

Development of Internet Services Based on Pure JAVA Technology

Karim Sbata¹, Pierre Vincent²

¹ENIC (Villeneuve d'Ascq - France)
Morocco

²INT (Evry – France)
France

Abstract. This article is about the developpement of new Internet services at the Network and Computer Science Departement of the National Institute of Telecommunications (INT) of Evry. It deals mainly with a project related to an applications server, purely Java and Web (HTML) oriented, which main objective is to offer a set of online applications to any Web user (even mobile in the future), and a multi-point multi-media architecture (client/server), also purely Java, aiming to allow connected users to exchange any type of data (text, audio, video, etc), independently from their transport protocol (TCP or UDP).

Keywords: Multi-Media, Hyper-Media, Application Server, Multi-Pointing Architecture, Multi-Terminals Clients, JAVA Technology.

1. Introduction

Traditionally, the worlds of data-processing, telecommunications and multimedia networks were regarded as distinct, even incompatible for some purists.

Indeed, in addition to the economic war that delivered (and still deliver!) the data processing and telecom lobbies, the philosophies of these various worlds were radically different: whereas the data-processing world privileges a statistical (optimal) use of the network resource, a free "best effort" service (for a right resource sharing between the users) and putting the complexity on the terminal (to sale its more and more powerful computers), the world of telecommunication prefers a deterministic reservation of resource (to ensure a quality of service), a service with variable QoS (proportionnall to the price payed by the user) and putting the complexity on the network itself, making it thus accessible from very basic terminals.

For a few years however, these two worlds have began to converge, thanks in particular to the progress made on digital technology (A/V compression, entirely digital telecom networks (e.g. GSM)) and networks performances (growing reliability and flows). It indeed proved that each one of these worlds would be soon technologically ready to ensure the services of the other, removing thus any barrier between these two fields formerly so hermetic.

The current tendency is then multimedia and multiservices networks (telephony, data transfer, A/V, etc), accessible from different terminals (PC, mobile phone, TV, and in a slightly more remote future, domestic machines (thanks to JINI technology)).

In this context, the new Internet applications will tend to be as portable as possible (100% Java) and to offer the most independent possible (from the terminal used) user interface (UI).

The Network and Computer Science Departement of the ENIC (Lille) and the INT (Evry) are currently developing a server of applications purely Java, accessible from any Web navigator (ensuring then an independence from the terminal) and whose objective is to offer complex

processing on simple parameters (e.g. URLs, files, etc), as well as a multimedia multipoint client/server architecture, also purely Java, which allows exchanges of all types of information (texts, images, audio, etc) between several users.

The object of this article is to detail these two projects (which can be gathered in one, the multi-point architecture being able to include the server of applications), to present their current state of advance (i.e. in July 2000), and to evoke their principal axes of evolution.

2. The Server of Applications

2.1. Principle

A server of applications is a Web server (thus based on HTTP) which gives access to various programs, runnable on line. Its main interest is to exempt the client to download on his terminal the executable code, to install it and to launch it. This last step is indeed very often penalising, for several reasons : first of all, the downloaded software is generally used very partially and only a small number of times; moreover, the "server" machine are often more powerful than the "client" machine and can thus be more effective for some applications (in particular in the field of multimedia); finally, it should be noticed that some terminals with very limited resources (e.g. mobile phones) can neither store nor load code.

The server of applications we are developing aims thus to provide an access to an open database of applications (i.e. allowing any programmer to contribute to it) to various users.

For that, it was first of all necessary to define a generic user interface, in order to offer the same possibilities to any client, whatever its performances. The choice of HTML language (and/or its derived languages: XML, HDML, WML [1], etc) appeared adequate.

Moreover, to be able to work with an open database of applications, it was necessary to define a standard for the applications, in particular for their input/output (to allow a single generic treatment for all the applications).

In order to avoid using directly the operating system to launch each application (from a command line or by using dynamic libraries), Java was selected like exclusive programming language. Indeed, in addition to its portability, Java offers all the advantages of an object-oriented language, in particular that of being able to instanciate and load any class from any other, to use its public fields and methods, etc.

Once these choices have been done, we had to put them into practice. The object of the following section is to give more details about this.

2.2. Implementation

The first stage was the development of the user interface. This one had to meet two conditions: first, to allow the user to choose an application and to send the associated parameters; to allow the reception of the result.

In order to have a generic interface, we standardized the form of the input parameters and the results: all the treatments are done either on files or on URLs; the results are always returned in form of files. We should notice that this standards are valid only for the traditional interface in HTML (the only one currently implemented). An interface in WML for mobile phones is under study. In this case, the input parameters will be URLs (to be treated) and an e-mail address to receive the result.

To develop such an interface in HTML was rather easy. Indeed, this language allows, thanks to its form functionalities, to send all kinds of information to the server. The result of the treatment will be simply returned in the HTTP message sent as a response to the request associated to the form.

The only difficulty encountered was the transfer of a file from the client to the server, without using FTP or any additional software on the client side (thing that would be in total contradiction with the philosophy of the project). This problem was rather quickly solved by using a relatively ignored resource of HTML's FORM tag: the input of type file (<INPUT TYPE = 'file' >), defined in [2].

This tag is recognized by the majority of recent Web navigators (from 4.x versions of Netscape and Internet Explorer). It allows the user to choose a local file (via an open file dialog box) and to send it to the Web server in a request HTTP message (using method POST), specifying a content-type "multipart/form-data". It was thus necessary to implement within the server a module dealing with this type of messages.

Once the problem of the interface solved, we oriented our efforts to the standardization of the applications. Like it was said previously, we first decided to use exclusively the Java language for programming, in order to have a total portability, on the level of the server itself (i.e. it will be runnable on any machine having a JVM) but also on the level of the applications associated. Indeed, as these last will be organized as an open database, the portability is necessary to allow all developers to contribute, whatever their programming environment (mainly their operating system).

The use of Java also has the advantage of simplifying the interactions server-applications, in particular by avoiding the use of the operating system for loading an application required by the client.

In addition to choosing the programming language, it was also necessary to define a standard for the inputs/outputs, to avoid the definition of a new treatment for each addition of a new application.

Thanks to Java, that could be done without major difficulties. Indeed, this language implements in a remarkable way the concept of streams (via the *java.io* package [3]). Thus, in the same way that CGI scripts communicate with a Web server via standard I/O streams, the applications will communicate with the server via an *InputStream* and an *OutputStream* (more exactly their generic data subclasses, the *DataInputStream* and the *DataOutputStream*).

To be integrated to the server, each application (class) will then have to contain a constructor with at least two parameters: a *DataInputStream* and a *DataOutputStream*. Other parameters are detailed in the first appendix.

We can notice that using streams of bytes allows all kinds of applications, treating any data (e.g. processing of an image sent by the client, conversion from an audio format to another, etc).

Currently, the server of applications implements only two completely operational applications: Calc, a simplified spreadsheet, which allows making arithmetic operations on numerical HTML tables, and Map, an editor of synopses, which allows extracting a synopsis from a standard HTML document (i.e. whose titles and subtitles use headers (tag <H>)).

Others are currently under development (e.g. organization of multimedia documents form, utilities of compression for directories, etc).

We will give more details in the following section about the prospects for evolution of the server.

2.3. Prospects for Evolution

The evolution of the server concerns mainly its applications, or more exactly the fields they are associated to. Currently, the developments are done through two main axes: the installation of an online HTML oriented toolbox (including in particular the applications Calc and Map previously described) and the development of multimedia utilities (e.g. organization of multimedia documents in a diaporama form).

The main interest of the HTML toolbox is to provide a set of online utilities which can be used either directly by the user, or by other applications. For example, in an Intranet environment, the Map application can be used by a dynamic application of type forum (internal news for example).

The development of multimedia applications, could in particular offer converters of format, compressors (audio, fixed or animated images) or applications of 3D rebuilding. Like the HTML toolbox, these tools could either be used directly, or by intermediate programs.

Other axes of development will probably appear soon. Indeed, the applications database being opened to any contribution, it will be able to evolve/move in the most various directions, with the liking of the imagination of the developers.

3. The Multipoint Server

3.1. Principle

There is several ways of conceiving a multipoint architecture. Indeed, according to the use it will be dedicated to, this one can vary from a centralized server model (for a reduced use, on the level of the number of potential users and on the level of the exchanged data) to a group address model (protocol IGMP [4], for newsgroups aiming to be unlimited), passing by multi-servers models.

It is noticed however that, in spite of their differences, all these architectures require the implementation of certain common functionalities: to allow the users to join the group or to leave it, to send to all the clients every new contribution, etc.

The architecture we are developing belongs to the first and last categories; it is based on a centralised server mode, which can be extended to an interconnected servers environnement to increase its capacity. Indeed, being intended initially to work in an Intranet or limited Internet environment, it does not require a complex and powerful implementation for the management of clients, broadcasting, etc. In this case, a single server is sufficient. But its extension to a larger environment needs more capacity and flexibility. In this case, a distributed model is recommended. This is why our model allows interconnecting servers (tunneling).

This architecture is based on a simple client-server model, implementing a broadcast functionality on the server. Actually, there is not only one server but three ones: two generic servers, one dedicated to UDP clients and another to TCP ones, and a specialised TCP server, implementing a basic protocol, dedicated to signalling and some specific data applications.

These three servers work mainly the same way, with few differences due to their transport protocol and their complexity.

At starting-up, they are put in a listening state on a specific port. When receiving a connection request (wich is automatically generated for TCP but not for UDP; actually, an UDP client has to send a datagram to be recognized by the server), they establishes a session with the client (via a TCP or UDP socket). Each new session is managed by a new process in the server side (the main process remains dedicated to listening). Once the communication established, the client can then send its contributions to the server. This one will broadcast them to all the connected clients (i.e. UDP or TCP clients).

As these contributions can be from very different types (text, binary files, audio, video, etc) for some applications, we developed a small applicative protocol between the client and the server and/or the clients [appendix 1].

Concerning the user interface, we wanted to preserve as possible a context of Web navigation as for the server of applications. However, in this case, the client has to be able to make a little more complex operations (to send data of different types, to list all the connected clients, etc). We were thus obliged to define more powerful interfaces than simple HTML forms.

3.2. Implementation

The first stage was the development of a server implementing broadcast functionalities, able to establish and manage several connections simultaneously. For this, Java offers very useful packages and classes.

For our network related part, we used the *java.net* [5] package. It contains in particular the *ServerSocket* (server side) and the *Socket* (client side) classes, which allow listening (*ServerSocket*) and establishing TCP connections. For UDP, it provides two complementary classes: the *DatagramSocket* and the *DatagramPacket*, which are used in both the server and the client sides.

To manage simultaneous communication sessions (sockets), we associated a Thread to each one of them. By using the ThreadGroup class, we can not only manage several connections (i.e. to test the alive threads, to list them, to put a threshold for the number of simultaneous connections, etc) but also to make broadcasting/multicasting with great facility. Indeed, it provide a very useful notion of group, which can be used to create users groups.

It was then necessary to develop an interface able of powerful functionalities (e.g. implementing the basic applicative protocol), without obliging the user to install any software on its machine. The choice of Java applets was then quite natural. Indeed, they allow the client to make intelligent processing from a simple Web navigator. On the other hand, this requires the use of powerful machines, excluding thus terminals like mobile phones or PDA. Other solutions could then be developed to ensure an independence of the terminal. One of them could be using gateways.




It was also necessary to develop a basic protocol of communication, to allow signalling commands and multi-media applications to recognize the type of transmitted data. This protocol is detailed in appendix 1.

However, it is still under development and will probably be improved by defining new types, in order to face the future evolutions of the model (introduction of multimedia streaming, etc).

3.3. Prospects for Evolution

In its current state of development, our model allows any users, via TCP connections or UDP datagrams, to exchange any type of data (ASCII, audio/video streams or messages, etc) with a network of users groups (network of inter-connected servers)

The main objective of the project has then been reached. However, there are still several improvements that can be done.

One of them is the definition of a multi-levels hierarchy ("user  user group" or "user  secondary user group  primary user group") to make real multicasting. Indeed, the current architecture allows multicasting based on connection points (servers) and types (TCP or UDP). To be completely multicast, the model has to permit users to define their own groups.

Another improvement can be done: it concerns the basic protocol detailed in appendix 1. In addition to the definition of new types of data, this protocol can be enriched by the possibility of conveying information concerning the users (profiles) or by increasing the set of signalling commands, to allow distant managing for example.

4. Conclusion

The two studied projects follow the current tendency of evolution of the networks and more particularly of the Internet world. Indeed, they amongst other things propose to find solutions to the problems of the independence of the terminals or resource sharing between user groups.

As they are technologically advanced, these projects are thus likely to evolve/move in directions sometimes still unknown. It is the case for example of the Web in a mobile context, which is not yet completely standardized (WAP standard).

The evolution of these projects can even become independent of the initial objectives, in particular for the server of applications. This one being based indeed on an open applications database, it will evolve in the direction the contributors will give it.

Appendices

Appendix 1: Protocol used by the multi-point architecture

The server side of our architecture is made up three twined servers: two raw servers, one for UDP and another for TCP, and a high-level TCP one. This last implements a basic protocol, specifying the type of data transmitted. It is very useful for multimedia (multi-data) applications and for signalling.

It is implemented right over TCP and has an ASCII header. A example of message can be as follows:

```
code:L CRLF
[byte1] [byte2] [byte3] ... [byteL]
```

L is the length (in bytes) of the message and code identify the type of the message.

Currently, the defined codes are:

"acp",	used to notify a pseudo (user identifier) acceptance
"lcd",	for listing all the sessions and their statistics
"end",	sent by a client to end its session
"err",	for error messages
"hlo",	for identificating a client (pseudo)
"lms",	used to update the members list after an inter-connection (request)
"a2l",	used to update the members list after an inter-connection (response)
"mbi",	for obtaining the members list
"url",	specify a URL message
"tst",	for testing loops (after an inter-connection)
"txt",	specify a text message
"ado",	specify an audio message (PCM at 8 KHz)

Remark: There is two types of codes: signalling commands and data types. As the server is not concerned by the data types, the user can define its own types.

Appendix 2: Class model for the server of applications

Our server of applications is actually a server of Java classes. It can load and run any class corresponding to a defined standard, which is detailed below:

```
// A standard class

package StandardClass;

// imports
import ...;
// end imports

public class StandardClass extends WhatYouWant
    implements SomethingElse{

    public StandardClass ( DataInputStream dis,
                          DataOutputStream dos,
                          String url,
                          String opt1, ..., optN){

        // starting treatments
        ...
        // ending treatments
    }

    // methods
    ...
    // end methods
}
```

To use this class, a user has to type the following URL in his Web navigator:

http://host:port/StandardClass:nb_of_opt:opt1:....:optN?URL in general
<http://host:port/StandardClass?URL> when there is no optional parameters

What is important to respect is the standard constructor, which has three obligatory parameters and an undefined number of optional ones (zero by default).

The optional parameters purpose is to give the developers more flexibility: they are free to use them anyway they want.

However, the obligatory ones are fixed: dis (resp. dos) is the InputStream (resp. OutputStream) associated to the navigator socket and URL is the URL to be treated (distant URL or local file). The developer can use dis and URL to get the data and to dos to send back the result to the client.

References

1. "WAP WML", june 1999, URL : <http://www.wapforum.org/what/technical/SPEC-WML-19990616.pdf>.
2. E. Nebel and L. Masinter, RFC 1867, november 1995.
3. E.R. Harold, *JAVA I/O*, O'REILLY, march 1999.
4. S.E. Deering and D.R. Cheriton, RFC 966, december 1985.
5. E.R. Harold, *JAVA Network Programming*, O'REILLY, march 1999.
6. D. Estrin, D. Farinacci, A. Helmy, RFC 2117, june 1997.