 summarizes technologies provided by Netscape company. In consists of description of three particular technologies - JavaScript language, plug-ins for Netscape browser and LiveWire technology.

Section 3 presents collaboratory systems. First the general idea of such environments is discussed. The three following examples which were studied during this project illustrate the concept of collaboratory systems in WWW environment.

Section 4 consists of the description of the developed system. The general architecture is presented, followed by the definition and rationale for each of the system components. The detailed design is provided for each system part.

Section 5 discusses the protocol used in the system to provide session management and event passing. The general assumptions are discussed and the protocol messages are described in detail.

Section 6 describes sample collaboratory system based on the design presented in section 4. The requirements specific for this system are provided as well as application to be part of the target system. Porting of application into system environment is covered in detail.

Section 7 gives reader the summary of the whole project, providing market context and describing the

ongoing work and directions for evaluation of the system.

1.WWW technologies

The meaning of World Wide Web became recently ubiquitous and fuzzy. It is no longer easy to say what belong to WWW and what does not. For the purpose of this work I assume the following definition: WWW is an evolving set of technologies based on Internet network. It encompasses such entities as servers, clients, protocols used to communicate among them.
This section does not pretend to present the whole phenomenon, but rather concentrates on particular Web technologies which were used while working on this document. This involves Java language, JDBC interface to bridge Java and database systems, and a subset of Netscape technologies, namely JavaScript, plug-ins and LiveConnect.

1.1.Java

1.1.1.Language description

As far as professional programming languages go, the Java Language is simple, yet flexible and powerful. The language is object-oriented (with single inheritance), statically typed, multithreaded, dynamically linked, and has automatic garbage collection [1] [2].
Java syntax is based on C and C++, so C/C++ programmers can learn it quite easily. There is less redundancy which means developers should be able to more easily read someone else's code. For example, the Java Language has no user-defined operator overloading, as is found in C++.
Java Language gives developers the ability to use three different paradigms of programming in one language. Like the symbolic programming language Smalltalk, the Java Language is object-oriented, has dynamic linking, and has a class hierarchy with single inheritance. For numeric programming, the Java Language has platform-independent data types, array bounds-checking, and well-defined IEEE arithmetic. These capabilities provide good foundation for writing stable numerical algorithms that give repeatable results. For systems programming, expressions, statements, and operators in the Java Language are in most cases the same as in the C language.
The Java Language encourages catching bugs early, during development, before the software is released. It does this by strong data typing, automatic garbage collection, array bounds checking, lack of automatic type coercion, and the lack of the pointer data type. These safeguards help in the age of the Internet, where developers are deploying software very rapidly.
The Java Language has multithreading built in, with a strong model of how thread-critical code can be synchronized to avoid race or timing problems. With the growth of multiprocessing and a decrease in processor costs, the Java Language is poised to enable a new generation of concurrent applications and services.
Exception and thread mechanisms are integrated with the language and its type system. In addition, the language includes dynamic linking of subclasses with methods that override or add functionality at runtime. In other environments, these features have typically been arcane and complicated system services. There is a great simplicity and advantage to having these facilities in the language and therefore portable between platforms. The language also defines what binary compatibility is, by defining a class (.class) file format, which includes the instructions for the Java Virtual Machine in the form of bytecodes. The complete compilation and execution Java environment is presented in Figure 1.

Figure 1 Java environment

The Java Platform enables developers to create two different kinds of programs:
Applications are programs that do not require a browser to run - they have no built-in downloading mechanism. When an application is called, it runs. In this way, applications are just like programs in other languages. They can perform traditional desktop tasks, such as that done with a word processor, spreadsheet or graphics application. Like an applet, an application requires the Java Platform for it to run; however the Platform can be available as a separate program, can be embedded directly within the underlying operating system, or presumably can be embedded in the application itself.
Applets are programs that require a browser to run. The <applet> tag is embedded in a Web page and names the program to be run. When that page is accessed by a user, either over the Internet or corporate intranet, the applet automatically downloads from the server and runs on the client machine. Because

applets are downloaded, they tend to be designed small or modular, to avoid long download times.
The syntax of the <applet> tag is supported by the Netscape Navigator 2.0 and by the beta version of HotJava and is the standard to be supported by other "java enabled" browsers [3].

```
<APPLET
      [CODEBASE = codebase_URL]
      CODE=classname.class
      [ALT = alternate_text]
      [NAME = applet_instance_name]
      WIDTH=pixels HEIGHT=pixels
      [ALIGN = alignment]
      [VSPACE = pixels]
      [HSPACE = pixels]
>
[<PARAM NAME=param1_name VALUE="param1_value">]
[<PARAM NAME=param2_name VALUE="param2_value">]
[alternate_HTML]
</APPLET>
```

[ ] braces indicate that this part of the tag is optional.
The minimum requirements to place an applet on a page are:

```
<APPLET CODE=classname.class WIDTH=pixels HEIGHT=pixels>
</APPLET>
```

Applet resources (including their classes) are normally loaded relative to the document-URL (or <base> tag if it is defined). The codebase attribute is used to change this default behavior. If the codebase attribute is defined then it specifies a different location to find applet resources. The value can be an absolute URL or a relative URL. The absolute URL is used as is without modification and is not effected by the documents <base> tag. When the codebase attribute is relative, then it is relative to the document-URL (or <base> tag if defined).
Here is an example:

```
<applet codebase="applets/NervousText"
      code=NervousText.class
      width=300
      height=50>
<param name=text value="Java is Cool!">
</applet>
```

Data from the applet tag (code, codebase, name, alt, width, height, align, vspace and hspace) is made available to the applet. In addition, any parameters defined in the <param> tags are also made available to the applet. The applet invokes the getParameter method to get a String that contains the value of one of the above parameters.
Applet resources are loaded relative to the codebase, including images, audio files, etc. However, the applet does have access to the document-URL if resources from that location are desired.
The version 3 of Netscape Navigator introduces the changes in <applet> tag syntax. There is now a new attribute called MAYSCRIPT which is required for an applet that wishes to use LiveConnect and the netscape.javascript classes to interact with JavaScript. The APPLET tag now supports an ARCHIVE attribute: This can improve applet download time by reducing the number of HTTP connections required to fetch applet code. Example:

```
<applet archive="Nuclear.zip"
code="NuclearPlant.class"
width=680 height=473>
</applet>
```

This will cause the file Nuclear.zip to be downloaded to the user's disk, and will search it for the NuclearPlant class and the classes it requires. The zip file is found relative to the codebase path, and must not be compressed. Classes not in the zip file will still be searched for via the old mechanism if required.

1.1.1.JDBC

JDBC (Java Database Connectivity) is a standard SQL database access interface. It provides Java programmers with a uniform interface to a wide range of relational databases, and also provides a common base on which higher level tools and interfaces can be built. Partners such as Intersolv, Visigenic, and a dozen other database-connectivity vendors are providing JDBC drivers in the next few months for dozens of DBMSs, including Oracle, Sybase, and Informix. These database companies, and leading tool vendors, such as Symantec and Borland, have already endorsed the JDBC API and are developing products using JDBC.

Figure 2 JDBC architecture

The JDBC API defines classes to represent constructs such as database connections, SQL statements, result sets, and database metadata. JDBC allows a Java-powered program to issue SQL statements and process the results [4]. In conjunction with JDBC, JavaSoft is releasing a JDBC-ODBC bridge implementation that allows any of the dozens of existing Microsoft ODBC database drivers to operate as JDBC drivers. The JDBC-ODBC bridge can run on the server rather than client side using a JDBC driver that translates to a DBMS-independent network protocol. The architecture of JDBC is presented in Figure 2.

JDBC architecture can be divided into three layers. The highest one is the application using database access. The applications are developed by programmers with use of JDBC API, which allows to communicate with Driver Manager in a product-independent manner.

Driver Manager is the middle layer providing database-independent access to the data and managing vendor-specific drivers. Driver Manager is provided by JavaSoft. The low layer is driver layer. Drivers provide means to connect to specific databases and should be developed by database vendors. It is also possible to create new drivers with use of JDBC Driver API also provided by JavaSoft.

There are two possible scenarios for using JDBC. In *two tier access* Java application communicates with RDBMS directly. In this case driver manager loads appropriate drivers to the application (or applet).This approach has certain security-related limitations in case of applets (the database has to be on the same host as HTTP server for the applet).In *three tier access* an application server is introduced between the client and RDBMS. The server makes database connection on behalf of all clients and it also can access other resources (local file system). This approach requires building a protocol to access the application server. This also can be accomplished with use of Remote Objects (this technique is actually in the early release stage).

The Driver Manager is part of JDBC provided by SunSoft. Its responsibility is to keep track of available drivers. When one wants to access the database one obtains the Connection object using DriverManager.getConnection method. The method takes an URL as a parameter. The URL contains the name of the sub-protocol. Driver manager checks each driver if it supports specified sub-protocol. The first suitable driver is used to connect to the database.

When Java application wants to use database connection it must include the Driver Manager classes. Driver Manger has methods to establish connections to the database. Database is specified by the URL describing the location and the proper protocol. When application establishes the connection, it can create and send the statements to be executed in the database. The results are returned in form of ResultSet objects, which can then be processed by the application.

1.1.Netscape technologies

The Internet Application Framework [5] provided by Netscape Inc. is a comprehensive set of technologies for building and deploying applications for the Internet or for internal corporate intranets. This section provides an overview of the framework; individual components are discussed in more detail below.

The Internet Application Framework has three components: the Network Platform, the Client APIs, and the Sever APIs. This architecture is illustrated in the picture attached to the end of this document.

NETWORK PLATFORM

The Network Platform is a rich collection of cross-platform, open standards that enable developers to quickly create and deploy network-centric applications.

The Network Platform comprises the following technologies:

Java, a programming language created by Sun Microsystems that enables the creation of powerful network-centric applications.

JavaScript, a simple scripting language for the creation of simple applications which may be entirely embedded in HTML documents.

HTML (HyperText Markup Language), the nearly ubiquitous language for creating richly formatted documents on the World Wide Web. It is an international standard whose specification is maintained by the Internet Engineering Task Force (IETF).

VRML (Virtual Reality Modeling Language), a modeling language for creating three-dimensional worlds through which users can navigate.

LiveMedia, a standard proposed by Netscape Communications for creating interoperable real-time data servers and clients over the Internet.

Live 3D, an open platform for writing visually exciting 3D applications that combines VRML viewing

technology and powerful 3D extensions with Java, JavaScript and Netscape's Plug-in application programming interfaces (APIs).

Security standards that enable private communication to occur over the public Internet. A good example of a security protocol is SSL (Secure Sockets Layer), an open, widely endorsed channel security protocol originally defined by Netscape and in the process of being signed over to the Internet Engineering Task Force, that provides message encryption, client and server authentication, and message integrity services to many application protocols such as HTTP, FTP, NNTP, and others.

Electronic commerce standards. A collection of standards and protocols are necessary to make the Internet a viable platform for electronic commerce. For example, the Secure Electronic Transactions (SET) protocol is a joint Visa and MasterCard specification for safeguarding credit card transactions over the Internet. Netscape is working with leading electronic payment solution providers to define further standards to make electronic commerce a reality.

Database Access APIs. JavaScript defines objects that allow developers to access databases. Netscape is also working with leading database vendors to define standard ways for accessing cross-platform databases such as ODBC and Sun's emerging Java Database Connectivity (JDBC)technology.

Internet protocols. The Internet's exponential growth has been fueled by a collection of fundamental protocols and standards such as TCP/IP, SMTP, POP3, NNTP, FTP, IRC, Telnet, and MIME. These protocols form the foundation of the Internet Application Framework. New protocols are constantly being proposed such as the Real-Time Protocol (RTP) and the Lightweight Directory Access Protocol (LDAP), and the Internet Application Framework will incorporate these emerging protocols as products and services become available to support them.

CLIENT APIS

The Client-Specific APIs allow developers to extend the capabilities of the client. The technologies in this component of the Internet Application Framework include:

Inline Plug-in APIs provide a means of incorporating dynamically loadable modules into the client process itself. Together with Java and JavaScript facilities, Plug-ins enable high-performance application delivery of performance-intensive rich content, such as sound, graphics and video.

Netscape Client APIs allow developers to write applications on native operating system platforms such as Windows, MacOS, and UNIX that can communicate and remotely control a client application such as Netscape Navigator. These APIs allow applications to communicate via Apple Events, OLE Automation, DDE (Dynamic Data Exchange), and X Events.

SERVER APIS

The Server-Specific APIs enable developers to write server-side applications to extend the capabilities of any Web server. CGI (Common Gateway Interface), the Internet standard interface for invoking server-based scripts or compiled programs at the request of clients. It is currently the most common way of providing clients dynamic content such as up-to-date stock information or a document in the language the user has requested.

Netscape Server APIs (NSAPI), provide developers with the ability to extend the native capabilities of the server. NSAPI is a highly granular API that allows developers to exert a great deal of control over the server's behavior by creating high-performance extensions. For example, developers may choose to implement their own mechanisms for access control, user authentication, and error logging.

Netscape is working with leading Internet software providers to define a standard server API defined in Java for cross-platform compatibility.

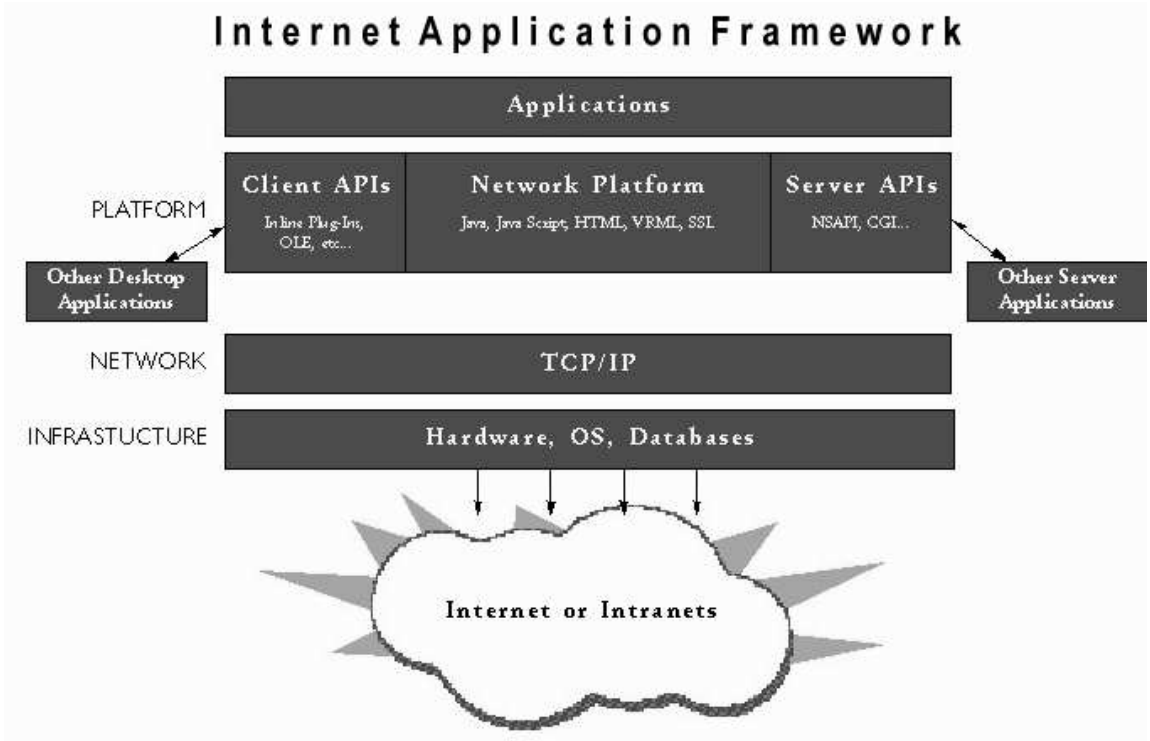


Figure 3 Internet Application Framework

1.1.1.JavaScript

JavaScript [6] is a compact, object-based scripting language for developing client Internet applications. Netscape Navigator 2.0 interprets JavaScript statements embedded directly in an HTML page.
In a client application for Navigator, JavaScript statements embedded in an HTML page can recognize and respond to user events such as mouse clicks, form input, and page navigation.
JavaScript is based on a simple object-oriented paradigm. An object is a construct with properties that are JavaScript variables. Properties can be other objects. Functions associated with an object are known as the object's methods. A JavaScript object has properties associated with it. The properties of an object are accessed with a simple notation:

```
objectName.propertyName
```

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure--a set of statments that performs a specific task. A function definition consists of the *function* keyword, followed by the name of the function, a list of arguments and the JavaScript statements that define the function. A method is a function associated with an object. They are defined in the same way as a standard function. The following syntax is used to associate the function with an existing object:

```
object.methodname = function_name
```

When a new page is loaded in Navigator, it creates a number of objects corresponding to the page, its contents, and other pertinent information.
Every page always has the following objects:

- window: the top-level object; contains properties that apply to the entire window. There is also a window object for each for "child window" in a frames document.
- location: contains properties on the current URL
- history: contains properties representing URLs the user has previously visited
- document: contains properties for content in the current document, such as title, background color, and forms

The properties of the document object are largely content-dependent. That is, they are created based on the content that you put in the document. For example, the document object has a property for each form and each anchor in the document.

JavaScript can be embedded in an HTML document in two ways:

- As statements and functions using the SCRIPT tag.
- As event handlers using HTML tags.

A script embedded in HTML with the SCRIPT tag uses the format:

```
<SCRIPT>
   JavaScript statements...
</SCRIPT>
```

The optional LANGUAGE attribute specifies the scripting language as follows:

```
<SCRIPT LANGUAGE="JavaScript">
   JavaScript statements...
</SCRIPT>
```

Scripts placed within SCRIPT tags are evaluated after the page loads.Functions are stored, but not executed. Functions are executed by events in the page.

```
<SCRIPT LANGUAGE="JavaScript">
function square(i) {
    document.write("The call passed ", i ," to the
function.","&lt;BR&gt;")
    return i * i
  }
  document.write("The function returned ",square(5),".")
</SCRIPT>
```

1.1.1.Plug-ins

Plug-Ins [7] are third party programs which allow users to extend the capabilities of the Netscape Navigator to include native support for new data types and additional features. Plug-Ins appear as supplemental capabilities of the Netscape client. Plug-Ins allow the delivery of multimedia content through Internet sites. Now users of Netscape can view not just still images, but also video, audio, and animation. One example of using plug-in to handle new type of data is described in [8] .

One of the fundamental uses of a plug-in is to fetch a URL with all the network functionality of the standard Netscape client. The data from such a URL is provided as a stream as it arrives from the network. This allows the plug-in to implement a progressive viewer, or make a particular decision without seeing an entire stream. In other words, plug-ins allow the user to see the data as it arrives, and not wait until the entire file is downloaded. While the user is receiving this data, they can do other things, such as view the rest of the web-page.

Plug-ins are embedded in the HTML page with use of <embed> tag. The syntax of this tag is the following:

```
<EMBED SRC=url  WIDTH=size  HEIGHT=size  PARAMETER_NAME=parameter_value
...>
```

The three first attributes are default and specify the URL of the source document that plug-in will handle, the width and height of the plug-in appearance in the browser window. The other parameters are optional and may set plug-in specific features. Plug-in may also have NAME parameter which specifies the name it will be known for other LiveConnect components.

Sample EMBED tags:

```
<EMBED SRC="MyMovie.mov", WIDTH=150, HEIGHT=250 CONTROLS=TRUE
NAME=Movie>
<EMBED SRC="DoomGame.ids", WIDTH=400, HEIGHT=300 SPEED=SLOW LEVEL=12>
```

The Three Plug-In Modes

There are three modes of operation for plug-ins : embedded, full-page, or hidden.

An embedded plug-in is a part of a larger HTML document, where the plug-in is visible as a rectangular subpart of an HTML page (similar to how a GIF is currently embedded). This image can be live and may respond to user events such as a mouse action. An example of this plug-in mode would be QuickTime movie players.

A full-page plug-in is a viewer that is not a part of an HTML document. The plug-in completely fills the inner frame of a Netscape window with its representation of some data type. An example of this plug-in mode would be an Adobe Acrobat viewer.

A hidden plug-in is one that runs in the background. No plug-ins of this type have been created, yet, but a good example would be a MIDI player.

How Plug-Ins Work

Each plug-in is associated with one or more MIME data types that the Netscape client has no native support

for. When Netscape encounters an unknown data type from a server, it looks for a plug-in that is associated with that MIME type and loads it.
A new instance of a plug-in is created whenever Netscape encounters data to be handled by that plug-in. There may be many instances of the same plug-in if there are mulitiple embedded objects on a single page, or if there are several Netscape windows open each displaying the same data type. Plug-in instances are deleted when a user leaves a page.
All events (e.g. user interface and window graphics) for an instance of a plug-in is dependent upon the platform. On the Macintosh the native window is shared between the plug-in and the Netscape client, and so it is Netscape that effectively runs the show by providing the plug-in with the events. On other platforms event processing is relative to the window hierarchy.
If a page is viewed that requires a plug-in that a user's Netscape does not have, the user is sometimes given the option to download that plug-in from a particular site.
Using LiveConnect, a Plug-In is made to interact with the Java object model and as such can interact with JavaScript and Java.

1.1.1.Netscape LiveConnect

LiveConnect [7] is the youngest Netscape technology, introduced in Netscape Navigator 3.0 in August 1996. It extends Java to other Netscape technologies like JavaScript, HTML, and Plug-ins so that they can interact seamlessly with one another.
LiveConnect links these objects together and enables Java and JavaScript to script and control them. This allows richer, more complex applications to be developed on a web page. A mouse click can trigger a JavaScript; a Java applet can use an audio Plug-In.
LiveConnect works because it is based upon the platform-independent, Java object model. Using LiveConnect, each component on a web page can expose its functionality via a Java interface. Likewise, each component can call other functions in other objects regardless of their composition. LiveConnect is built on the top Java Runtime Interface which formalizes interface to Java services.
With its Java library Netscape provides two classes - Plugin and JSObject.

- The Plugin class represents the Java reflection of a plugin. Plugins which wish to have Java methods associated with them should subclass this class and add new (possibly native) methods to it. This allows other Java entities (such as applets and JavaScript code) to manipulate the plugin.
- JSObject allows Java to manipulate objects that are defined in JavaScript. Values passed from Java to JavaScript are converted to wrappers, which can be used to access methods and fields of the java object.

This is the example how Java program can get a handle of a plugin and use its method:
Suppose we have a HTML document with the following tags:

```
<applet code="Control.class" name="myApplet" width=200 height=300
mayscript>
</applet>
<embed src="sample.263" name="myPlugin" width=300 height=300>
```

We have to insert the following code in our applet to get the reference to the plugin embedded on the same HTML page.

```
JSObject doc = (JSObject)JSObject.getWindow(this).getMember("document");
MyPlugin pl = (MyPlugin)doc.getMember("myPlugin");
```

After this is done, Java applet can call methods defined in the plug-in, either in its Java or native C part.
This technology was used to provide graphical interface to the plug-in plying H263 video sequences, developed by Janusz Bulawa. Originally, the control button to start and stop the sequence were written in C with use of the Motif interface. Replacing them by Java applet allowed to reduce the amount of plug-in code from 150 to 10 lines and introduced much better code portability.

1.Java and WWW Environments

This section presents three collaborative frameworks based on the technologies presented in previous section. Because all of them are relatively young, they are still evolving. Although they do not provide the whole functionality that we need (see paragraph 4 for details), some of the ideas introduced by these systems contributed to our design. While describing these environments I focus on the features relevant to the design of our system. In the case of Habanero environment it is its collabratory model and wrapped interface concept. The Caltech project introduces communication mechanisms in Java across the Internet. The InSoft environment is a collaboratory environment tightly integrated with Netscape browser.

1.1.Habanero

Habanero [10] is a framework for sharing Java objects with users distributed around the Internet. Included, or panned, are all the networking facilities, routing, arbitration and synchronization mechanisms necessary to accomplish the sharing of state data and key events between collaborator's copies of a software tool. Authentication and privacy features are also planned. There is no inherent limit in the number of tools per session, nor is there a limit on the type of tools that may be shared. As the project progresses, additional capabilities will enable routing of Habanero session information to a very wide number of participants. A limited, but representative set of applications is provided with current version. No security model is presently in place.

The idea behind Habanero is that applications become collaborative by sharing the same state, and by sharing Actions, which are delivered to each collaborative client, to change state. Actions are standard events in Java. Normally they are caused by user input, such as keyboard press or mouse click. Java then looks for a GUI component which knows how to handle the Action event. Habanero provides framework which catches Action events and, apart from normal processing, sends them to all others applications participating in the session.

In order to turn an applet into a collaborative application:

- Its constructor must take no arguments. If yours takes arguments, make one that uses defaults or static variables and add a way for the user to set them.
- It must implement the Wrapped interface. Here are the relevant calls:
- All state changes that should be shared must happen through Actions. The easiest way to do this is to use AWT ACTION_EVENTS. These are automatically translated by the system into a kind of Action. See the sample code :
-
- `Event ac = new Event(this,Event.ACTION_EVENT,someMarshallableObject);`
- `postEvent(ac);`MACROBUTTON HtmlDirect </P>
-
-

In order to use an ACTION_EVENT, it is often useful to create a new class which is used to hold information which needs to be sent to each collaborative client. This should implement the Marshallable interface and should be passed as the arg parameter of the Event.

If the action already generates an event, such as an mouseMove or mouseDown one needs to call `f.addEventCode(EVENT)`. f is the MirrorFrame. Here is an example: MACROBUTTON HtmlDirect </P>

```
f.addEventCode(Event.MOUSE_DOWN);
```

One way to make use of the Marshallable properties of other objects is to use the Marshall and Unmarshall methods. These methods allow to create objects with specific subclasses, but one does not have to worry about the specific types as they are passed through the Habanero kernel. To use Marshall and Unmarshall, one needs to declare the classes as subclasses of Marshalling. This provides the Marshall and Unmarshall method to the program. Here is an example:MACROBUTTON HtmlDirect </P>

```
        public class BaseClass extends ncsa.habanero.Marshalling
                 implements ncsa.habanero.Marshallable {
            ...
    }MACROBUTTON HtmlDirect </P>
```

Then when one needs to marshall an object derived from this `BaseClass`, one call Marshall. To read the object, Unmarshall should be called and the result should be casted to the desired class. For instance, MACROBUTTON HtmlDirect </P>

```
//  Saving the object
UnknownObject.Marshall (out);

//  Reading the object
UnknownObject = (BaseClass) Unmarshall (in);
```
MACROBUTTON HtmlDirect </P>

Habanero system is built in a client/server architecture. The server manages the sessions and distributes events among clients. Client provides user with Graphical User Interface to visualize session, participants and available applications. The sample view of Habanero client is presented in Figure 4 and Figure 5. The former shows main session control screen. Using provided interface user can either establish a new or connect to already existing session. It is possible to enter user personal information (name, photo, address) to present it to other session participants. Figure 6 presents control screen for the session. From this window user can start the applications (visible as icons). System ensures that after one user starts an application, Habanero opens its instance in all other session participants. If a user joins a running session all currently running applications will be also opened for him and current application state will be restored with use of the Marshalling mechanism described before.

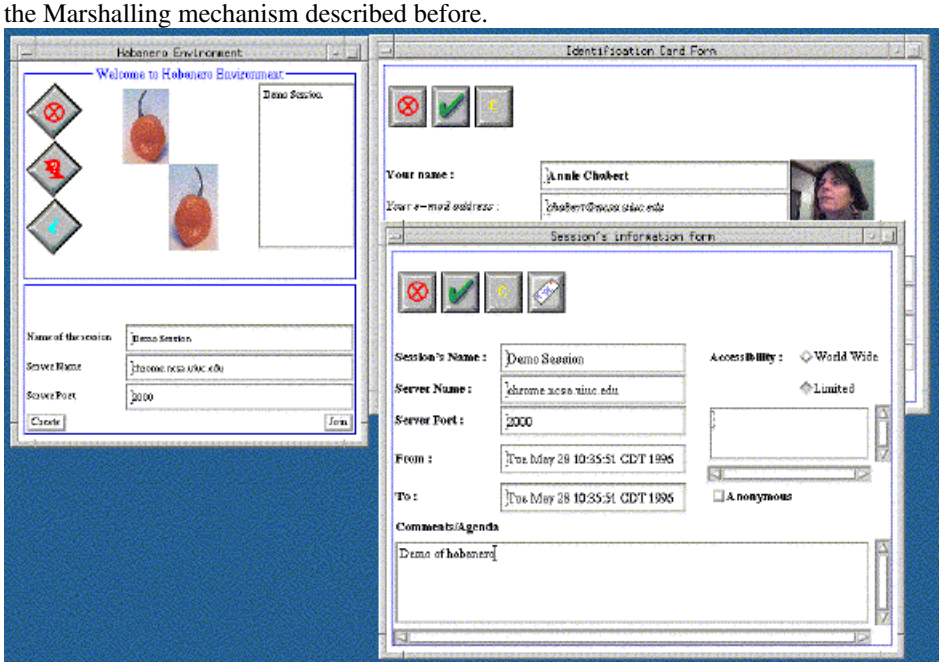


Figure 4 Habanero session management windows

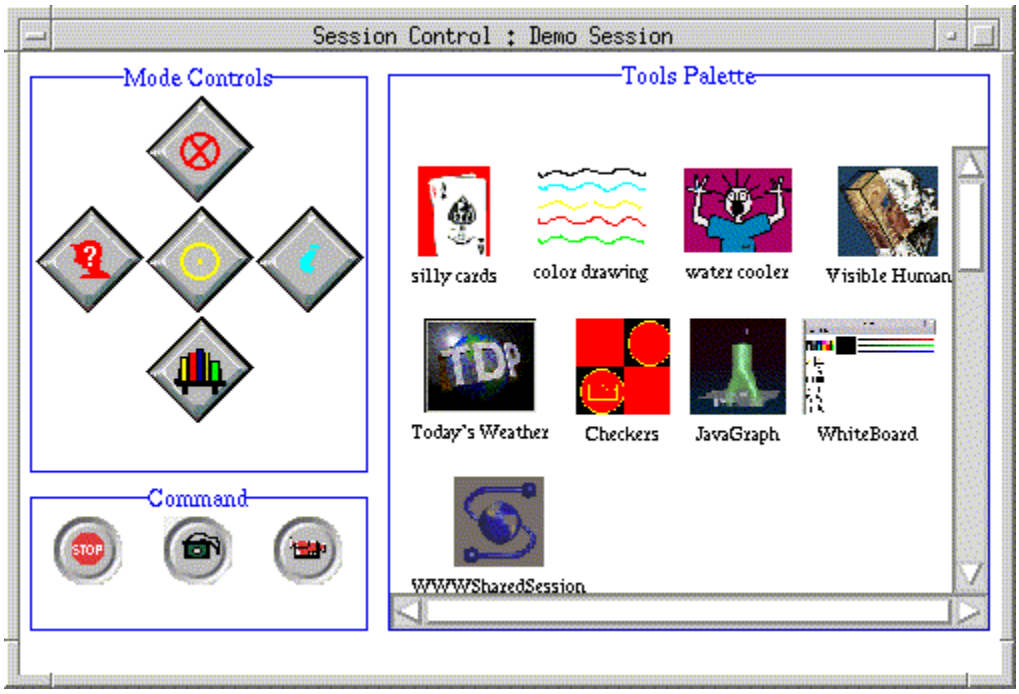


Figure 5 Habanero sample session window

1.1.WWW distributed system at Caltech

This section describes the distributed system designed in California Institute of Technology [11]. The system is built with use of Java language and supports peer-to-peer communication among processes spread across a network.

The authors introduce the notion of "dapplet" to distinguish a process used in a collaborative distributed application from processes used in traditional distributed systems. Dapplets are composed together to form distributed sessions. A session is a temporary network of dapplets that carries out a task such as arranging a meeting time for a group of people. Sessions need not be static; after initiation, they may grow and shrink as required by the dapplets.

Consider the example of a center director setting up an executive committee meeting with members from different sites. Prior to the session, each committee member has installed a calendar dapplet. A calendar dapplet is a process: it operates in a single address space, it communicates with files by standard I/O operations, and it communicates with other processes through ports. Associated with each dapplet is an Internet address (i.e., IP address and port id).

A session is an instance of an application, implemented as a network of dapplets. A session consists of many different types of dapplets. For instance, a calendar application may have calendar user processes and secretary processes. Programs corresponding to each process type are installed on the appropriate machines; for the session in Figure 6, the calendar user dapplets and secretary dapplets are processes running on their respective users' desktop computers.

Associated with each session is an initial process --- an initiator dapplet --- that is responsible for linking dapplets together. In our example, the center director invokes an initiator dapplet, and passes it a directory of addresses (e.g., Internet IP addresses and ports) of component dapplets that are to be linked together into a session, as illustrated in Figure 6.

MACROBUTTON HtmlResImg

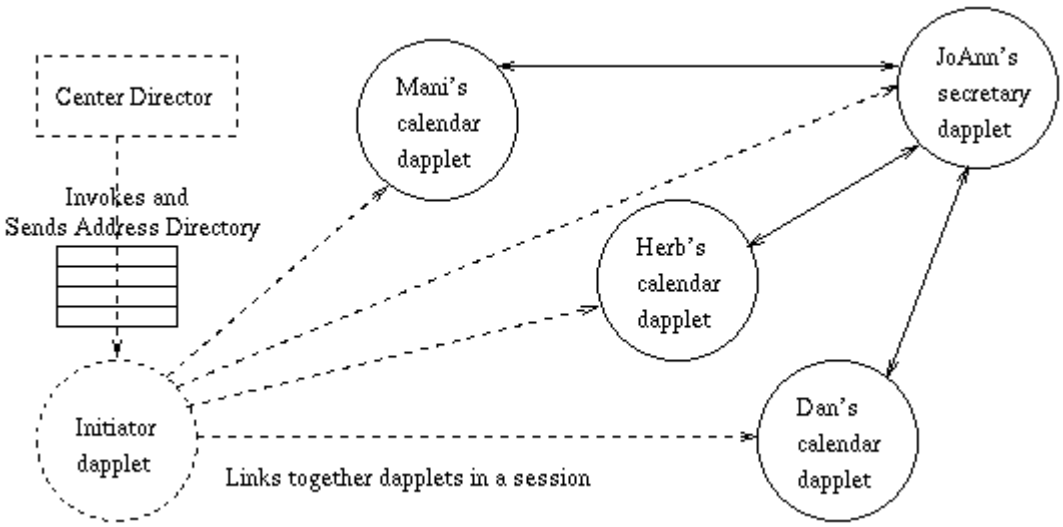


Figure 6 : initiator uses the invoker's address directory to set up a session between existing dapplets.
Dapplet connections are achieved using the address directory. The initiator dapplet sends a request to the component dapplets; this request asks the components to link themselves up to form a session. For example, in our calendar session, each calendar user dapplet may be linked to a common coordinating secretary dapplet, as is done in Figure 6. As another example, in a distributed card game session, a player dapplet may be linked to its predecessor and successor player dapplets (which correspond to the players to its left and right, respectively).
A dapplet, on receiving a request to participate in a session, may accept the request and link itself up, or it may reject the request (because the requesting dapplet was not on its access control list, or because it is already participating in a session and another concurrent session would cause interference). When a session terminates, component dapplets unlink themselves from each other.

1.1.1.1.Messages.

Objects that are sent from one process to another are subclasses of a *message* class. An object that is sent by a process is converted into a string, sent across the network, and then reconstructed back into its original type by the receiving process. Java methods are used to convert an object to a string and to create an instance of the sending object at the receiver.

1.1.1.1.Inboxes, Outboxes, and Channels.

Each process has a set of *inboxes* and a set of *outboxes*. Inboxes and outboxes are message queues. A process can append a message to the tail of one of its outboxes, and it can remove the message at the head of one of its inboxes. The methods that can be invoked on inbox and outbox objects are described later.
Each inbox has a global address: the address of its dapplet (i.e., its IP address and port) and a local reference within the dapplet process.
MACROBUTTON HtmlResImg

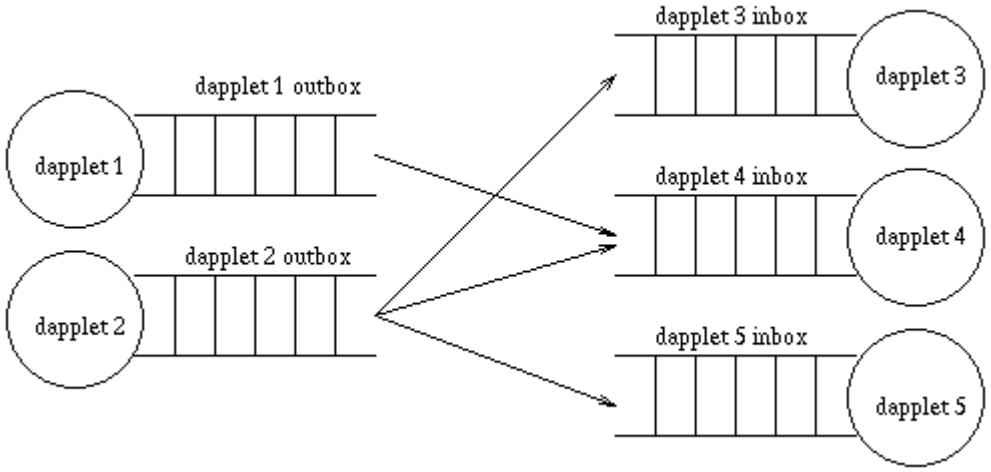


Figure 7 example of dapplet inbox and outbox connections

Associated with each outbox is a set of inboxes to which the outbox is bound; there is a *message channel* from an outbox to each inbox to which it is bound; an example of a set of bound dapplet inboxes and outboxes is given in Figure 7. Each message channel is directed from exactly one outbox to exactly one inbox. Messages sent along a channel are delivered in the order sent. Message delays in channels are arbitrary.
As shown in Figure 7, an outbox can be bound to an arbitrary number of inboxes. Likewise, an inbox can be bound to an arbitrary number of outboxes. Therefore, there are an arbitrary number of outgoing channels from an outbox, and there are an arbitrary number of incoming channels to an inbox.
The distributed computing layer removes the message at the head of a nonempty outbox and sends a copy of the message along all channels connected to that outbox. The network layer delivers a message in a channel to the destination inbox of the channel. The delay incurred by a message on a channel is arbitrary; the delay is independent of the delay experienced by other messages on that channel, and it is independent of the delay on other channels. Also, if a message is not delivered within a specified time, an exception is raised.

1.1.InSoft environment

InSoft, Inc., currently part of Netscape Inc., markets collaborative computing, desktop conferencing and distributed digital video development tools and business applications. The company's products are based on its open software architecture, Digital Video Everywhere, DVE [12], which enables:

- collaboration between multiple desktop systems
- transparent network connectivity
- video interoperability

Collaborative applications all share at least some of the following elements:

- Session Management
- In collaborative applications conference members are added or dropped, and session ended.
- Event Distribution
- Collaborative applications often require events to be distributed to members of the session
- Data Distribution
- Data needs to be distributed to the appropriate members of the session or subgroups. The data might be text, graphics, images, files, as well as audio and video.

The OpenDVE architecture has a built-in client-engine. This architecture provides the common functionality in a Conference Engine with the ability to add modules whenever necessary. These modules are called Plugins. The Conference Engine provides the application writer with an "abstract network" and may be thought of as the runtime module for OpenDVE applications. Each Plugin is connected to the Engine through an independent IPC mechanism. Plugin modules can communicate among themselves using the services and connections provided by the Conference Engine. Plugins provide the user with functions

making-up the substance of collaborative applications. Plugins are responsible for the actual processing and staging of data as well as presenting the conference to the user.
The architecture of OpenDVE is presented in Figure 8 OpenDVE architectureFigure 8. More information can be found in [9].

Figure 8 OpenDVE architecture

1.1.Critical assessment of presented environments

The three systems provide users with new and very interesting features, enabling them to truly cooperate across the network. The approaches used in each of projects described above have been studied to verify how they fit to our requirements.
At the beginning Habanero system seemed to be very suitable for our purposes. It offers integrated environment to manage user activities, provides means to migrate existing applications to the system and offers interesting communication model. However, after more careful study, it turned out that it would be difficult to satisfy all our requirements. Despite the use of Java, Habanero environment is completely standalone, and all applications being part of the system must reside locally on each host. Porting the system to the Web environment would not be straightforward because of applets security limitations. The communication model is tightly integrated with Java language (makes use of Java event distribution). This solution is very elegant for pure Java applications, but causes problems when one wants to attach for instance C application.
The Caltech system is in fact not a working product, but more an academic research program. The system they propose contains very interesting concepts of inter-applications communication using ports, but does not have any high-level mechanisms to create sessions running different kinds of applications. Because of its distributed architecture it would be difficult to port it to Web browser and also to implement database access capabilities.
The OpenDVE environment is probably the most mature collaboratory system running in the Web environment. From our studies we realized that it would be possible to create a cllaborative environment with the features we need using OpenDVE architecture. However it would be quite a significant effort, because the programmers interface provided by OpenDVE is very often insufficient to fully control the environment. Nonetheless, we will still use OpenDVE as part of our system, providing videoconference capabilities, while relaying on our system for connection and session management.
After the evaluation of these three cases we decided to create our own system, using the ideas introduced by studied environments, and adding capabilities that would meet our requirements.

1.System architecture

This section describes architecture of the collaboration platform used as a communication backbone. In general, elements of the whole system may run on different platforms. In addition, accessibility of a particular system element should depend on the privileges granted to the user. Various functions may be efficiently implemented using different mechanisms and environments.MACROBUTTON HtmlDirect </P>
These basic requirements may be efficiently satisfied in a system which is composed of a set of separate applications. Each application will serve a particular set of functions. On one hand these applications will be independent in the sense that they provide separate sets of functions and may be implemented on different platforms. On the other hand, they are parts of one complicated system and have to be unified and interact in a way giving the user sense of using one big application.MACROBUTTON HtmlDirect </P>
One of the main assumptions while developing the system is that applications may be written in any programming language and may run on any hardware or software platform. Keeping this in mind we use only the most generic system communication mechanisms.MACROBUTTON HtmlDirect </P>
Other important requirement is that the set of applications constituting the whole system is not complete and may change in the future. As the result, the structure of the communication system must not depend on the applications connected to it. The system implements a generic set of communication mechanisms. These mechanism allows applications to communicate one to each other. Application independence guarantees that the communication system will be able to serve any applications that will be developed in the future and added to the system.
While integrating different kinds of applications, the system will focus on using Web technologies.MACROBUTTON HtmlDirect </P> Especially the GUI that user will be presented with will

be based on Web browser and elements provided by it, like hyperlinks, frames and applets. Although significant part of the code will be downloaded through the network, there will be still applications that must be present locally, for instance for performance reasons.
One of the important functions of the system apart from the communication itself is constant monitoring of the system events. All actions of all simulation participants will be recorded in a database. This database may be analyzed after the action. The "backward flow of control" will enable simulating all events by driving the system from the database. This will enable using the system as a simulation or training environment to put, for example, the crew in a particular point in time in a particular scenario and begin simulation.MACROBUTTON HtmlDirect </P>
The communication platform is one of the most important elements in such a system. It must provide means of efficient data sharing, interaction among applications, and constant monitoring and recording of system events in a database. Various kinds of communication are considered. These include:MACROBUTTON HtmlDirect </P>

- communication among parts of the same application running on the same host
- communication among instances of application running on various hosts
- communication among various applications

1.1.Dictionary

Applet - program written in Java language running in WWW environment. Applets' code is transported through the network and runs in the Java-capable browser on the client machine. Applets will be a specific case of client application. MACROBUTTON HtmlDirect </P>
Central Server - part of the system which task is to pass messages between demons, according to established Virtual Connections. Central Server maintains a connection to the database storing user profiles, privileges and all system events. MACROBUTTON HtmlDirect </P>
Client Application - each of the applications running on a client host and connected to the collaboratory system. Client applications are connected to the system through demons. Client applications communicate with each other with use of application-specific protocols. The collaboratory system passes the messages to all the applications of the same type belonging to the same session. MACROBUTTON HtmlDirect </P>
Communication System - collaborative environment used to transport events between client applications. System consists of demons, central server and database. MACROBUTTON HtmlDirect </P>
Control Application - one of client applications. Its task is to provide user with graphical interface to the system. Control application will allow launching other applications on local and remote hosts and change parameters of already running applications. Control application will display all sessions, the user has rights to see (according to his/her privileges). MACROBUTTON HtmlDirect </P>
Database - database connected to the Central Server and used to store persistent data about users and all system events. Database keeps track of all system activity and allows replaying parts of it at any time.MACROBUTTON HtmlDirect </P>
Demon - part of the system residing on each participant's host. Its task is to connect client applications to the system. One demon corresponds to one Participant. Each demon is connected to the Central Server.MACROBUTTON HtmlDirect </P>
Event - there are two types of events: application events and system events. Client Applications share their state by transmitting application specific events through the system. Applications define the events themselves. System events are generated by Control Application. They correspond to events like launching application, creating session, joining session etc.MACROBUTTON HtmlDirect </P>
Participant - each user which is currently logged into the system. Participants can use Client Applications according to rights granted by the system. MACROBUTTON HtmlDirect </P>
Session - abstract grouping of instances of particular type of application, created to enable collaborative work. Session may contain one or more application instances. When session contains single application instance it is running locally in single user mode. When session contains more application instances all of them communicate to enable collaborative work. Session information is maintained in database, and used by Central Server to replicate incoming messages. MACROBUTTON HtmlDirect </P>
User - each person which is privileged to connect to the system. Information about Users, including their rights is stored in the Database. A User becomes a Participant when he/she connects to the system.

MACROBUTTON HtmlDirect </P>
Virtual connection - abstract link connecting client applications and providing means of passing application-specific events MACROBUTTON HtmlDirect </P>

Abbreviations

AID - application or applet id MACROBUTTON HtmlDirect </P>
UID - user id MACROBUTTON HtmlDirect </P>
HID - host id MACROBUTTON HtmlDirect </P>
AT - application type MACROBUTTON HtmlDirect </P>
SID - session id MACROBUTTON HtmlDirect </P>
CA - control application MACROBUTTON HtmlDirect </P>
CS - central server MACROBUTTON HtmlDirect </P>
LD -local demon

1.1.Overview MACROBUTTON HtmlDirect </P>

We propose the following architecture. On each host there will be a demon running. On one of the hosts there will be a specialized server running. The server will maintain connections with demons on other participating hosts. The scheme of this architecture is presented in Figure 9. MACROBUTTON HtmlDirect </P>

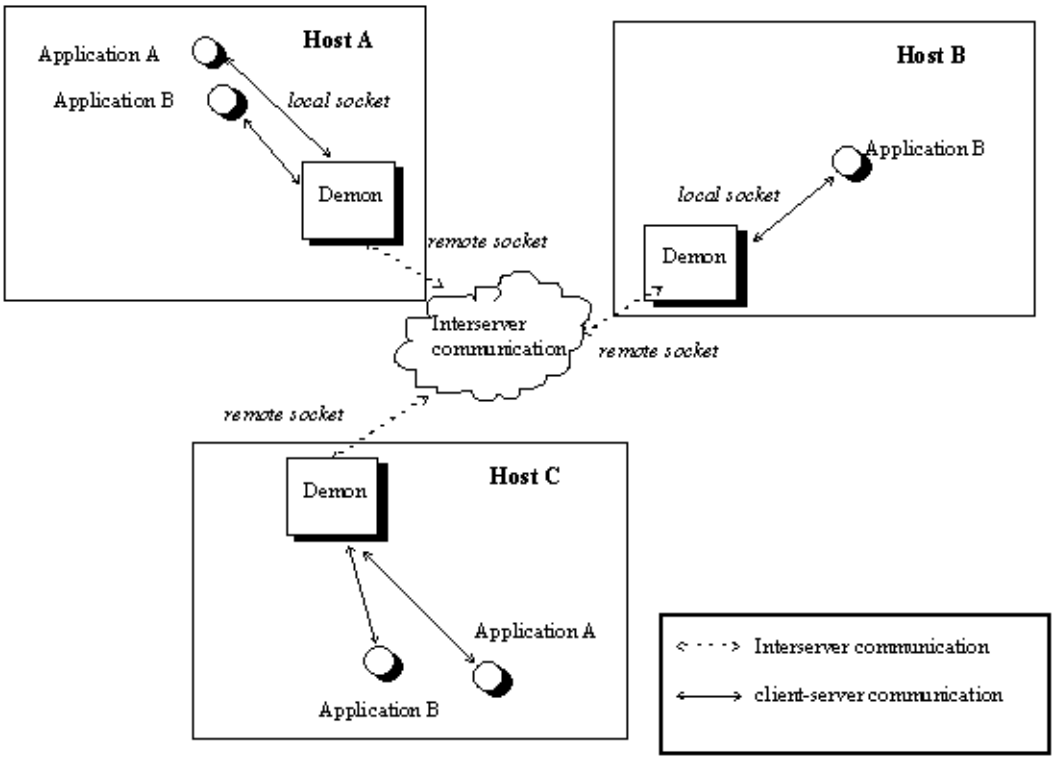
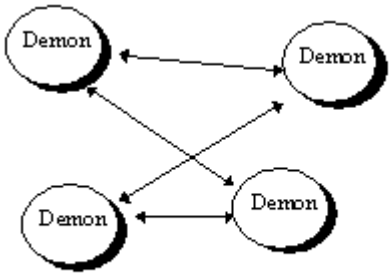


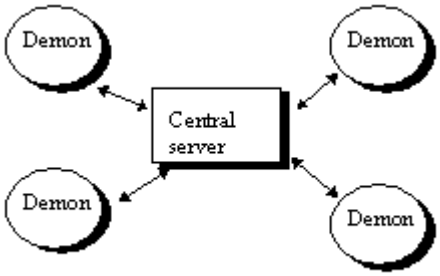
Figure 9 Overview of system architecture

1.1.1.MACROBUTTON HtmlDirect <P ALIGN="CENTER">Interserver communication aspects

On the logical level the architecture of the system is fully distributed. On each host there is a separate demon running. This demon is responsible for maintaining connections to all local applications and lunching local applications when necessary. These demons can establish Virtual Connections to other demons running on other hosts. Virtual Connections are established for particular application. These connection are used to exchange information between collaborating applications. See Figure 10.MACROBUTTON HtmlDirect </P>



MACROBUTTON HtmlResImg
Figure 10 Fully distributed configuration (logical level)
MACROBUTTON HtmlDirect <P ALIGN="CENTER">MACROBUTTON HtmlDirect </P>
On the implementation level, however, the structure is slightly different. The requirement to record all system events in the database, causes that all event have to be sent to the demon connected to the database. In order to avoid message duplication, and simplify the session management, this server will be resending the messages to all session participants. The host running central server will also provide access to the database and services for user authentication and event logging. Such architecture will also enable controlling the whole system by events coming from the database. See Figure 11.MACROBUTTON HtmlDirect </P>



MACROBUTTON HtmlResImg
Figure 11 Central server configuration (implementation level)
MACROBUTTON HtmlDirect <P ALIGN="CENTER">MACROBUTTON HtmlDirect </P>
In the first version of the system incoming messages will be sent to each application in particular session separately. In the future we plan to use multicast technique to increase server performance.
MACROBUTTON HtmlDirect </P>
Another important element in the design is the host running HTTP server. Its task will be to serve applets (see point 4.2) and provide applications with data (for example, for GIS application). MACROBUTTON HtmlDirect </P>

1.1.1.Interapplication communication

The demon-application interface will be provided on local sockets and procedure calling basis. It will be responsibility of each application to read and process each message properly. We assume that each application will be started with a number of port assigned to it, passed as a starting parameter. It will be responsibility of the demon to provide valid port number and send messages to the application. Server must also be prepared for the case that application finishes abnormally (e.g. crashes). In this case connection must be closed properly and other applications notified. MACROBUTTON HtmlDirect </P>

1.1.Architecture details

1.1.1.System components

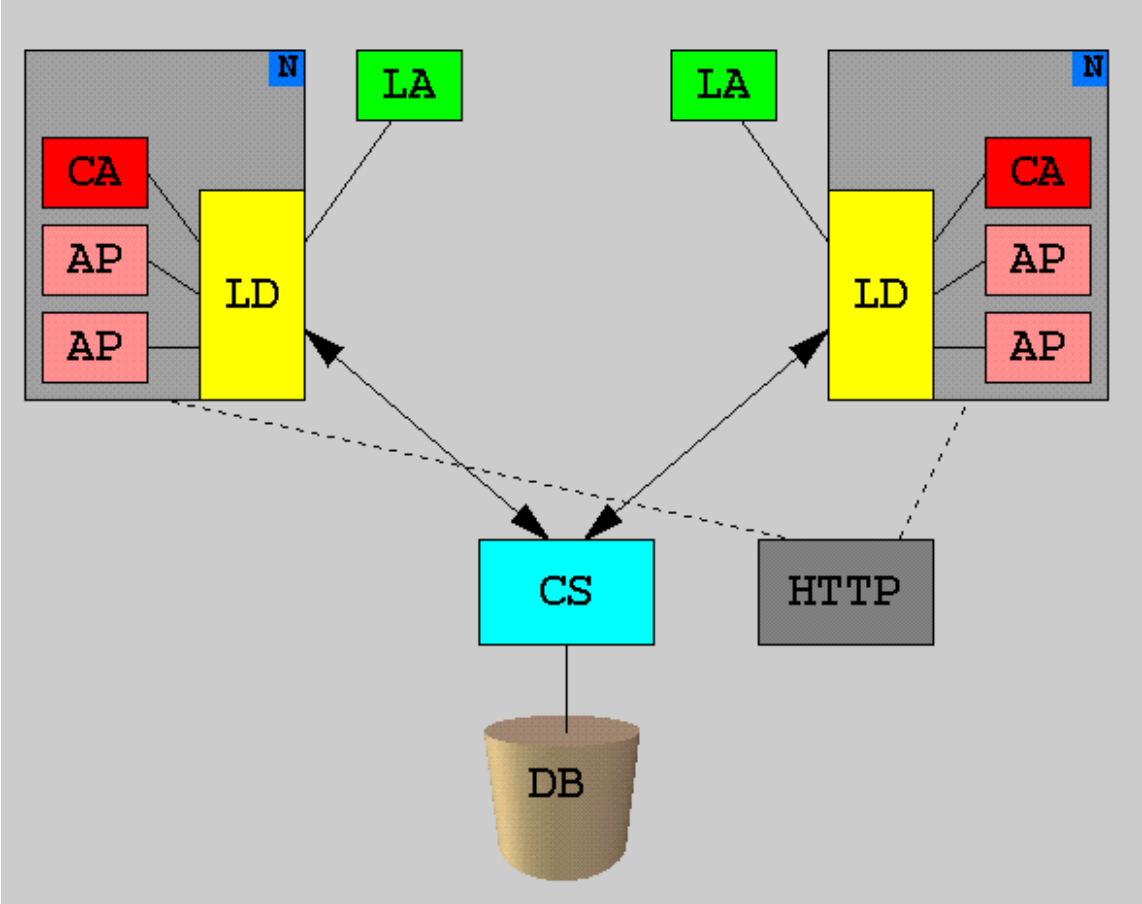


Figure 12 Communication model

Local Demons (Demons)

Local Demon is relatively simple piece of software. It is implemented as a plug-in in netscape browser. Its main task are maintaining two way communication between user applications, applets and Central Server, and lunching local applications.MACROBUTTON HtmlDirect </P>

Central Server

Central Server is the main communication element. All Local Demons communicate with the Central Server. This server maintains the whole system state (Participants, Applications, Sessions). Its main tasks are routing of messages among applications participating each session and recording all events in the database. Central Server is written in Java and runs as an application. It uses JDBC standard to connect Oracle database system.MACROBUTTON HtmlDirect </P>

Local Applications

Local Applications running on a host will be connected to the Demon running on this host. Local Application can be written in any programming language. All Local Applications communicate with Local Demon by the use of sockets. The Demon is responsible for starting applications, routing messages to and from applications. This architecture will provide a seamless transmission of messages among applications. The application specific protocols are not interpreted by the communication system.MACROBUTTON HtmlDirect </P>

Java Applets

Java Applets are downloaded through the network from an HTTP server. They are executed inside Netscape browser. Communication between Java Applets and Central Server is maintained by the Local Demons. Java Applets communicate with Local Demon by calling its functions.

Control Application

Control Application is a specialized application which serves as an interface to a user. Control Application is used to launch applications locally or remotely, create and connect existing sessions. This applications allows logging into the system. Structure of the Control Application depends on the user privileges giving a

user only these features he/she is allowed to use.MACROBUTTON HtmlDirect </P>
Control Application communicates with the Local Demon. Through the Local Demon all the communicates to and from Control Application are sent.MACROBUTTON HtmlDirect </P>
Control Application is written as an Java applet. A small part of it is launched on the beginning giving a user a possibility to login to the system. Then the rest is downloaded from the http server.
MACROBUTTON HtmlDirect </P>

1.1.Communication between applications

There are two modes of communication between applications: MACROBUTTON HtmlDirect </P>

1.1.1.Partner mode

In partner mode applications can communicate with each other symmetrically. All users can perform the same kind of actions. Applications using this pattern are Chat, Videoconferencing. However, a logical master exists for each session. This master can decide, for example, whether or not another user can join this session.MACROBUTTON HtmlDirect </P>

1.1.1.Master-slave mode

In master-slave mode there is one distinguished "master" application which performs activities. The other applications of the same type connected with "master" serve as slaves and display the activities performed by "master". There will be an option to change the master dynamically during the session if user privileges allow such operation. Applications using master-slave pattern are Visible Human, GIS and Search System.
MACROBUTTON HtmlDirect </P>

1.1.Network services

The system will provide services of user authentication and event logging. These tasks will be done by Central Server which will maintain connection to the database. MACROBUTTON HtmlDirect </P>
MACROBUTTON HtmlDirect <P ALIGN="CENTER"> MACROBUTTON HtmlResImg

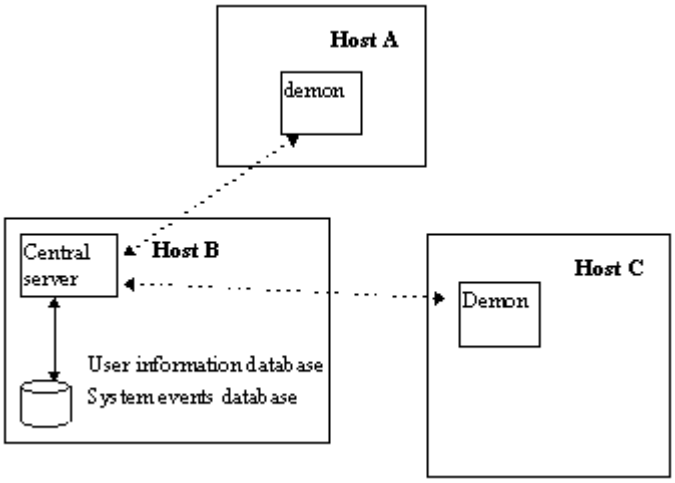


Figure 13 Network services
MACROBUTTON HtmlDirect <P ALIGN="CENTER">

1.1.1.User authentication and security policy

Database will contain all user information. User privileges will be a part of this information. Depending on privileges granted, users may be or may not be able to initiate various actions (launching new applications, sending messages).
Each user of the system will have to identify himself/herself to start working with the system. Identification will be based on the password associated with each user name. After verification each user will be granted rights specific for him. There will be the following types of rights:MACROBUTTON HtmlDirect </P>

- launching particular application in single user mode (creating a session) - in this mode user can use application without any collaboration with other participants. It is possible however for other participant

with sufficient rights to launch another application of the same type and connect to the user to monitor his activities.

- joining the session - this property allows User to connect local version of the application to an existing session.
- launching an application on another computer - this property allows to launch application on a remote computer with another Participant. The remote application joins the session. The local application runs in master mode.
- using application in master mode - when applications runs in master-slave mode (see 4.4) this property allows to dynamically switch to become the session master.

The rights to perform each of the actions mentioned above depends only on the User and type of application. MACROBUTTON HtmlDirect </P>

1.1.1.Event logging in the system

There will be a mechanism to recreate the sequence of events in the system basing on information gathered and stored in the database. MACROBUTTON HtmlDirect </P>

Each event sent by each application in each session will be recorded in the database. It will be possible to use information stored in the database to restore system activity in the given period of time or between given users or applications.MACROBUTTON HtmlDirect </P>

1.1.Session management

A Session is a group of application instances currently working together in collaborative mode. All applications belonging to the same session exchange information and share behavior. How particular application operates in collaborative mode depend on this application.

Each application belongs to a session, even if it is not currently used for collaboration. In such case the session consists of this one application only.MACROBUTTON HtmlDirect </P>

In all Sessions there is one distinguished user which is considered to be a Master. Master of the Session has special privileges of controlling the behavior of applications and/or controlling access of other users to this session. The privileges depend on the application type.MACROBUTTON HtmlDirect </P>

There are three possible actions which may change state of a session:MACROBUTTON HtmlDirect </P>

1.1.1.Creating a session

Session is created when a Participant launches a local application. On the beginning the session contains only this one application instance. The Participant becomes automatically the Master of this session. Other Participants may then connect to this session.MACROBUTTON HtmlDirect </P>

1.1.1.Joining an existing session

There are two possible ways of joining an existing session:MACROBUTTON HtmlDirect </P>

- by launching a new application and connecting it to a session
- by connecting to a session with already existing application instance - in this case the application performs "Leaving a session" and then connects to the other session

1.1.1.Leaving a session

When a Participant leaves a session there are three possible cases which need consideration:MACROBUTTON HtmlDirect </P>

- Application runs in Master-Slave mode and Participant is not the Master - in this case leaving the session means only removing given application instance from the session.
- Application runs in Master-Slave mode and Participant is the Master - in this case the session ends. All participants are removed from the session, the application ends, and session is deleted.
- Application runs in Partner mode - in this case if there is more than two participants the session remains and only one application instance is removed. In case of two collaborating applications, the session ends and both application end.

1.1.Event flow

This section describes flow of events in the system accompanying user actions. The set of actions includes:MACROBUTTON HtmlDirect </P>

- Entering the system
- Leaving the system
- Launching local application
- Joining a session with new application
- Joining a session with existing application
- Launching remote application
- Switching to master mode

1.1.1.Entering the system

Description:

Logging into the system is the first activity each user has to perform before using the system. Logging will be done on the base of two parameters typed by the user: name and password. System will verify these parameters, grant user specific privileges and enable user to perform other actions.MACROBUTTON HtmlDirect </P>

Events:

0User types username and password to Control Application
1Demon sends the information to Central Server (user id, user passwd)
2Data is verified in the database
3Central Server returns user rights to Control Application (user rights)
4Central Server returns description of existing sessions (session list)

41.1.Leaving the system

Description:

After finishing his/her work user has to log off from the system. This will remove him/her from all sessions he/she was participating and close all his/her applicationsMACROBUTTON HtmlDirect </P>

Events:

0User selects exit option from Control Application (exit)
1Demon sends the information to Central Server (exit)
2Central Server updates information about virtual connections
3Demon closes all running client applications.

31.1.Launching local application

Description:

This activity will enable a user to launch an application on the local host. The actions performed by user will not be shared with any other users, unless another user will join this session. These messages will be sent to Central Server and logged in the database. MACROBUTTON HtmlDirect </P>

Events:

1.User chooses application type in Control Application (app type)
2.Information from Control Application is sent to the demon (app type)
3.Demon sends the information to the Central Server, information is stored to the database, a session is created (app type)
4.Central Server generates unique application id and sends it to the Demon (app id)

Standalone Application

5. Demon generates new port number (port number) MACROBUTTON HtmlDirect </P>
6. Demon launches proper application with number of port (port number)

Java AppletMACROBUTTON HtmlDirect </P>

5. Control Application launches Java AppletMACROBUTTON HtmlDirect </P>
6. Demon generates new communication channel MACROBUTTON HtmlDirect </P>
7. Applet connects Local DemonMACROBUTTON HtmlDirect </P>

01.1.Joining a session with new application

Description:

This activity allows a user to launch a new application and connect it to an existing session. Control Application on local host displays all sessions, the user is privileged to join.MACROBUTTON HtmlDirect </P>

Events:

0User chooses the session from Control Application (session id)
1Information is sent to Central server through the Demon (session id)
2Master of this session is notified (user id, session id)
3If master agrees, the application id is added to the session list
4Demon on initiator's machine is notified (accept)
5Demon launches and connects the application

51.1.Joining a session with existing application

Description:

This activity allows a user to connect already running application to an existing session. Control Application on local host displays all sessions, the user is privileged to join.MACROBUTTON HtmlDirect </P>

Events:

0User selects session from the list of session he/she is allowed to join (session id)
1Information is sent to Central server through the Demon (session id)
2Database checks if master for this session exists, if so he/she is notified (user id)
3If master agrees, the application id is added to the session user list
4Demon on initiator's machine is notified (accept)

41.1.Launching remote application

Description:

This action will launch application of a given type on the host of another user currently logged in the system. We assume that before launching application on remote host user will have application of this type running locally. Before launching application remote user must confirm it. After remote application is started it joins the session with the local application on the initiator's host. Initiator has to be master of the given session to perform this operation.MACROBUTTON HtmlDirect </P>

Events:

0User chooses remote launching option in Control Application and remote user id (session id, user id)
1Remote user is notified with a message that application is about to start, he must accept it (app type, master user id)
2Server generates new application id and sends it to the remote user demon (app id)
3New virtual connection is created (app id is added to the session list) and message is sent to remote user's demon (app id)
4Remote user demon generates port number and launches application (port number)

41.1.Switching to master mode

Description:

Changing to master mode may be invoked by a user with proper privileges for an application running in client mode. The application becomes then master of the session and starts transmitting events.MACROBUTTON HtmlDirect </P>

Events:

0User chooses change to master for a given session in Control Application (session id)
1Information goes to the Demon (session id)
2Demon sends message to Central Server (session id)
3Central Server sends information to current session master (user id)
4When master agrees for change, server sends information to all users (session id, user id)
5All other participants are notified (user id, session id)
6Information about master in the database is updated (user id, session id)

7Communication protocol

1.1.Overview

Applications will communicate with each other by participating in the same session. The session will consist of one or more Virtual Connections. Virtual Connection will link application with Central Server. Virtual Connection will consist of connection between Application and Demon and Demon and Central Server. The connections will be implemented as Internet socket connections (The only exceptions are Java Applets communicating Demon by the use of method invocation). Demon to Central Server connection will use TCP stream. Application will communicate with Demon using either TCP streams.MACROBUTTON

HtmlDirect </P>
When Application is started by the Demon it obtain a port number to communicate with. On server side the application will be assigned unique application id number (AID). This number will be known to the Demon which will maintain a table translating AID to communication port numbers. Information coming from the Server is resent to appropriate Applications. MACROBUTTON HtmlDirect </P>
Server identifies the user by the communication link it uses to contact the Server. Then the appropriate session for this particular application is found, and message is resent to the Demons of all other session participants.MACROBUTTON HtmlDirect </P>

1.1.Message structure

1.1.1.Application <-> Demon

From the system point of view messages transmitted between applications will by arrays of bytes. To handle them properly system will have to know their length. Messages sent from application to the demon will consist of three parts:MACROBUTTON HtmlDirect </P>

- message type (inter-application or system message)
- message length
- message body

Message type is required to distinguish between messages sent by ordinary applications and control applications. The latter have to be interpreted by server (e.g., to establish new sessions).MACROBUTTON HtmlDirect </P>
This message structure will be used as well in communication between applications and local demon as between applets and applet demons.MACROBUTTON HtmlDirect </P>

1.1.1.Demon <-> Central Server

To route the messages properly Central Server will have to know AID of application sending the message. The structure of message between Demon and Central server will be the following:MACROBUTTON HtmlDirect </P>

- message length
- application id AID
- message body

Message length is used to read appropriate number of bytes from the input stream. Messages sent between applications have AID greater than zero. In this case server does not have to interpret the message contents. The message is simply re-sent to all other session participants and stored in the database.MACROBUTTON HtmlDirect </P>
AID equal to zero indicates a control message. In such case the contents of the message must be interpreted in the server. The actual attributes constituting this message depend on the message type.MACROBUTTON HtmlDirect </P>

1.1.MACROBUTTON HtmlDirect </P>Central Server - Local Demon protocol

This section describes the communication protocol used between Local Demons and Central Server. It contains: MACROBUTTON HtmlDirect </P>

- MACROBUTTON HtmlResAnchor specification of messages
- MACROBUTTON HtmlResAnchor description of actions undertaken by server
- MACROBUTTON HtmlResAnchor description of actions undertaken by demons

Messages exchanged between Central Server and Local Demons
(s) indicates that the message is received and interpreted by server MACROBUTTON HtmlDirect </P>
(d) indicates that the message is received and interpreted by demon MACROBUTTON HtmlDirect </P>
(c) indicates that the message is received by demon and forwarded to Control Applet. Interpretation takes place in Control AppletMACROBUTTON HtmlDirect </P>
In general there are two types of messages: MACROBUTTON HtmlDirect </P>

- MACROBUTTON HtmlResAnchor application messages - exchanged among collaborating applications. These messages are not interpreted by server. Server resents application messages to all other participants of the session.
- control messages - these are messages sent by demons to change state of the server (establishing of new sessions, adding participants, etc.)

Application messages:
All application messages are encapsulated in one general message type: MACROBUTTON HtmlDirect </P>
(d)(s) EVENT (AID, type, contents) - message carrying information between applications MACROBUTTON HtmlDirect </P>
Control messages:
These messages are not re-sent by the server. The contents of these messages is extracted and interpreted to perform message dependent server actions. MACROBUTTON HtmlDirect </P>
server JOIN AT, SID this command is used to join the session with the application
control app. JASK AT, SID, UID request for joining session sent to master of the sessionMACROBUTTON HtmlDirect </P>
server JANS AT, SID, UID, yes | no answer for JASK sent by master of the sessionMACROBUTTON HtmlDirect </P>
demonMACROBUTTON HtmlDirect </P> COMMAND AT, parameters, AID, SID this command is send to the demon to launch the application
server FINISH AID command sent from the demon after an application terminates MACROBUTTON HtmlDirect </P>
server RJOIN AT, SID. Remote UID message used to launch application on remote host
control app. RASK AT, SID, local UID message to acknowledge user on remote host if he/she wants the application to be launched
server RANS AT, local UID, remote UID, SID, yes | no answer for RASK MACROBUTTON HtmlDirect </P>
control app. SUPD add | remove | master | login | logout, SID, AT, UID, username
server MASTER SID request for master MACROBUTTON HtmlDirect </P>
control app. MASK SID, UID request for the previous master MACROBUTTON HtmlDirect </P>
server MANS SID, UID, yes | no answer from previous master
server LOGIN username, passwd login to the system MACROBUTTON HtmlDirect </P>
control app. LOGACC username, UID, privileges login confirmation MACROBUTTON HtmlDirect </P>
server LOGOUT UID logout from the system
Server actions
We use the following convention: description of actions is provided with SQL-like language with several extensions. MACROBUTTON HtmlDirect </P>

- SEND means sending message back to the demon which sent the message currently processed. MACROBUTTON HtmlDirect </P>
- SEND_TO(UID) means sending a message to the demon working with user of identifier UID. MACROBUTTON HtmlDirect </P>
- SEND_TO_ALL means sending a message to all demons. MACROBUTTON HtmlDirect </P>

EVENT (AID, contents) MACROBUTTON HtmlDirect </P>
1.SELECT SID FROM APPLICATIONS WHERE AID=AID
2.SELECT AID FROM APPLICATIONS WHERE AID!=AID AND SID=SID
3.INSERT INTO EVENTS (AID, contents, timestamp=getTime());
4.FOR (aid IN AID) SEND_TO(getUID(AID), EVENT(AID, contents));

JOIN (AT, SID) MACROBUTTON HtmlDirect </P>
1.if (AT in SID for UID(UID)) stop

2.if (SID==0)
3.SID = generate new SID;
4.MASTER=UID(UID);
5.INSERT INTO SESSIONS (SID, AT, MASTER);
6.generate unique AID
7.INSERT INTO APPLICATIONS (AID, SID, UID(UID)); ;
8.SEND (COMMAND(AT, params, AID, SID));
9.SEND_TO_ALL (SUPD(add, SID, AT, UID(UID), username);
10.SEND_TO_ALL (SUPD(master, SID, 0, UID(UID), username);
11.stop
12.SELECT mUID = master FROM SESSIONS WHERE SID=SID;
13.SEND_TO (mUID, JASK(UID, SID));

JANS (AT, SID, UID, yes | no)MACROBUTTON HtmlDirect </P>
1.if "no" stop
2.generate unique AID
3.INSERT INTO APPLICATIONS (AID, SID, UID); ;
4.SEND_TO(UID, COMMAND(AT, params, AID, SID));
5.SEND_TO_ALL (SUPD(add, SID, AT, UID, username);

FINISH (AID) MACROBUTTON HtmlDirect </P>
1.SELECT s=SID FROM APPLICATIONS WHERE AID=AID
2.DELETE FROM APPLIATIONS WHERE AID=AID
3.SELECT M=S.master FROM SESSIONS WHERE SID=s;
4.if (M==UID(UID)) DELETE FROM APPLICATIONS WHERE SID=s;
5.if (!SELECT count(*) FROM APPLICATIONS WHERE SID=s)
6.DELETE FROM SESSIONS WHERE SID=s
7.SEND_TO_ALL (SUPD(remove,SID, 0, 0, ""));
8.stop;
9.SEND_TO_ALL (SUPD(remove, SID, UID(UID));

RJOIN (AT, SID. remote RUID) MACROBUTTON HtmlDirect </P>
1.SELECT C=count(*) FROM APPLICATIONS WHERE SID=SID AND RUID=UID
2.if (C>0) stop
3.SEND_TO(RUID,RASK(AT, SID, UID(UID)));

RANS (AT, local UID, remote RUID, SID, yes | no) MACROBUTTON HtmlDirect </P>
1.if (yes)
2.generate unique AID
3.INSERT INTO APPLICATIONS(AID, SID, UID(UID))
4.SEND (COMMAND(AT, params, AID, SID))
5.SEND_TO_ALL (SUPD (add, SID, AT, RUID, rusername))

MASTER (SID) MACROBUTTON HtmlDirect </P>
1.SELECT M=MASTER FROM SESSIONS WHERE SID=SID
2.if (M==UID(UID)) stop ;
3.SEND_TO (M, MASK(SID, UID(UID));

MANS (SID, UID, yes | no) MACROBUTTON HtmlDirect </P>
1.if (yes)
2.UPDATE SESSIONS SET master=UID WHERE SID=SID;
3.SEND_TO_ALL(SUPD(master, SID, AT, UID, name);

LOGIN (username, passwd) MACROBUTTON HtmlDirect </P>

1.SELECT p=user password, u=UID FROM USERS WHERE user name=username;
2.if (p==passwd)
3.SEND (LOGACC (username, UID, privileges));
4.INSERT INTO PARTICIPANTS (UID,getHost());
5.SEND (INFORMATION ABOUT CURRENT SESSIONS AND LOGGED USERS);
6.SUPD(add, SID, AT, UID, username) - for each user in session
7.SUPD(master, SID, AT, UID, username) - for each master of the session
8.SEND_TO_ALL(SUPD(login, 0, 0, UID(UID), username(UID))) - new user logged
9.else SEND (LOGACC(username, 0));

LOGOUT (UID) MACROBUTTON HtmlDirect </P>
1.SELECT a=AID FROM APPLICATIONS WHERE UID=UID;
2.for (AID in a) FINISH(AID);
3.DELETE FROM PARTICIPANTS WHERE UID=UID;
4.SEND_TO_ALL(INFORMATION ABOUT FINISHED SESSIONS AND LOGGED OUT USER);
5.SUPD(remove, SID, AT, 0, "") - for each session closed
6.SUPD(remove, SID, AT, UID(UID), name) - for each session from which user UID is removed
7.SUPD(logout, 0, 0, UID(UID), username(UID)) - user logs out

Demon and Control Applet actions

COMMAND (AT, parameters, AID, SID) MACROBUTTON HtmlDirect </P>
1.generate free port number
2.map AID to port number
3.find physical path to application of type AT
4.launch application with port number as one of the parameters

JASK (AT, SID, UID) MACROBUTTON HtmlDirect </P>
1.display message box notyfiying that user UID wants to connect session SID
2.get user confirmation
3.SEND (JANS(UID, SID, yes | no)

SUPD (add | remove | master | login | logout, SID, AT, UID, username)MACROBUTTON HtmlDirect </P>
1.update information about Participants and Sessions in Control Application

EVENT (AID, contents) MACROBUTTON HtmlDirect </P>
1.find port number mapped for the AID
2.write contents to the outgoing queue on this port number
RASK (AT, SID, UID) MACROBUTTON HtmlDirect </P>
1.display message box that user UID wants to launch application AT in session SID
2.get user confirmation
3.SEND (RANS(AT, UID(UID), UID, SID, yes | no))

MASK (SID, UID) MACROBUTTON HtmlDirect </P>
1.display message box that user UID wants to become master of session SID
2.get user confirmation
3.SEND (MANS(SID, UID,yes | no))

LOGACC (username, UID) MACROBUTTON HtmlDirect </P>
1.if (UID!=0) show the information that user login was accepted else refuse login;

1.1.Interfaces

The interfaces described in this paragraph are used by application programmers to connect their application into the system.

1.1.1.Java interfaces

Figure 14 Classes used for interfaces

```
public class Message {
      private byte[] content;
      private int type;
      public Message(int type,byte[] c);
      public byte[] getContent();
      public int getType();
}

public interface Base {
      public void register(int AID);
      public void send (Message m);
      public void receive(Message m);
}

public class AppBase implements Base {}

public class AppletBase extends Applet implements Base {}

public class CABase extends Applet implements Base {
      public void register();
}
```

Class Message is the base class for implementing messages between parts of the system. All applications will have to implement methods of extracting data from this class.
Interface Base defines two methods - *register* is used to connect to the Demon, *send* provides means to send the message. Classes implementing interface Base will implement these two methods.
AppBase class is used to connect stand-alone Java applets with the Demon. It will create socket connection in its register method. Method send causes sending message to the Demon through previously opened socket.
AppletBase provides connection with Demon for applets. In its register method it will register itself in the plug-in by calling its method. The send method causes passing message to the Demon.
CABase is a base class for the Control Application, which is always started first in the sytem.

1.1.1.C interface

Similar interface will be provided in C language.

The C inteface will provide the following structures and functions:MACROBUTTON HtmlDirect </P>

```
struct Event {MACROBUTTON HtmlDirect </P>
   int type;
   int len;
   char *data;
}MACROBUTTON HtmlDirect </P>
int register(int AID, int port);MACROBUTTON HtmlDirect </P>
void sendEvent(int sockfd, struct Event *e);MACROBUTTON HtmlDirect </P>
(struct Event *) receiveEvent(int sockfd);MACROBUTTON HtmlDirect </P>
(struct Event *)createEvent(int len, char *data);MACROBUTTON HtmlDirect
</P>
int getType (struct Event *e);MACROBUTTON HtmlDirect </P>
int getLen (struct Event *e);MACROBUTTON HtmlDirect </P>
int getData (struct Event *e);MACROBUTTON HtmlDirect </P>
void destroyEvent( struct Event *e);MACROBUTTON HtmlDirect </P>
```

Function register will connect application to the Demon by establishing socket connection and transmitting AID number. It will return socket descriptor to be used by other functions.MACROBUTTON HtmlDirect </P>

Function sendEvent will be used to send messages to the system. It will take socket descriptor and message wrapped in Event sructure as parameters.MACROBUTTON HtmlDirect </P>
Function receiveEvent should be called each time we want to receive a message.MACROBUTTON HtmlDirect </P>
Functions getLen and getData are used to retrieve the content of Event structure.MACROBUTTON HtmlDirect </P>
Function destroyEvent is used to destroy the Event structure and free the memory.MACROBUTTON HtmlDirect </P>

1.1.1.Example of interface use

Let us consider a simple case study of porting one, initially non-collaborative application into our environment.

Current application description

Current Search is built entirely in HTML with use of forms as a GUI. It gets user input describing search criteria, sends it to the HTTP server where information is processed by the appropriate CGI program, then the results are sent back to the client browser.

Collaboration scheme

The collaboration scheme for this application assumes that Search applications running on different computers will be connected in a session. One of applications will be acting as a master and will get user input for search. After submitting this information all connected Search applications will display the same result.

Steps to migrate into collaborative environment

We define Java applet providing user with the Graphical User Interface allowing him/her to input the search criteria. In the simplest case it will consist only of an text input area and a button for submitting. The applet extends provided class AppletBase which gives means to connect the applet to the collaboration system. Our applet gets user input in the text input area and after pressing the "submit" button performs the following operations:

- on the base of user criteria proper URL is formed and Java method *showDocument* is invoked. This causes browser to send URL to HTTP server where CGI program is executed and results are displayed in client browser
- at the same time system message is created and sent to the demon. Our system takes care of distributing it to all the applications being connected to the same session. Application after receiving the message invokes the same method as if it received user input

In order to do so our application will have to make the following steps:

- its main class should be subclass of our interface class AppletBase
- after staring applet should call *register* method to register itself in the Demon
- in order to send any information applet must change into table of bytes, create new Message object containing these bytes and send it with use of *send* method
- applet must implement method *receive* specifying actions that will be undertaken after receiving a message. In our case it should simply produce URL string from the array off bytes and invoke loading the document on the base of it

1.1.Database scheme

Database connected to the Central Server will store all the persistent information about users, user privileges, existing sessions and messages transmitted through the system. To keep all this information the following database scheme is needed:MACROBUTTON HtmlDirect </P>

Sessions

SID_	number	session id, primary key
AT	number	application type
master	UID	current master's id

Applications

AID	number	application id, primary key
SID	number	session id
UID	numberMACROBUTTON HtmlDirect </P>	user id

Participants

UID number user id, primary key
host stringMACROBUTTON HtmlDirect </P> host name

Users
UID number user id, primary key
user name string user name in the system
user password stringMACROBUTTON HtmlDirect </P> user password
Events
AID number application id
message array of bytes message contents
timestamp timestampMACROBUTTON HtmlDirect </P> message timestamp
Privileges
UID number user id, primary key
AT number application type, primary key
local launch boolean right to launch an application locally
remote launch boolean right to launch an application for another user
joining session boolean right to join the session
changing to master booleanMACROBUTTON HtmlDirect </P> right to change to master mode
AppDetails
AT number type of application, primary key
app_name string application name
master booleanMACROBUTTON HtmlDirect </P> determines if application requires master

1.Requirements for the project

This section describes use of the collaboration system in the Collaborative Interaction & Visualization project pending in NPAC. The main goal of the whole project is to develop a prototype of the Command & Control Center for military purposes. Command & Control Center is a tool used by military authorities to trace current situation, recognize threats and to give the proper command in order to neutralize them. The C&C system consists of many application destined to perform specific tasks, e.g. tracing maps. In our case we implement C&C as a set of various applications running on different hosts. They will be connected and managed by our collaboratory system.
In case of C&C there are two main aspects to be considered

- applications to be part of the system - it has to be determined what kind of collaboration they will provide
- providing information specific for military C&C application - to demonstrate the system in action sample simulator is needed to imitate real world events

1.1.Existing applications

This is the current list of applications to be integrated into C&C system.

1.1.1.Chat

Description:
Chat application may be used to exchange textual information among participants. Chat provides on-line connections. Users can type text which is immediately sent to other participants. It appears on screens of all users which are connected to this particular chat session. One user may participate simultaneously in multiple chat sessions.MACROBUTTON HtmlDirect </P>
Implementation platform:
Java applet, NetscapeMACROBUTTON HtmlDirect </P>
Collaboration modes:
Chat can run only in collaborative mode. Number of chat session participants ranges from two to the number of users logged to the system. Multiple simultaneous sessions are possible.MACROBUTTON HtmlDirect </P>
No master user.MACROBUTTON HtmlDirect </P>

1.1.1.Whiteboard

Description:
Whiteboard application behaves in similar way as chat, but instead of text messages drawings are exchanged between users. It uses the same paradigm of sessions.MACROBUTTON HtmlDirect </P>
In whiteboard each user has a window and a set of tools enabling him drawing pictures. These may contain features as pen, text, basic two dimensional shapes. As the result of collaborative work one picture is created. All participants of the session participate in the process of creating the drawing. Whiteboard will be most likely used with another collaboration application as chat or videoconferencing.MACROBUTTON HtmlDirect </P>
Implementation platform:
Java applet, NetscapeMACROBUTTON HtmlDirect </P>
Collaboration modes:
Whiteboard may run as well in single as in collaborative modes. In collaborative mode number of participants is not limited: it may range from two to number or users logged into the system. Whiteboard sessions are independent of any other collaborative sessions.MACROBUTTON HtmlDirect </P>
No master user.MACROBUTTON HtmlDirect </P>

1.1.1.Visible Human

Description:
The task of Visible Human application is to provide users with images and three-dimensional models of human body. It may be used in examining and describing casualties from the hypothetical crisis place. In collaborative mode one person point at various parts of human body and the other participants may see the same places.MACROBUTTON HtmlDirect </P>
Implementation platform:
Java applet, NetscapeMACROBUTTON HtmlDirect </P>
Collaboration modes:
The Visible Human application may run as well in single as in collaboration modes. In single mode the application gives a single user access to the information databases and provides him/her with appropriate models or images. In collaboration mode there is one user which serves as master. This user behave exactly like in single mode. The rest of the participants may observe the results in the same way as they appear on the screen of the master user.MACROBUTTON HtmlDirect </P>

1.1.1.GIS - Geographic Information System

Description:
Geographic Information System is a tool which may be used to visualize various kinds of geographic-related information. The visualization may be performed in two- or three-dimensional modes with some information in textual form. Various kinds of information may be visualized depending on the user demand. Main types of information are:MACROBUTTON HtmlDirect </P>

- three dimensional model of the terrain
- satellite images, maps, demographic information, weather prediction as two dimensional maps or overlays on three dimensional terrain model
- three-dimensional models of clouds
- three-dimensional models of electromagnetic models of aircrafts, radars, etc.
- textual information about major places in the model, e.g. cities, rivers, islands, etc.

Implementation platform:
C++, Open Inventor libraryMACROBUTTON HtmlDirect </P>, java applets MACROBUTTON HtmlDirect </P>
Collaboration modes:
Geographic Information System may run in single as well as in collaborative modes. In single mode there is one user operating the system. In collaborative mode there is one master user operating like in single mode and any number of other participants which may observe exactly the same results as the master user.MACROBUTTON HtmlDirect </P>

1.1.1.Videoconferencing system

Description:
This system will be used to provide participants with real time video and audio

communication.MACROBUTTON HtmlDirect </P>
Implementation platform:
C, Netscape plug-in
Collaboration modes:
Video conferencing may run only in collaboration mode

1.1.1.Search system

Description:
The search system will be used to provide participants with access to the news data. Participants will have possibility to search information by keywords, dates and so on. MACROBUTTON HtmlDirect </P>
Implementation platform:
Netscape, CGI, PL-SQL
Collaboration modes:
The search system may run in single as well as collaborative modes. In single user mode it simply provides a user with access to the data. In collaborative mode it allows also to show the same results on other participants screens.MACROBUTTON HtmlDirect </P>

1.1.Military scenario and simulation

In order to use our system for CIV scenario the following steps have to be accomplished:
MACROBUTTON HtmlDirect </P>

- introduction of an event source driving the simulation
- specialization of the architecture to provide scenario-specific behaviors

We assume that simulation events will be generated by special program called Simulation Demon (SD). Its purpose will be to deliver the system applications messages simulating outside world behavior e.g. enemy, president. To fulfill this task Simulation Demon will support the notion of virtual applications. From the system point of view it will behave as if there were applications of each type connected to it. Virtual applications will be connected in session with applications of the same type on the other hosts. They will be sources of application-specific messages driving the simulation. For instance virtual application of GIS type will provide other GIS with position and movement data of all airplanes. MACROBUTTON HtmlDirect </P>
The commands in the simulation will be executed through the Control Application of each Participant. All CA will be connected in one session. On the Simulation Demon there will be virtual CA which will catch the commands and run the simulation accordingly. MACROBUTTON HtmlDirect </P>
Scenario provided by simulation engine may be executed in two modes: MACROBUTTON HtmlDirect </P>
In the simulation mode SD provides system with events simulating enemy behavior, e.g. events about position of enemy aircraft. It is the responsibility of participants of the simulation to launch proper applications and invoke the commands. MACROBUTTON HtmlDirect </P>
In the demonstration mode all actions are performed automatically. SD not only sends events describing enemy behavior, but also simulates participant's response by launching applications and simulating work of the emergency staff. MACROBUTTON HtmlDirect </P>
To accomplish these task specific constraints must be assumed:

- In simulation mode users are prevented from launching applications locally. They can only connect to the sessions created by SD.
- In demonstration mode SD sends messages to imitating behavior of the simulation participants.

Let's consider the following scenario: MACROBUTTON HtmlDirect </P>

- SD starts simulation. It starts generating enemy aircraft data through its virtual GIS. At the same time from virtual chat the message about discovering the aircraft by radar is sent
- Participants of the simulation may start the appropriate applications to verify the situation.
- Through the conference channels decisions are being made.
- From the Control Application user starts sending commands to the fighter to intercept the target.
- Information to the fighter is interpreted by SD which sends new events to the GIS
- When user decides to shoot the enemy aircraft, GIS ends the simulation.

To implement Simulation Demon one needs to know all the application messages that may be sent.

MACROBUTTON HtmlDirect </P>Specifying these messages is beyond the scope of this description.

1.Conclusions and future development

Systems basing on the World Wide Web are relatively new technology, yet their growth during the last year was tremendous. For instance *Gamelan*, the most popular repository for Java, indicates currently about 3500 entries, including complex systems e.g. http servers and multi-user network games. The industry recognized already importance of this phenomenon and strongly supports the growth of Web technologies. Recently a consortium of market leaders - Sun, IBM, Oracle, Netscape and Apple announced a common profile for the "Network Computer" - a new computer device , which would run in the Internet environment [13]. Also Microsoft promotes its own Internet technology called ActiveX. Both activities are said to interoperate in the future creating a brand new paradigm in computing.

Facing this tendency it is very important for system developers to keep the place in the mainstream of new technology development and provide solutions compliant with widely accepted standards. The system presented in this document makes heavy use of the most popular technologies, which are guaranteed to be still in use in the future. This allows it to be gracefully developed as new enhancements of standards emerge. While depending on new technologies, it still allows to incorporate legacy applications created for non-network environments.

During writing this document the system was still at the development stage with three programmers working on it. Although not finished, the existing part of the system (Central Server and prototype Demon) allowed me to create a sample collaborative application coordinating documents showed by Web browsers connected to the system. This experiment confirmed the feasibility of the whole project.

As a side-effect of the system development several interesting ideas have emerged. One worth mentioning is the use of Java applet to provide Graphical User Interface for Netscape plug-ins. Traditionally these GUIs were written in C with use of native functions for each architecture. Using Java not only greatly simplified creation of a GUI, but also made it fully portable across different platforms.

Currently, because of heterogeneous environment, the system only uses most primitive communication means, sending messages as tables of bytes. I presume that, with the network environment evolving toward greater interoperability between its key technologies, all legacy applications will be eventually replaced by new, fully network-aware software. Having this in mind we want to introduce new communication system in the future version of the system. It will be based on remote object invocation and object passing.

Introducing these features into the system will change it into fully distributed platform in which whole parts of the code will be freely transported among client sites.

1.Bibliography

[1].Gosling, B. Joy, and G. Steele. The Java Language Specification. Addison-Wesley Developers Press, Sunsoft Java Series, 1996.

[2].Gosling, F. Yellin, and the Java Team. The Java Application Programming Interface. Addison-Wesley Developers Press, Sunsoft Java Series, 1996. .

[3].Dick Oliver. Netscape2 Unleashed. Sams.Net, 1996.

[4].G.Hamiltion and R. Cattell. JavaSoft JDBC: A Java SQL API Version 1.00
`http://splash.javasoft.com/jdbc/`

[5].The Internet Application Framework: A White Paper
`http://home.netscape.com/comprod/server_central/tech_docs/oif.html`

[6].JavaScript Guide
`http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.html`

[7].The LiveConnect/Plug-in Developer's Guide
`http://home.netscape.com/eng/mozilla/3.0/handbook/plugins/index.html`

[8].K.Olszewski, R. Wojciechowski, R. Trzaska, L. Lecornu, Cz. Jedrzejek and T. Major. Netscape Plug-in for Wavelet Compressed Images. Presented at Dual-Use Technologies & Applications Conference. Syracuse, June 1996.

[9].J. Bulawa. Integration of Multimedia Collaboratory Environment with Web Browser. Master Thesis. EFP Poznan 1996.

[10].NCSA Habanero Project
`http://www.ncsa.uiuc.edu/SDG/Software/Habanero/`

[11].M. Chandy, A. Chelian, B. Dimitrov, H. Le, J. Mandelson, M. Richardson, A. Rifkin, P.A.G. Sivilotti,

W. Tanaka, and L. Weisman. A World-Wide Distributed System Using Java and the Internet . Presented at High Performance Distributed Computing (HPDC-5) - Focus Workshop on Multimedia and Collaborative Environments, Syracuse, August 1996. Also available as Caltech CS Technical Report CS-TR-96-08 and CRPC Technical Report Caltech-CRPC-96-1.

[12].OpenDVE Architectural Overview. InSoft, 1995

[13].Network Computer Reference Profile.
http://www.nc.ihost.com/nc_ref_profile.html