Building an integrated communication environment

Pedro Sousa <pns@uminho.pt>
Rui J.P. José <rui@uminho.pt>
António Costa <costa@uminho.pt>
Vasco Freitas* <vf@uminho.pt>

Abstract

Over the past few years, WWW has become the most successful information service on the Internet. This can be partially justified by the level of integration achieved by client software: an easy to use interface based on hyper-link navigation and a common naming convention (URLs) well supported by a multi-protocol machine (HTTP, FTP, etc).

This out-of-the-box solution fulfills most of user needs, since the integration design process privileged the most commonly used set of services.

A look backwards also brings up another example of success through integration: The BBS paradigm of communication. Traditional BBSs, successfully join small communities of users with common interests, offering them a way to share information, talk to each other (chat), send and receive private e-mail messages, and participate in open discussions using public messages.

This paper shares the authors' recent experience in the design and development of a communication environment with BBS-like functionalities, which is based upon WWW technology. Relevance is given to server side integration: a common set of data and configuration files, a set of administrative procedures and tools centered upon an HTTP server.

Since many other types of on-line services may have similar requirements, in this paper the emphasis is on analyzing the implementability of those communication models under WWW technology.

I. Introduction

This paper intends to share some of the experience gained in the design and implementation of a Bulletin Board System (BBS) based upon WWW technology and tools.

In the BBS paradigm of communication a centralised system offers small communities of users with common interests, a way to share information, talk to each other (chat), send and receive private e-mail messages and participate in open discussions using public messages. A BBS *forum* simply consists of a special interest group within the BBS community of users. Overall, the BBS is a set of *fora*.

Although the project had to take into consideration several topics specific to BBS systems such as *forum* management, the organization of image libraries and user information, interface design, and so on, the emphasis of this paper is neither on the final product nor on any BBS specific issues but rather on presenting the main technological issues raised by the attempt to place functionalities like exchanging public and personal messages, file transfer or chat on a single integrated WWW service.

II. The MARDUK service

The service aims to integrate a set of functionalities with the purpose of creating a community communication environment. In such an environment, the service would act as the tool that manages the exchange of information thus creating the necessary conditions for people in a given *forum* to access available information, discuss problems and actively contribute to the process of communication.

It was apparent that such an objective could be achieved by putting together a set of Internet servers like HTTP[1], FTP, News or E-Mail and thence integration would be viable at the level of the individual user interface by means of a multi-protocol browser being adopted as the universal client for all those services.

This level of integration, however, would not be enough. Integration had to be built at the server side as well, which would involve replacing the lot of independent tools by a single server providing not only the full set of functionalities but also management functions and access control.

The resulting server, which has been named MARDUK, was implemented, in its first version, by a set of CGI scripts and data files enabling an HTTP server to support centrally all the speci-

*author for correspondence

fyed functionalities.

In its second version, currently under development, a customised HTTP server is being built to allow a more robust and flexible solution. One of the main reasons for the move towards a proprietory server is the the fact that using CGI imposes several limitations upon the way the overall architecture works. The next section briefly discusses such limitations.

III. CGI limitations

The Common Gateway Interface (CGI), is a standard for interfacing external applications with HTTP servers. This mechanism has been successfully used to create gateways and, in general, to allow access to information which is not on static pages or files. It was therefore the obvious starting solution for the implementation.

This approach, however, has revealed a series of limitations:

- Although CGI intends to be a standard for the interface between an HTTP server and another program, it is in fact no more than an agreement between server implementors and therefore some small differences can be found from one implementation to another.

  One of such differences concerns authentication. The current CGI interface specification only defines two environment variables (AUTH_TYPE, REMOTE_USER), which are clearly insufficient. Some servers allow CGI scripts to fully handle authentication, but each one uses its own environment variables for that purpose.

  This small difference may be enough to create problems when trying to migrate CGI scripts to other HTTP servers.

- CGI-scripts accept input data from servers through *stdin* and return output data through *stdout*. However, *stdin* and *stdout* are usually opened in text mode, which means that carriage-return-line-feed (CR-LF) combinations are translated into a single line-feed (LF) on some non-UNIX operating systems. These conversions are adequate for text data but completely inappropriate, and even destructive, over binary data.

  In order to have binary CGI output, such as image files, there must be a special care in avoiding such translations.

- For each HTTP request a new instance of the CGI program is run. This is not a problem for simple and small scripts, but for long and complex programs, performance problems may arise because, eventually, they will take longer to load and initialize data structures.

  The problem of initializing internal data structures, like the ones used for authentication, message indexing or logging, is particularly important and performance issues must not be forgotten when planning their implementation on a CGI program.

During the initial phase of development, not so much attention was payed to these limitations. There was the feeling that performance degradation would be unavoidable, but not relevant. The other problems were not known at all at that time. However, as soon as the first results were made available, it started to be clear that the CGI constraints could not be ignored at all and design decisions would have to take them into serious consideration.

The first of the limitations pointed out above, was also the first to be noticed. The software was being writen for the Windows NT system, and an HTTP server was chosen for that platform. Fortunately the server allowed for CGI scripts to manipulate authentication resources, in a non-portable way however, since the CGI specification does not address this subject.

The dynamical generation of HTML[2, 3] pages by the the CGI script revealed the problem of passing binary data image files through the CGI interface. With special care on header generation and changing the open mode of *stdout* to binary, it was possible to overcome this small drawback, but in an unclear and unportable way.

In order to support storage and retrieval of authentication and access control information, a relational database was planned and a selected commercial database product was used. The schema design required three tables. Since all user requests had to be validated, at least one database access was needed for each request, but more often several accesses were actually performed.

Incorporating database access into the CGI script soon revealed serious performance problems. Opening the database was really the most time consuming operation, due to some heavy and unavoidable initialization procedures. The overhead introduced was measured: 10 seconds minimum for a single table ! (This value is obviously hardware dependent). Since the overhead is proportional to the number of tables, it has serious implications in the service response time.

Of course, the need for a relational database for this particular type of information is questionable. Small structured text files would do in this

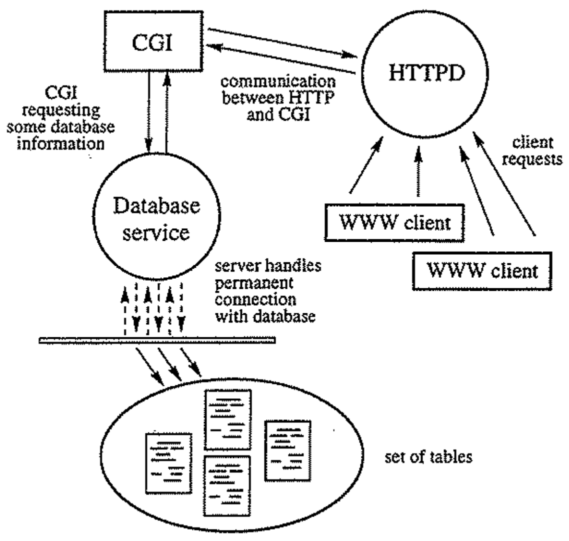
CGI
HTTPD
communication between HTTP and CGI
CGI requesting some database information
client requests
Database service
WWW client
WWW client
server handles permanent connection with database
set of tables

Figure 1: The database server

case, but what to say if its use becomes really inevitable? On the other hand it might be argued that several seconds are really bearable in database access. A possible answer may take into consideration the need for such access. If an user is trying to obtain information which is on a database he will probably be more tolerant to the delay than if he is performing some simple request which only uses the database for authentication purposes.

A new solution was therefore needed in order to keep the database as planned without affecting performance. It involved the development of a server which would handle all database manipulations and be permanently active to answer database access requests. After this was done the delay overhead became irrelevant since it only occurs at boot time. The CGI retrieves and stores information by sending appropriate requests to this server. Figure 1 illustrates this strategy.

IV. Integration issues

The integration into MARDUK of all the functionalities required to create the proposed communication environment, has raised several technological issues which will now be addressed.

IV.A. Authentication and Access

Most HTTP servers support mechanisms for authentication and access control. Although some of those mechanisms are very flexible and extensive they are not adequate to be used for controlling access to the MARDUK server. There are several reasons why specific mechanisms should be implemented:

- The typical protection mechanism of an HTTP server deals with entities like directories, files and password files. By creating a specific protection mechanism one can deal with higher level entities like *fora*, users or messages thus making management a more comprehensive task.
- Having a custom database is essential to support such protection schema but it can also constitute the base for supporting many other extra functionalities like specific logging, information related to user preferences on accessing the service or service management by custom tools.
- Controlling the access mechanism is a mandatory requirement should session control be an essential funcionality.
- The lack of standards on access protection profiles for HTTP servers makes the adoption of a common protection scheme very difficult.

In the MARDUK server the Basic Authentication Scheme[1] has been used for user authentication. The only difference being that on the server side the process is not handled by the HTTP server but by CGI which receives the necessary authentication information. For this to be possible an HTTP server which passes such information to CGI is necessary.

IV.B. Session support

The existence of user sessions is a common concept in many on-line services. A user session is a period of time during which the user is connected to the service and is delimited by *login* and *logout* procedures. Delimiting user sessions can be useful in several ways:

- it provides an easy answer to the question "Who is logged in?". That information can be used to initiate talks or other interactive services.
- the knowledge of a user's typical behavior within a session may be an important measure of interface adequateness. Avoiding for now the possibility of personalized adaptative interfaces, one may at least think of specialshort cuts for each user.
- statistical data, with adequate granularity, can be made available both to users and to administrators either with informal or formal objectives. Illegal or irregular access can also be more easily detected.

Supporting user sessions becomes a difficult task due to the nature of the HTTP protocol which is stateless and not oriented to connections. Each request is self-contained and independent and there is not a clear beginning or end of the user interaction with the server.

The implementation of a mechanism to support user session must take into consideration the authentication method being used. A possible solution for the Basic Authentication Scheme will be presented.

In this implementation the login procedure corresponds to the action of accessing for the first time, since browser activation, any protected document in the MARDUK server. When this happens the browser will prompt the user for his name and password keeping it for further accesses to the same realm. When the server receives an authenticated request it will check whether that user is already set as being logged in and if he isn't will set him as such. Since the user will never perform a session logout, the server always remembers his last access so that via a timeout mechanism it will eventually consider the user has being logged off as from the time of that access.

This is a simple but not very effective solution with some important disadvantages:

- There is no clean way of providing a logout mechanism. The browser will keep a password for each realm. This is clearly a potential problem of security specially in environments like for example schools were a terminal is shared by several users.
- For the same reason a clean mechanism of re-authentication cannot be provided either.
- From the service point of view there can never be a clear idea on what the limits of a session are. The view of sessions on the user side may be completely different from the view on the server side.
- Since the process of prompting the user for a password is handled by the browser, there is no control over it and an adequate interface cannot be created.

Some alternatives are currently being studied to overcome these problems. A potential improvement could be achieved by using dynamic realms. When a user logs in he receives a realm indication which was generated for him and for that session. As usual the browser will keep the password and will continue to use it for accessing that realm. But when either the user gives a logout instruction or some timeout is reached the server will mark the user as being logged off and will generate a new realm for the next session. If a new request is issued using the same authentication information the server will detect that the user is not logged in and will issue a new challenge indicating the new realm to be used. The browser will then prompt the user to re-enter the authentication data. This solution, however, requires full control of realm processing which cannot be achieved unless the HTTP server is customised.

A completely different approach can be followed should authentication be done outside HTTP. In such a situation the login procedure is performed by filling in a form. The information is sent to the server which will then generate a temporary authorization key that will be placed on the HTML pages it generates. When a link is followed, the key is sent over to the server which will in turn generate a new one to be placed on the returned page. Since a much tighter control is available it becomes a lot easier to create *login* and *logout* procedures.

IV.C. File Transfer

This section addresses the question of how to support file transfer within the HTTP protocol. The objective, which would not be realised by the use of FTP tools, is to provide a fully integrated file transfer.

A basic functionality of the MARDUK server is the existence of a structured repository of information made of HTML documents, images and several other file types. Users can read this repository but they also expect to be able to contribute by incorporating their materials into it.

Giving users the possibility to read this repository presents no major problems. Users can easily browse the HTML pages, see their images and download existing files. Attention must be paid however to the fact that the MARDUK server uses the CGI interface. As mentioned in section III. if the *stdio* and *stdout* of the CGI is not opened in binary form then images and other binary files will not pass safely through the interface unless encoding mechanisms are used. This same problem occurs in the opposite direction when files are to be uploaded.

Uploading, however, presents some additional problems. Even though the POST method of the HTTP protocol can be used for that purpose and the server may be prepared to handle it, there is no clean method for generating such request from a WWW browser. These browsers do not provide users with the means for selecting the files he wants to submit. Another important feature would be the possibility of the server to express file upload requests to the client. A form based solution for this problem which extends the HTML language by adding the FILE option to the TYPE attribute of the INPUT element is proposed in [4]. When the form is submitted, all its data, including the file, is posted to the server. For the moment, browsers do not support such feature yet.

The solution was to build a custom client just to perform file uploading. The main reason for this approach has been to provide a solution for the file upload problem without having to wait for technological evolution, but other features were added.

The client asks the user about the file that he wants to transmit, his identification, the destination *forum* where the file is to be placed and some extra options that will be explained latter. After that, an HTTP message is constructed specifying the POST method and the CGI that handles the request. Additional information (such as *forum* identification, user identification, and so on) is placed in the URL[5]. The body of the HTTP message will be the contents of the file. The message is then prepared to be sent to the server.

On the server side the CGI only needs to extract the body of the HTTP message and, if the user has the required permissions, place it in the destination *forum*.

In the particular case of an HTML document transfer it is possible to select of a Follow Links option. Using this option the user only needs to introduce the identification of the main page which he wants to transmit and all local pages referred in the document will also be transmitted. This feature is implemented by using a local *robot* which performs a recursive references search algorithm. Every non-local or repeated references will be discarded by the client.

IV.D. Public Messages

A discussion list is a space for the exchange of public messages within a group of people. During the planning phase, several approaches for supporting this functionality were considered but discarded as ineffective in face of the integration requirements.

The first of these approaches has been the use of a mailing list. But mailing lists have a fundamental difference from the kind of messaging system envisaged. Messages are sent to the user's mailbox. The desired functionality is that of a discussion group to which users may pop-in when they want to and not a redirection mechanism that bombs the user's mailbox with messages from all the lists he is participating in.

To solve this problem, a variant to the mailing list has also been considered, in which messages are not sent to the user but archived and accessed with a front-end on the WWW browser.

In any case the existence of a mailing list implies the existence of a mailing system and its correspondent set of management information which would compromise the desired integration.

Another possible approach for implementing public messages is the Usenet News system. Although this would correspond in terms of functionality to the intended service and some WWW browsers support direct access to NNTP servers, integration would again be compromised. Just as it happened in the mailing list solution the need for an extra system, in this case a local NNTP server, and correspondent management information would make it very difficult to support some essential requirements like an integrated access control.

Considering that mailing lists and newsgroups can not be adequately integrated into the MARDUK service and that such situation is not expected to change in the short term, a choice has been made to implement a custom public messages system (PMS). Only by creating this self-contained system inside the MARDUK server would the necessary level of integration with the other components of the server be achieved.

Using the HTTP protocol for this purpose presents no major problems. The user composes its messages using a HTML form, selects the submit button and the message is then posted to the HTTP server which will deliver it to the PMS module. The PMS module, which is the module responsible for all the messages management functionalities like for example storing them and maintaining indexes, will then perform the necessary access control verifications. The use of forms to compose and submit messages provides a better control of that process than using the interfaces of WWW browsers or mail applications:

- The possibility of creating an adequate interface which may even be adapted to the context or to the user.

- The possibility of providing several default values for the fields in the form.

The chief advantages of the customised approach are a complete integration with the other services and the possibility of including additional features like attachments, off-line reading or a form based interface.

IV.E. Electronic Mail

In an integrated community communication environment, the existence of some mechanism for interpersonal data exchange is of paramount importance. Electronic Mail aims at this kind of service.

Integrating this functionality into the MARDUK server involved the discussion of two different aspects. The first was the kind of interface to be provided to the users. The other was the architecture that should be adopted on the server side.

Considering that the MARDUK server lives in a WWW environment it was only natural that access to the e-mail service be done via the WWW client which sugested that some HTML form be presented to the user to allow him to visualize and submit messages. However, use of other e-mail clients should be allowed and the corresponding mechanisms to integrate this situation be catered for.

Therefore some practical situations such as this had to be taken into consideration before deciding on an architecture.

One of them concerns the level of experience of users that integrate the community. Some might already have access to an e-mail service before joining the community, for whom it would make no sense to force them to a new mailbox in the MARDUK server. Others might have no previous experience with e-mail and even have no mailbox of their own, for whom the MARDUK server would have to provide the full service.

The conclusion was that the server had to support a local repository for the user's mailboxes but should not confine the e-mail service to this functionality only. The required solution, which figure 2 illustrates had to be flexible enough to support the different situations. A WWW client is used to send and read mail. When reading mail, the WWW client communicates with the Mail Reading Module which accesses the user mailbox *on-the-fly* and converts its contents into HTML. When posting mail, the client uses the Mail Submission Module which contacts the SMTP server for posting the message into the community spool area. This feature can also be made available through the use of a mailto URL. Both the Mail Submission Module and the Mail Reading Module can coexist in the same CGI which implements the MARDUK functionalities.

The POP or IMAP server element must be added to the server platform in order to provide the possibility of remote access to the community spool from an IMAP or POP client. This feature

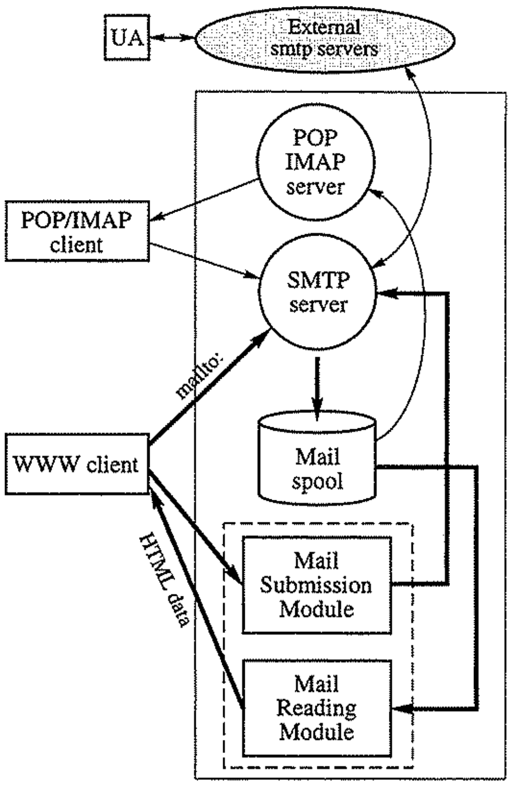

Figure 2: Architecture of the e-mail subsystem

can be used if a user wants to read his messages without entering the MARDUK environment.

Depending on policy, the communication between the community SMTP server and other external e-mail servers is possible. This feature allows the exchange of e-mail messages between users with a local mailbox and users with external ones. Members of the community are free to use other external mailboxes.

Again, depending on policy, a closed e-mail environment can be established. In this case all users are forced to a local mailbox and communication is restricted to the members of the community. For that purpose it suffices to turn off permissions for external access by the SMTP community server.

IV.F. Interactive services

A services like Multi-user Chat can be viewed as a textual conferencing system where users can virtually meet each other and have synchronous conversations. This kind of interactive services is very appealing and therefore important. However, its full integration in the WWW is rather complicated as the HTTP protocol is asynchronous and designed for unidirectional communication.

Nevertheless some features of the HTML language may be used to advantage, for instance, the possibility to invoke a telnet session and thence to have a connection to a multi-user talk daemon. Other possibility is the configuration on the local Web Client of some MIME types that when received by the client will originate the local execution of a specific *chat* client which in turn contacts a server which provids the corresponding service.

Both solutions can be followed in order to support synchronous sessions but they share a common problem which is the availability of specific clients (like telnet or chat) correctly installed on the user's side.

More fundamental than the solutions to the problems raised is the fact that they are not real integrating solutions, the WWW being only used for calling external entities which handle the rest of the session. Nevertheless they constitute a first approach towards integration.

Its possible that newer versions of the HTTP protocol will address this kind of problems. There are, for example, some client pull implementations which provide for an automatic page refresh mechanism. With such a facility it is possible that, after having received a HTML document, the WWW client periodically generates a new request of the document to the server. This feature sugests a possibility to overcome some of the difficulties raised by the asynchronism and unidirectionality properties of the HTTP protocol.

V. Conclusions

This project provided an insight into the issues raised by the design and implementation of an integrated communication environment based upon the BBS paradigm of communication using WWW technology. The analysis of the particular aspects discussed concerning the requirements that the system must satisfy shows that a complete and *clean* integration is not yet possible thus making it necessary to use ad-hoc solutions which do not completely fulfill the desired level of integration. However, the fast progress in WWW technology is constantly providing developers with new solutions but is also leading them to permanent redesigns.

In such an evolving environment, todays solutions may quickly become outdated and a balance must therefore be decided between developing ad-hoc solutions that work today and waiting for new system specifications and standards. The consideration of this principle and the foresight of the short term technological developments which are expected in WWW technology as well as the costs of implementing particular ad-hoc solutions, led the project to the specific choices in the way the different functionalities were to be supported.

The crucial strategic decision was the move towards the development of a customised, dedicated, HTTP server instead of combining an existing *standard* HTTP server with a set of CGIs. Only this way could the CGI limitations referred in section III. be completely overcome. The extra effort introduced by this decision is now paying back with the increase in size, features and computational work of the MARDUK server.

VI. References

[1] T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen. Hypertext Transfer Protocol - HTTP/1.0. *INTERNET-DRAFT, HTTP Working Group*, October 1995. Work in Progress from the HTTP Working Group of the IETF, <URL:ftp://nic.nordu.net/internet-drafts/draft-ietf-http-v10-spec-04.txt>.

[2] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0. *RFC 1866, MIT/W3C*, November 1995. <URL:ftp://funet.fi/rfc/rfc1866.Z>.

[3] Dave Raggett. HyperText Markup Language Specification Version 3.0. *INTERNET-DRAFT, HTML Working Group*, March 1995. Work in Progress from the HTML Working Group of the IETF, <URL:ftp://nic.nordu.net/internet-drafts/draft-ietf-html-specv3-00.txt>.

[4] E. Nebel and L. Masinter. Form-based File Upload in HTML. *RFC 1867, Xerox Corporation*, November 1995. <URL:ftp://funet.fi/rfc/rfc1867.Z>.

[5] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). *RFC 1738, CERN, Xerox Corporation, University of Minnesota*, December 1994. <URL:ftp://funet.fi/rfc/rfc1738.Z>.

Author Information

Pedro Sousa graduated in Systems and Informatics Engineering in 1995 at the University of Minho, Portugal, and is currently studying for a Masters degree in informatics. Since his final graduate year he has been with the Computer Communications Group of the University of Minho where he has been colaborating in the current project.

Rui José has recently joined the Department of Informatics of the University of Minho as an

Assistant Lecturer. He graduated in Systems and Informatics Engineering in 1993 and received his Masters degree in 1995 both at this University. He has been involved in several projects on X.500, WWW and Information Services.

António Costa is Assistant Lecturer in Computer Comunications at the University of Minho, Portugal. He graduated in Systems and Informatics Engineering in 1992 at this University and colaborated in the operations of the national R&D network (RCCN) until 1994. His main interests are network information services and protocols. He has been involved in several R&D projects under contract with the Computer Communications group in this area.

Vasco Freitas is Associate Professor of Computer Communications at the University of Minho, Portugal. He graduated in electronic and telecommunications engineering in 1972 at the University of Lourenço Marques and received his M.Sc. and Ph.D. degrees from the University of Manchester (UK) in 1977 and 1980 respectively. From 1989 until 1994, he was in charge of the establishment and management of the Portuguese R&D Network (RCCN). He represented this network in the RARE Association, DANTE and Ebone from their beginnings until 1994. Currently, his interests concern networked information services and protocols and the specification, modelling and prototyping of communication protocols.