Diplomarbeit in Telematik, TU Graz

Institut für Informationsverarbeitung und Computergestützte neue Medien

The World-Wide Web Gateway to Hyper-G: Using a Connectionless Protocol to Access Session-Oriented Services

Christian Derler

Gutachter: O.Univ.-Prof. Dr.phil. Dr.h.c. Hermann Maurer

Betreuer: Dipl.-Ing. Dr.techn. Frank Kappe

Graz, im März 1995

Abstract

In this thesis, possibilities are studied how session-oriented services on the Internet can be made accessible to users of client programs that use a connectionless application-level protocol.

The concepts and properties of two Hypermedia Information Systems, the World-Wide Web (WWW) and Hyper-G, are described and a comparison between these two systems is given. A gateway program was developed which is used as a protocol converter between the session-oriented Hyper-G client-server protocol and the connectionless Hypertext Transfer Protocol (HTTP). A mechanism was implemented that allows to differentiate HTTP requests and to assign them to Hyper-G sessions. The gateway provides users of World-Wide Web clients with session-oriented access to information residing on Hyper-G servers.

Table of Contents

1. Introduction

As public interest for a global information infrastructure currently increases tremendously, modern hypermedia information systems are becoming more and more important on international information networks. At the moment, there are different systems already in wide use but also under development. Interoperability is therefore one of the key issues.

In fact, different information systems, all based on the client-server principle, usually differ in the application-level protocols which they use for communication between clients and servers. As long as this is only limited to diverse naming conventions and other "minor" differences, conversion between two protocols can easily be performed.

A problem arises, however, if one system uses a connectionless and totally stateless client-server protocol and the other system is based on a connection-oriented protocol that maintains a state between consecutive transactions. This is the case with the two hypermedia information systems World-Wide Web and Hyper-G. Moreover, there are features that are only supported by one of these systems.

The main goal of this thesis is to study possibilities how a connection-oriented access to a Hyper-G server can be provided for users of WWW clients.

The approach taken for this project was to build a gateway that can be seen as a WWW server from the WWW client's view and as a Hyper-G client from the Hyper-G server's view. In fact, the gateway program is a complete WWW server since it meets two requirements.

- It talks the client-server protocol used by the WWW when communicating with clients.
- It is able to provide documents in formats used by the WWW.

The difference to a usual WWW server is that the gateway does not retrieve resources from a simple file system, but has a Hyper-G server as its back-end database.

The gateway is designed to work as a normal Hyper-G client which uses a connection-oriented client-server protocol and maintains an individual session for each user of a WWW client. For this purpose, a concept was developed that allows to distinguish requests from different WWW clients and to assign them to already opened Hyper-G sessions.

The fact that the gateway has now been in use with every Hyper-G server for some time may be seen as an indication that the project was a successful attempt to improve interoperability between Hyper-G and the World-Wide Web.

In chapter 2 of this thesis, an overview of the World-Wide Web project is given before the three key concepts of the WWW, the Hypertext transfer protocol, Uniform Resource Locators, and the Hypertext Markup Language are dealt with in some more detail.

Chapter 3 describes Hyper-G, its concepts, features and applications. At the end of this chapter, a comparison between Hyper-G and the WWW is drawn as differences and similarities have an important impact on the design of the gateway.

Finally, the implementation of the gateway program is described in chapter 4. The goal of the project and the concepts developed are explained in detail. After the structure and the functionality of the program is shown, a detailed description for users of the gateway is given. A separate section deals with possibilities of program customization. Last but not least, ways how to use the gateway together with the Hyper-G document cache as a proxy server are discussed.

2. The World-Wide Web

The official definition describes the World-Wide Web (WWW, W3 or sometimes simply called The Web) as a *"wide-area hypermedia information retrieval system"*. Its aim is to give universal access to a large universe of documents. It is one of the information systems currently working on top of the Internet and it is probably the most popular at the moment. Although it is strongly based on hypertext, it includes the handling of other types of data as well, such as image, sound, movie, etc. Being the first widespread hypermedia system available through the Internet, the Web has replaced some "traditional" information systems. Information about the WWW in general can be found for example in [BERNERS-LEE92], [BERNERS-LEE92a], [HUGHES93] and [BERNERS-LEE94a]. Useful information for the WWW user can be found in [KROL94], which also compares the WWW with other information systems available on the Internet.

2.1. History

The birthplace of the World-Wide Web was the *European Laboratory for Particle Physics (CERN)* located in Geneva, Switzerland. For a long time there has been a special need for wide-area hypertext to support the research activities of particle physicists. The need arises from the geographical dispersion of research institutes working together on the same projects. Very often researchers including students and visiting scientists only work at one institute for a limited period of time before leaving for some other location. "New" researchers replacing them have to get "up to speed" on projects quickly. Much of the information they need is in fact available online, but retrieval always needed some special knowledge of the network. One had to remember complicated instructions for different protocols, host names, and terminal types. One of the original aims of the initiative at CERN was to provide a system with a "point-and click" interface.

The first project proposal dates from the year 1989 and was written by Tim Berners-Lee. In October 1990 the project proposal was reformulated with Robert Cailliau as co-author and in November of the same year a prototype of the initial WWW text browser/editor was developed for the NeXT using the graphic user interface tools NeXTStep. At Christmas 1990 the NeXTStep and a line mode browser were presented at CERN. In May 1991 the line mode browser (www) was released on central CERN machines. The *National Center of Supercomputing Applications (NCSA)* at the University of Illinois, Urbana-Champaign, started to develop a multi-platform interface to the World-Wide Web and in February 1993 the first alpha version of Marc Andreessen's "Mosaic for X" [ANDREESSEN93] was released. In September of the same year the NCSA released working versions of Mosaic for all major platforms including versions for PC/Windows and the Macintosh. At CERN the first international World-Wide Web Conference was held in May 1994.

A brief history, written by R. Cailliau, who has followed the development of the Web from the very beginning, can be found in [CAILLIAU95]. A detailed list of dates concerning the history of the WWW can be obtained from `http://info.cern.ch/hypertext/WWW/History.html`

2.2. Architecture

The World-Wide Web, like many other applications on global networks, uses a **client-server model**. WWW clients, often called "WWW browsers", try to fetch information from remote servers, where the information resides and present the data to the user. WWW servers must make information available. This sometimes includes the ability to perform searches. The server is free to generate requested documents by sending real files or by generating virtual hypertext on the fly. This makes it possible that a "gateway" server could pass a hypertext image of another system to a WWW browser.

The advantages of such an architecture are that the user interface remains the same, no matter what type of server the client is connected to for a particular transaction and that the user need not understand the differences between protocols. The concept of hypertext is sufficiently simple to require no training for a computer user. As hypertext is transferred in mark-up form, which is a logical representation compared to the actual form in which the information is presented to the user, the client program can make optimal use of fonts, colours, and other interface resources available on the user's computer.

To make this flow of information possible three major "standards" and conventions were defined. At first, a consistent address system using a compact syntax for a "Uniform Resource Locator" was defined. This is described in chapter 2.3. In addition to other protocols which may be used with the WWW a new protocol, the "Hypertext Transfer Protocol" was defined giving performance and features not otherwise available. This protocol is explained in chapter

2.4. A set of data formats can be used in which data can be transferred. One of them is a new data format, the Hypertext Markup Language, described in chapter 2.5., which is the main format for transmitting hypertext.

Ideally, these conventions should allow current and even future software systems to work together across different platforms.

Interoperability with other systems

Standard protocols used by the WWW are **FTP**, which allows access to archives of all kinds of information including software and documentation, and NNTP which is used to access **network news**. By the way, news articles make good examples of documents which can be converted into hypertext, as they contain references to other articles and news groups.

Other protocols used by existing information systems are supported by the WWW. In particular there are the **Wide Area Information Service (WAIS)** [KAHLE92], which allows searching through indexed material and finding articles based on what they contain, and **Gopher** [LINDNER94], which provides a menu system for browsing resources on the Internet. The functionality offered by WAIS can be used by WWW browsers as well as Gopher menus can easily be converted into a list of text elements with hypertext links to other documents. So the Gopher space can be used as part of the WWW.

Providing Information

One of the strategies of the WWW is that existing online information, no matter in what format it is stored, maintained and originally made available, may be "published" as part of the web. This can be achieved by giving access to data through small and simple server programs. Rather than having one single type of WWW server program, every server which understands at least one of the specified protocols and which can provide data in at least one of the specified formats used by the World-Wide Web is a "WWW server". This approach is probably one of the reasons why the use of the WWW is increasing so quickly.

2.3. Uniform Resource Locator (URL)

This chapter gives a definition of the Uniform Resource Locator as specified in [BERNERS-LEE94], which is also called the "URL specification" in this chapter.

In the first part, the general syntax is described. The second part deals only with the special rules which are applied to the HTTP protocol. Rules for other protocol schemes are omitted here. In the third part, the so-called URL encoding is described. And finally, some considerations about security of this mechanism are mentioned in the fourth part.

A URL is a formalized piece of information for location and access of resources on the Internet. The URL-scheme is derived from concepts introduced by the WWW global information initiative. It has been successfully used by the World-Wide Web for quite a long time and is one of the main ideas that form the whole concept of the WWW.

2.3.1. Syntax

Originally, URLs are a mapping of physical addresses of objects which are retrievable using already established protocols. However, the generic syntax allows the design of new schemes for names to be resolved using protocols which have not been specified up to now.

A complete URL consists of a *naming scheme specifier* as the first element followed by a string. The way of defining a new URL scheme can easily be explained. If you can encode all protocol parameters necessary to access the object into a string with a prefix to identify the protocol, you will have a new scheme.

The exact format of the string is specific to the naming scheme. However, those schemes which are used in locators of information on the Internet have a common syntax for the string which is the so-called *Internet protocol part.* The naming scheme specifier is separated from the string by a colon.

Naming scheme specifiers

http	Hypertext Transfer Protocol (see chapter 2.1.2.)
ftp	File Transfer Protocol
gopher	The Gopher Protocol
mailto	Electronic mail address
news	Usenet news
telnet, **rlogin**	Reference to interactive sessions
wais	Wide Area Information Servers

and some others which are less important. Moreover, new schemes may be registered by future specifications.

<u>The Internet protocol part</u>

The second element of the URL consists of the *Internet Protocol (IP) address part* followed by the *path*.

<u>The Internet Protocol (IP) address part</u>

This starts with a double slash "//" and ends with the following slash "/". The double slash also indicates the presence of the IP address part in a certain URL. Within that part are

- *a user name,*	which is optional and rarely used.
- *the internet domain name*	of the host. Its format is specified in RFC1034 [MOCKAPETRIS87]. Instead of the domain name the IP address as a set of four decimal digits can be used.
- *the port number,*	which is given in decimal notation after a colon if it is not the default port number for the protocol.

<u>The path part</u>

This is the rest of the locator and it includes information that should not be processed by the client. In general, the slash "/" indicates a level in a hierarchy, with the lower level part on the right side.

2.3.2. The HTTP Scheme

The HTTP protocol specifies that the path part is handled transparently. Only the servers de-reference URLs. The path is passed by the client to the server with any request, but is not actually understood by the client. According to [BERNERS-LEE92a], the document name usually contained in the path part should be composed of printable characters and should not contain any information about particular formats available for a document or its length.

If the URL which is sent to the client refers to the same server that sends the URL, the host part is omitted. In this case, only the path part is sent by the server. However, when a server is being used as a gateway (especially a "proxy gateway"), then the complete URL is passed on the HTTP command line. If there is a search part present, it is treated as part of the path and therefore sent as part of the HTTP command.

2.3.3. Encoding of Prohibited Characters

When a system has a local addressing scheme, it is often useful to define a mapping from local addresses into URLs. In this case, references to objects within the addressing scheme may be accessed (possibly via gateways) globally. The URL specification allows the definition of any mapping scheme provided that it is unambiguous, reversible, and produces valid URLs. It is recommended that hierarchical structures in a given local naming scheme may be mapped onto the hierarchical URL path syntax. This allows the partial form of the URL to be used.

According to the encoding scheme proposed by the URL specification, any character which is not allowed in a URL, or any character whose use, although technically allowed, would possibly cause problems of corruption by imperfect gateways or because of different character sets, should be encoded as follows. Characters in question should be represented in the URL by a percentage sign "%" followed by two hexadecimal digits (0-9, A-F) which represent the value of the character.

2.3.4. Security Considerations

It should be mentioned that there is no guarantee that objects which could be located by certain URLs at one time will be removed or moved to another location at some later time, which results in changed URLs. The same URL could even point to one object at one time and to another object at some other time.

Another point is that it is sometimes possible to construct a URL with the intention to perform a harmless operation, such as the retrieval of an object according to one protocol, that will in fact cause a destructive remote operation to occur. Typically, this is the case when the URL contains a non-standard port number for the given protocol. If the server is in fact running a different protocol on that port number, this could cause problems, especially when the instructions contained in the URL have a totally different meaning according to the other protocol.

Finally, as stated in the URL specification, the use of URLs containing passwords is clearly unwise.

2.4. Hypertext Transfer Protocol (HTTP)

In this chapter, the application-level protocol which is used by the World-Wide Web for information retrieval and manipulation is explained. This version of the HTTP protocol, known as HTTP 1.0, is an upgrade of the original protocol implemented in the earliest WWW releases.

The purpose of the Hypertext Transfer Protocol is to provide a fast and flexible mechanism for following references between units of information which are distributed at different locations across the Internet. Its name may be misinterpreted. It is not only a protocol for transferring hypertext, but for transferring all sorts of information with the efficiency that is needed for following links in a hypertext system.

The protocol is *stateless* and *object-oriented.* 8-bit transfer is always used. It provides several commands for retrieval, manipulation and searching of data in a distributed collaborative hypermedia information system. One of its properties is that the set of HTTP commands could easily be extended.

The process of retrieving or manipulating information using this stateless protocol always consists of two operations. A *request* is sent from the client to the information source, the server. This request contains the type of operation (retrieval, searching, etc.) which the client wants the server to perform and secondly, some sort of identifier for the object on which this operation should be performed. The type of operation is set by the command, which is called the "method" of a HTTP request. The problem of referencing a certain object is solved by using the concept of Uniform Resource Locators which is described in chapter 2.3. In addition to that, for some types of operation the client has the possibility to send additional data to the server within the request. The server, after trying to fulfil the request, sends a *response* to the client. The response of course depends on the request and on the fact, whether or not the request has been processed successfully.

A connection between client and server is established by the client. Normally, when TCP/IP is used, the default port for HTTP transaction is the reserved number 80. But other port numbers may be used, if they are specified in the URL of the request. After setting up a connection, the client sends the request and waits for the response from the server. After that, the connection is closed by either both parties. In other words, a connection between client and server is only set up for one transaction.

2.4.1. Request

The client's request to the server consists of the following parts:

- The first line[*]
- The request headers (are explained in 2.4.1.2.).
- An empty line
- Data

All other parts except for the first line are optional. Their presence depends on the method and sometimes (for the headers) on the implementation. An empty line has to be sent to indicate the beginning of the data body. If no data body is present, an empty line indicates the end of the request.

The *data body* can be any MIME conforming message [BORENSTEIN92], therefore also binary data. Its length is given on a header line (see 2.4.2.2.).

To make clear how the server can know about the **end of the request**, this can be explained as follows. Lines can be read until an empty line occurs, which indicates that all header lines have been read. If a *Content-length* field is present in the header, it tells the server that the following data body contains the specified number of bytes. If the field is omitted, usually a method is used which implies that there is no data body following, as it is with GET, for example.

The *first line* is defined as: **`<Method> <URL> <Version>`**

`<Method>` is one of the commands described in 2.4.1.1.

`<URL>` is defined in 2.3. A partial URL is normally given here if the scheme (http:) and the server are obvious. Whereas, if the server is being used as a gateway, the complete URL is given (see 4.7.1.). The URL should be encoded using the mechanism described in 2.3.3. at least to a extend that spaces and control characters (decimal 0-31 and 128-159) are escaped.

`<Version>` is the protocol Version which the client uses; usually this is: `HTTP/1.0`

[*] Lines are always terminated by CRLF. However, it is recommended that servers should be tolerant if lines are terminated by LF only.

If `<Version>` is not present, it can be assumed that the client uses HTTP Version 0.9, which uses a simple form of request. This simple form is also allowed for HTTP 1.0 for back-compatibility.

The *simple request* is defined as: `GET <URL>`

It has no request headers and no data body.

2.4.1.1. Methods

The method field which is the first element on the first line of a HTTP request indicates the operation to be performed on the object identified by the URL following as the second element.

What follows is a list of currently defined methods, as described in [BERNERS-LEE93].

GET	means retrieve the object that is identified by the URL. This is also used for searches, which is explained later.
HEAD	is the same as GET, but returns only headers and no actual data.
POST	creates a new object linked to the specified object. The new object is sent as the data body of the request.
PUT	stores the object which is sent as the data body under the specified URL. The URL must already exist.
DELETE	requests the deletion of the specified object.
LINK	links an existing object to the specified object.
UNLINK	removes the link from an object.
TEXTSEARCH	Objects may be searched for with a text string. This is implemented using the search form of the GET method.
SPACEJUMP	The object will accept a query which contains coordinates of a point within the object. Actual implementations use the GET method with special URLs as described later.

In addition to that, the following methods are defined: CHECKOUT, SHOWMETHOD, CHECKIN, SEARCH. These methods are not explained here because most of them are still under discussion and not yet implemented. For details, the reader is referred to e.g. [BERNERS-LEE93].

Currently, only the methods GET and POST are widely used in implementations of WWW browsers. The method POST is usually used in connection with "fill-out" forms (see 2.5.4.).

GET can also be used to perform ***searching***. The URL used in this case is suffixed by a "?" character and the text to be searched for. Like in every other URL, prohibited characters have to be encoded. In search terms, keywords are separated by plus signs ("+").

Example: `GET /info-server/searching?snowboard+races HTTP/1.0`

Another use of GET is the implementation of ***image maps***. By specifying a special URL, the client provides the server with information about certain coordinates within an image object. In reality, the user can click on a certain region of an image displayed by the browser. The browser then sends the coordinates of the point on which the user has clicked to the server.

Example: `GET /info-server/map/0.5+0.9 HTTP/1.0`

2.4.1.2. Headers

The following header lines are sent by the client in a request. As some of the headers are not yet specified in detail, others may be changed, and perhaps new fields will be defined in future, this list only contains the most important headers. However, actual implementations only use some of these headers. In any case, header lines have a format which is defined in RFC 822 [CROCKER82]. It is recommended that unrecognized fields should be ignored by the server. The order in which the lines appear in the header is not significant.

From:

This field can be used to pass the e-mail address of the user to the server. There it can be used for logging purposes. Whereas, using it as a simple form of access protection seems to be very insecure.

Accept:

More than one line of this field is allowed. These lines build a list of representation schemes for the object the client is ready to accept from the server. The values are the same as for the Content-Type: field of the response (see 2.4.2.2.) except that only for the Accept: field an asterisk ("*") is allowed as a wildcard.

Example: `Accept: text/plain; text/html`

`Accept: image/gif`

`Accept: */*` can be interpreted as "give me what you've got."

If there is no Accept: field, the server can assume that the client will at least accept text/plain and text/html.

Accept-Encoding:

This tells the server which encodings of the object are acceptable by the client.

Example: `Accept-Encoding: x-compress; x-zip`

Accept-Language:

A list of preferred languages for the object in the response is given here. It is not illegal for the server to respond in a language that is not specified here.

User-Agent:

The name of the client software product is given on this line. The first white space delimited word must be the product name followed by an optional slash and a version number. If other products (e.g. program libraries) are part of the client software, their names can be given as additional words on this line.

Example: `User-Agent: Mozilla/0.94 Beta (X11; SunOS 5.3 sun4m)`

Referer:

With this field the client can tell the server where the URL of the request was obtained from. The URL of the document which contained the link is given here. By gaining this information the server would be able to generate lists of back-links to objects and apart from logging purposes this allows bad links to be traced. Partial URLs are meant to be relative to the URL of the object.

Example: `Referer: http://hgiicm.tu-graz.ac.at/`

Authorization:

This line contains the name of the authorization scheme, a user name and an optional password. This provides only a low-level security as used by unmodified Telnet or FTP.

Example: `Authorization: user myname:mypassword`

However, this field could be used to provide other, more sophisticated, authorization schemes.

ChargeTo:

This line contains account information. It has been rarely used by actual implementations up to know. The format should be a function of the charging system. It is proposed that it could include a maximum cost for the transaction and a cost unit.

2.4.2. Response

The response from the server consists of the following parts:

- The status line (is explained in 2.4.2.1.).
- The response headers (are explained in 2.4.2.2.).
- An empty line
- Response data

In order to communicate with old servers which do not send a status line because they use the protocol originally implemented in the WWW, clients should tolerate such responses. In such cases, the client can assume that the response is a document of type text/html.

Response data may follow the headers in the format of a MIME conforming message body. The contents depends on the status code and the content type is given on a header line. The length of the message should be given on a header line with the *Content-length* field. As the connection is simply closed by the server after sending the last byte of data, the client can assume that all data has been read and the **transaction is terminated** when the connection was closed, even if no Content-length field was present in the header.

2.4.2.1. Status line

The status line, which is always the first line of the response, has the following syntax:

`<http version> <status code> <reason>`

`<http version>` is usually `HTTP/1.0` at the moment.

`<status code>` is a numeric 3 digit code. The values are listed later.

`<reason>` is a human readable text that explains the status code. It can provide some sort of diagnostic information.

In the following list all status codes are described. The `<reason>`-fields added to the status codes are those used in [BERNERS-LEE93]. However, different servers use different texts, as only the numeric code is significant.

The first of the 3 digits indicates the main category to which the code belongs to. Codes starting with **"2"** have the meaning of ***success***, **"3"** has something to do with ***redirection***, codes with **"4"** or **"5"** as the first digit are ***error*** codes, where "4" means that the server recognises an error of the client and "5" is intended to be an indicator for an error of the server. The last two cases are often not distinguishable, so the difference is not very important.

If the status code indicates success, the data body is the requested object. In cases of an error or of an redirection code, the data body can contain a document which is a sort of error message with a detailed description of the error in human readable form. This document is intended to be presented to the user by the client.

Status codes

200 OK

> The request was fulfilled. Data follows.

201 CREATED

> As a response to a POST request, it indicates that the document was successfully created.

202 ACCEPTED

> The request has been accepted, but processing has not been taken place because this is done through an asynchronous operation. There is no further possibility for indicating the actual result of such an asynchronous operation.

203 PARTIAL INFORMATION

This indicates that the returned metainformation is not from a server with a copy of the object, but may include annotation information about the object, for example, from a private overlaid web.

301 MOVED

The location of the requested object has changed; the change is permanent. The response must contain at least one of the header lines of the form:

```
URI: <url> string
```

`<url>` is the new URL of the object. `string` is an optional comment.

Browsers should automatically relink to the new URL, if they have the capability for link editing.

302 FOUND

This has the same meaning as 301 MOVED with the exception that the change is not permanent and the redirection may be altered on occasion.

303 METHOD

This also indicates that the client should try another URL. Moreover, a different method other than GET should be used. The response contains a header of the form: `Method: <method>` indicating the alternate method (see 2.4.1.1. for a list of methods). The data body can contain parameters to be used for the method.

400 BAD REQUEST

This usually is an indication of an syntax error in the request.

401 UNAUTHORIZED

The user has no access rights for the requested object. The response should contain information about suitable authorization schemes. The client is intended to retry the request with an "Authorization:" header.

402 PAYMENT REQUIRED

> Parameters in the response should contain specifications of the acceptable charging system. The client should retry with the proper "ChargeTo:" header.

403 FORBIDDEN

> The user has no access rights for the requested object. Unlike for status code 401, authorization will not help.

404 NOT FOUND

> The server was unable to resolve the specified URL. The object could not be found in the server's database.

500 INTERNAL ERROR

> The server was prevented to fulfil the request by an unexpected condition.

501 NOT IMPLEMENTED

> The server does not support the requested feature.

2.4.2.2. Headers

Like the header lines in the request (see 2.4.1.2.) all response headers are optional and the order in which they occur in the header is not significant. Unrecognized header fields should be ignored. The following list contains only the more important header fields. Only some of them are actually used by servers at the moment.

Allowed:

This line contains a list of methods (compare 2.4.1.1.) which are allowed for the specified URL.

Example: `Allowed: GET HEAD PUT`

If this is not present, the default methods are GET and HEAD.

Public:

This is the same as Allowed:, but lists only those methods which anyone may use. The default method is GET.

Content-length:

According to [BERNERS-LEE93], this implies that the data body is binary and should be read directly from the communications link, without parsing. However, this field is normally used in all responses in order to tell the client in advance how many bytes of data the client is going to receive. Clients can display this information, so that the user can estimate how long the transmission is approximately going to take.

This field is also used in requests, especially with method POST, where the data body is not empty.

Example: `Content-length: 123`

Content-type:

On this line the type of the document is given.

Examples:
```
Content-type: text/html
Content-type: image/gif
Content-type: video/mpeg
```

In a POST request that contains the contents of a fill-out form (see 2.5.4.), the following header is usually used: `Content-type: application/x-www-form-urlencoded`

Date:

This is the creation date of the object in GMT. The format is defined in RFC850 [HORTON83].

Expires:

This field gives the date when the information will be not valid any more. This can be used to control the caching mechanism of the client.

Last-modified:

The last time the object was modified is given here. The format is the same as with Date.

Message-id:

A document should have a unique identifier. No other document, even no different version of the same document, may have the same message-id. The id must be unique in all time.

URI:

This gives a URL to be used for the object. This field is normally used in responses which cause an automatic redirection (compare 2.4.2.1.).

Language:

This line gives the language code for the language in which the document is written. The language code is an ISO 3316 language code.

Example: `Language: en`

Cost:

This gives the cost for retrieving the object. The format is not yet specified.

2.4.3. Considerations about Log Information

A HTTP server is able to collect personal data about information requested by users. Especially the Referer: field, which allows to draw reverse links, together with the From: field, which gives the user name, can provide information which could be very powerful. For example, reading habits of certain users can be exactly studied.

On the one hand this may be very useful for information providers, but on the other hand the log information may be abused. Handling or publishing such information may be restricted by law which is, of course, different from country to country.

2.5. Hypertext Markup Language (HTML)

The design of HTML is closely related to the World-Wide Web initiative. Although it is only one of the data representations used with WWW, it is probably the most important.

HTML was "invented" because the WWW needed a common basic language for hypertext documents. To be more precise, HTML is a language for communication. It should mainly be used for *interchanging hypertext* over the network. There is no need for servers that files are stored in HTML format because HTML should be so easy to generate by programs that texts can be converted into HTML on the fly with each request [BERNERS-LEE94a]. On the other hand HTML is designed to be sufficiently simple in order to allow HTML documents to be easily produced by people, not only by machines.

One of the most important properties of HTML is that it is a *SGML document type*. The Standard Generalized Markup Language (SGML) [GOLDFARB90] can perhaps be thought of as a programming language for style sheets. Therefore HTML can be seen as a collection of styles described in a document type definition (DTD) according to the SGML standard. This means, in practical terms, that an HTML document can be parsed by an SGML parser provided with the HTML DTD.

This chapter consists of four parts. The first one gives a brief overview of different versions or levels of HTML specifications, because the further development of HTML is an important point of discussion on the Internet at the moment.

The specification of HTML is changing from time to time and there are several variants implemented in different WWW browsers. The description of the syntax and the elements of HTML given in the second and third part is mainly based on material found in [BERNERS-LEE93a].

2.5.1. Specification Levels

Specifications of HTML can be divided into 3 levels.

HTML level 1

This specifies a set of HTML properties which is understood by almost all WWW browsers at the moment. In addition to the HTML of the initial clients, it defines the use of inline images.

HTML level 2

This specification includes additional properties like forms for user input (compare 2.5.4.). Some of its features are already in use by various clients, for instance Mosaic, although the specification is still being refined.

HTML level 3

In addition to previous specifications, this defines more sophisticated features for HTML such as tables, figures, and mathematical equations. Sometimes this specification is also referred to as HTML+. It is under construction at the moment. A brief overview of HTML 3.0 properties is given in 2.5.5.

The properties of HTML which are described in the following sections of this chapter (2.5.2. - 2.5.4.) are basically those of HTML level 1, plus a feature from HTML level 2, which is the fill-out form support. The reason for this is that almost all WWW clients and most of the WWW servers are currently working with this set of properties and also the implementation of the WWW Gateway for Hyper-G (see chapter 4.) was based on these features.

2.5.2. General Syntax

An HTML document consists of simple text, except that parts of the text are interpreted as markup. For this purpose, a number of *elements* was defined. Every element starts with a *tag* and every element that is not empty ends with a tag. A start tag looks like `<tag>` and an end tag looks like `</tag>`. The name of the element (here `tag`) is case insensitive and has to follow immediately the tag open delimiter.

A start tag can contain attributes. An attribute consists of a name, an equal sign "`=`" , and a value. The value is a string surrounded by single quotes or double quotes.

Whitespace is allowed between the element name and the closing delimiter "`>`", around attributes, and also around the equal sign within an attribute.

You can use *character references* or *entity references* to represent characters that would otherwise be recognized as markup. Characters available by the writer that are not part of the character set can be expressed this way. A character reference starts with &# followed by a decimal character number and a semicolon.

Example: `<TITLE>Here is a less than sign &#60; in a title</TITLE>`

An entity reference starts with a & followed by an entity name and a semicolon. Some important entities are `&<;` for the less than sign, `&>;` for the greater than sign, `&amp;` for the ampersand sign & itself, and `&quot;` for the double quote sign. A list of all entity names can be found in [BERNERS-LEE93a].

Example: `Gr&uml;&szlig;e aus &Ouml;sterreich`

which results in: Grüße aus Österreich

2.5.3. Elements

Basically, HTML documents should contain an initial **HEAD** element followed by a **BODY** element, although this is not obligatory. The HEAD element contains all information about properties of the whole document, but it does not contain any text which is part of the document. All the information that belongs to the document is in the BODY element. Some elements that can appear in the HEAD element are listed in 2.5.3.1. All other sections (2.5.3.2. - 2.5.3.8.) describe elements which can only appear in the BODY element.

Browsers which do not support certain tags should ignore them. This allows new tags to be introduced easily.

2.5.3.1. Elements within the HEAD Element

The order in which these elements appear is not significant.

`<TITLE>`

The string between the start and the end tag should be the title of a document to be used in bookmark lists by the browser in order to label the displaying window, etc.

`<BASE>`

This allows the URL of the document to be specified for situations in which the document may be read out of context. Moreover, it gives a base address for resolving partial URLs found in anchors within the BODY of the document. If this element is not present, the browser will use the URL which was used to access the document to resolve partial URLs. The attribute HREF is used with this element.

Example: `<BASE HREF="http://hyperg.tu-graz.ac.at/search.html"`

`<ISINDEX>`

This tells the browser that the document may be queried with a keyword search. The browser usually reacts by displaying a keyboard entry field for entering keywords by the user. Normally, this tag is inserted by the server automatically if the server has the capability for handling queries. Obviously, it does not make sense to insert such a tag by hand if the server has no search engine.

2.5.3.2. Anchors

The text between the anchor start tag **`<A>`** and the anchor end tag **`</A>`** is expected to specify a source or a destination anchor according to the optional attributes included in the start tag. Either HREF or NAME is necessary for the anchor to be useful.

If **`HREF=`** is used, this specifies a source anchor. The value of this attribute is the address (usually the URL) of the destination document, the identifier of a destination anchor within the same document prefixed by the hash sign "#", or the URL plus an identifier which refers to a certain destination anchor in a different document. The text of the anchor is "sensitive" ; for instance, it can be displayed "click-able" by the browser.

The value of **`NAME=`** is a string which forms the identifier of a destination anchor. It must be unique within the document. The text of the anchor is the destination of the link.

Examples:
```
A <A HREF="http://info-server/descript">detailed
description</A> is given here.

Go to the <A HREF="#first">first chapter.</A>

<A NAME="first">First chapter</A>
```

2.5.3.3. Headings

There are six levels of heading elements defined:

`<H1>, <H2>, <H3>, <H4>, <H5>, <H6>`

The browser, of course, is responsible for a suitable rendering. That includes the generation of vertical whitespace before and after such heading elements. Therefore, it is not required to put a paragraph mark after a heading element.

Example: `<H1>Main Title</H1>`

2.5.3.4. Character Highlighting

The following tags allow the formatting of individual parts within the text in different styles. Some of the elements define more logical styles, others are closely related to their physical representation. Whenever possible, logical styles should be preferred.

`<I>`	Italic
`<B>`	Boldface

`<U>`	Underline
`<TT>`	Fixed-width font
`<EM>`	Emphasis, typically rendered as italic
`<STRONG>`	Strong emphasis, typically boldface

There are several highlighting elements that should be used to format sections of text with specific contents. They provide a sort of semantic markup. The actual rendering is configured by the browser.

`<CITE>`	A citation
`<SAMP>`	A sequence of literal characters
`<CODE>`	Example of code
`<KBD>`	Text typed by a user in a manual
`<VAR>`	A variable name
`<DFN>`	A definition

2.5.3.5. Paragraphs

To separate two pieces of text, you can put the paragraph mark `<P>` in between. It has no end tag. Typically this results in a line brake with some additional vertical space between the paragraphs.

2.5.3.6. Lists

There are **definition lists** and **simple lists** defined for HTML.

A simple list can be composed by putting a number of list items between a list start tag and a list end tag. Each list item starts with the list item tag `<LI>`, which has no end tag. Two different list tags are possible. `<OL>` starts a *ordered list*, where each item is numbered in some way. An *unnumbered list* starts with `<UL>`. The rendering software often marks its list items with bullets.

Example of an unnumbered list:

```
<UL>
<LI> list element
<LI> list element
</UL>
```

A definition list (or glossary) starts with the `<DL>` tag and contains term elements marked with `<DT>` and corresponding definition elements marked with `<DD>`. Term and definition must appear in pairs. The attribute `COMPACT` can be used with the `<DL>` tag, which may result in a more compact rendering.

Example of a definition list:

```
<DL>
<DT>-h <DD>help text
<DT>-v <DD>debug output
</DL>
```

2.5.3.7. Other Formatting Styles

A section of a document which contains the name and the address of the author can be enclosed between **`<ADDRESS>`** start and end tags. This usually results in a special rendering of that section, typically as a separate paragraph and formatted in italic.

Another possibility to format a section of text in another style is the **`<BLOCKQUOTE>`** element. The section is usually put in a separate paragraph rendered with an extra indent and the text may have an italic font.

If you want to have parts of a document displayed in a fixed width font suitable for preformatted text, you can use the **`<PRE>`** tag.

2.5.3.8. Inline Images

An HTML document may contain embedded images. They are specified within the text using the `<IMG>` tag. The browser has to retrieve the image data whose URL is specified with the `SRC` attribute automatically. The difference to a normal link is that the image is not displayed as a separate document, but embedded within the text on the location of the `<IMG>` tag.

Attributes

`SRC`	The value is the URL of the image document.
`ALIGN`	This can have the values `"TOP"`, `"MIDDLE"`, or `"BOTTOM"`. It defines whether the tops, the middles or the bottoms of the image and text should be aligned vertically. The attribute is optional with `"BOTTOM"` being the default.
`ALT`	As not every browser is capable of displaying inline images, a text can be given as the value of this attribute which should be displayed instead of the image by text-only browsers.

2.5.4. Fill-Out Forms

Forms are probably one of the most interesting extension of the initial HTML specification. Their implementation in Mosaic for X 2.0 was perhaps the key for the widespread use by various information servers. The following section gives a brief description of fill-out Forms. See [BINA94] for details.

In order to introduce a form element into HTML the **`<FORM>`** tag together with the following attributes is defined.

`ACTION`	The value is the URL of the location where the contents of the form should be submitted to. If this attribute is omitted, the browser will send the contents to the URL of the current document.
`METHOD`	This specifies the HTTP 1.0 method (compare 2.4.1.1.) to be used for submitting the form. Principally, there are two possibilities. If method GET is used, the contents will be prefixed with a question mark '?' and appended to the URL as if there was a normal query. However, it is recommended to use method POST because this results in sending the contents in a data body rather than as part of the URL.
`ENCTYPE`	This can be used to specify the type of encoding for the form contents. At the moment, this is normally omitted because only one type, "application/x-www-form-urlencoded", is used.

There can be several different forms in one HTML document, but forms must not be nested.

Within forms, three different elements, each of which with several attributes, are allowed. These are INPUT, SELECT, and TEXTAREA elements.

`<INPUT>`

This is a standalone tag; there is no end tag. It causes, depending on the type, various interface elements (radio buttons, checkboxes, text entry fields, pushbuttons) to be displayed by the browser. The different attributes together with their possible values are as follows.

`TYPE`

`"TEXT"`	displays a text entry field. This is the default if no TYPE attribute is present with the INPUT tag.

`"PASSWORD"`	is the same as TEXT except that entered characters are not echoed to the screen.
`"CHECKBOX"`	shows a single toggle button (on or off).
`"RADIO"`	is the same as CHECKBOX except that more tags of that type with the same value for the `NAME` attribute are grouped into a "one of many" behaviour.
`"RESET"`	displays a pushbutton. When pressed, it causes all values of the input elements of a form to be reset to their initial values that were displayed right after receiving the document containing the form.
`"SUBMIT"`	is a pushbutton that causes the contents of the whole form to be put into a query string and sent to the server. A submit button is always necessary in a form with the only exception that the form contains only a single input element which must be an INPUT tag with TYPE "TEXT" or "IMAGE". In these cases, the form is submitted when the user hits return in the text entry field or clicks on the image.

`NAME`	Its value is the symbolic name for the input field. This must be present except for submit and reset buttons and is needed to form the name/value pairs which are sent to the server as the contents of the form.
`VALUE`	has different meanings depending on the type of the input element. For pushbuttons the value specifies the label of the button. For text and password entry fields it specifies the initial contents. For checkboxes and radio buttons it specifies the value of the button when checked. This value is sent to the server. No name/value pair is sent for unchecked buttons. If no value attribute is present, the default value for checked radio buttons and checkboxes is `"on"`.
`CHECKED`	for checkboxes and radio buttons this attribute specifies if the button is initially checked. It has no value.
`MAXLENGTH`	specifies the maximum number of characters accepted in text and password entry fields. If it is omitted, an unlimited number of characters is allowed, of course within a scrollable input field.

`SIZE`	for text and password entry fields this gives the size of the input field in characters. Multiline input fields can be specified as SIZE="width, height" (e.g. `SIZE="30,2"`). If the attribute is not present, the default value is 20.

`<TEXTAREA>`

This element causes a multiline area with scrollbars for entering text to be displayed by the browser. Text between the start and the end tag must be plain text. This text is the initial contents of the area. Attributes are NAME, whose value is the symbolic name for the element, ROWS and COLS, which specify the height and the width of the text entry area in characters.

Example:
```
<TEXTAREA NAME="message" ROWS="5" COLS="40">
Hello!
</TEXTAREA>
```

`<SELECT>`

These elements are usually represented as option menus and scrolled lists. Between the start and the end tag only a number of `<OPTION>` tags each followed by a piece of plain text is allowed.

The attributes of `<SELECT>` are as follows.

`NAME`	gives the symbolic name for the element needed for submission.
`SIZE`	If it is omitted, or if its value is `"1"`, the select element is represented as option menu. If its value is `"2"` or more, a scrolled list will be displayed with the value specifying the number of visible elements.
`MULTIPLE`	This has no value. If present, it allows multiple selections. Anyway, a scrolled list is displayed, regardless of the value of SIZE.

The `<OPTION>` tag can have the attribute `SELECTED,` which specifies that the marked option is initially selected. If `MULTIPLE` is present, more than one option can be marked with `SELECTED.`

Example:
```
<SELECT NAME="menu" MULTIPLE>
<OPTION SELECTED> First option
```

```
        <OPTION> Second option
        </SELECT>
```

In conclusion, a remark may be made on the *submission* of the form contents. As stated before, either method GET or method POST may be used to submit the form to the server. In any case, the form contents are put into a query string made up of *name/value pairs*. This is the reason why NAME attributes are needed. Names and values are separated by equal signs "=" and the pairs are connected by ampersands "&".

Example: `name1=value1&name2=value2&name3=value3`

If method POST is used for submission, this string is sent as a data body. Its length should be given in the HTTP header field `Content-length:` and its type is usually `Content-type: application/x-www-form-urlencoded` (compare 2.4.2.2.).

In case of method GET, the query string is prefixed with a question mark "?" and appended to the URL specified with the ACTION attribute.

2.5.5. HTML 3.0

This specification, which is currently being developed, tries to define the next generation of HTML which contains a lot of extended features. The WWW browser Arena [RAGGETT94] serves as a testbed for HTML 3.0.

The most important features are:

- Mathematical Equations and Formulae
- Resizable Tables
- Improved Layout Control (e.g. ALIGN="CENTER" etc. for headings and paragraphs, named horizontal tabs, control of textflow)
- Figures with Local Event Processing
- Document Specific Toolbars
- Predefined Guided Tours

2.6. Future Developments

- The goal of computer supported collaboration can only be reached if more powerful hypertext editors are developed which allow non-expert users to publish interlinked documents.
- The implementation of a name service that will allow documents to be referenced by name, no matter where the document actually resides, is desirable.
- Commercial publishers demand more sophisticated document type definitions.
- Specifications for charging and commercial use of the Internet have to be defined and applied by the WWW.
- Real-time features such as teleconferencing and virtual reality should be integrated into the WWW.

3. Hyper-G

Hyper-G is the name of a system which can be seen as the first *"second-generation"* hypermedia system. The Hyper-G project has been in progress at the Institute for Information Processing and Computer Supported New Media (IICM) of Graz University of Technology for a number of years. Its aim is the development of a general purpose, large-scale, distributed, hypermedia information system.

The first part of this chapter gives an overview of the concepts behind Hyper-G. The second part explains the organization of data in the Hyper-G system. For a detailed description the reader is referred to [KAPPE91a]. Information about the architecture of Hyper-G and the requirements of a modern hypermedia information system can also be found in [KAPPE91] and [KAPPE92]. The most important aspects of the internal concept are described in [KAPPE92a].

Hyper-G provides high level **navigation tools**, different **search strategies**, a **multi-lingual concept** for both documents and user interface, and a **user identification concept** with different levels of identification. The third part of this chapter gives a short description of these features. A good overview of Hyper-G can be obtained from [KAPPE93b] and [KAPPE93a].

The fourth part takes a look at some applications of Hyper-G, some of them already in use, some existing only as a concept.

In the fifth part, certain features of Hyper-G are compared with those of the World-Wide Web and the advantages of the more complex concept behind Hyper-G for a distributed hypermedia system are explained. (see e.g. [ANDREWS94] and [ANDREWS95] for details).

3.1. Design Concepts

The Hyper-G system operates on a *heterogeneous* computer network. Not only data, but also processes are **distributed**.

Like most other modern information systems, Hyper-G is based on the **client-server model**. Its advantages over conventional terminal-based systems are obvious. The client program can use all the features of the local hardware and software (such as graphics, sound, etc.) to produce a sophisticated user interface, which remains the same if the client connects to different server systems. In terminal-based systems, where the user logs into another system, the user interface may change depending on the server system. Also, the user interface has to be reduced to the smallest common denominator, which is usually that of a simple text terminal (e.g. VT100). In the client-server model, only the essential data is transmitted between client and server, usually in larger blocks. In conventional systems, every keypress has to be transmitted, which is a waste of bandwidth and, of course, leads to an unacceptable performance problem in case of a global distributed system.

Hyper-G is implemented in a very **modular** way. On the client side, rather than having a single client process running, you have a session manager controlling the entire Hyper-G session and a different viewer process for each document type. The Hyper-G server is not a single program either, but consists of concurrent processes, which may be distributed among a number of server machines. This increases performance and leads to an extensible system.

The design of the Hyper-G system is very much influenced by **object oriented** principles. All major parts of the software are developed with C++.

In the concept of Hyper-G there is a separation between the user interface and the underlying Hypermedia engine. This allows to run Hyper-G with **different user interface metaphors** (compare [MAURER90]), which is one of the reasons why Hyper-G can be called a general purpose hypermedia system (see 3.4. for possible applications).

The concept allows great **extendibility**. As every document type is handled by a separate document manager on the client side, new document types can easily be introduced by providing a new document manager. New user interface metaphors can be introduced without having much impact on the server.

3.2. Data Organization

3.2.1. Collection Hierarchy versus Hypertext

In Hyper-G, documents are members of so-called **collections**. Collections group together documents and other collections with related contents. As collections can contain other collections (called sub-collections) recursively, data can be organized in complex hierarchies.

This structure is not a tree, but a directed acyclic graph (DAG). This means that one document may belong to multiple collections, called parent collections of the document.

But this is not the only way how information can be organized in Hyper-G. Also the node-link model of **hypertext** is supported. Of course, this is not limited to text documents, but to all kinds of multimedia document types. In addition to the simple node-link model, Hyper-G does not only allow to follow a link in one direction, but in both directions. Moreover, links can have types and even access rights, which is used to achieve different types of annotations. The link management is described in 3.2.3. in more detail.

Combining both concepts in one information system can make the user's navigation much easier. One positive effect of providing a hierarchical structure is that the number of links can be reduced. Links often have a bad effect on the orientation of the user in information space (compare [MAURER94]). Together with the advanced search facilities (they are described in 3.3.1.) this is a powerful means for decreasing disorientation of users browsing in large amounts of documents.

3.2.2. Document Clusters

Similar to a collection, a *document cluster* contains a number of documents that are semantically related. The difference is the way in which they are presented to the user. When a cluster is selected, all components may be displayed simultaneously, whereas a collection is normally presented as a list of items with only their titles displayed.

Clusters are the way how **multimedia documents** are implemented in Hyper-G. If a cluster contains, for instance, a text document, a video document, and an audio document, the text and the video are shown in different windows while the audio clip is played. This can be done simultaneously because, as stated in the previous section, each document type is handled by its own document manager.

The cluster concept is also used to implement **multilingual documents**. A cluster may contain different versions of the same document, each written in another language. Depending on the setting of the preferred language by the user, the client will display the version written in the corresponding language, if available in the database. Together with the multilingual user interface provided by Hyper-G clients this concept meets the requirements of an *international multi-user environment.*

3.2.3. Link Management

As mentioned in 3.2.1., Hyper-G supports bidirectional links. Systems that support only unidirectional links can only show links that depart from the current document but not those that arrive at the document. The system cannot answer the question: 'What other documents refer to the current document?'. They are also unable to give an overview of the structure of the information network around the current document (graphical map). Such a map would decrease disorientation of users.

Supporting bidirectional links can easily be done by keeping links and anchors in a separate database, called the **link server** [compare KAPPE92a]. This link server is an object oriented database. It holds information about Hyper-G "objects", which can be links, anchors, descriptions of documents, relations between such objects (e.g. which documents belong to which collection, which source anchors are attached to which destination anchors, which anchors are attached to which documents, etc.) and additional attributes such as title, author, creation date, modification date, etc.

Unique *object IDs* are assigned to Hyper-G objects. A modified object receives a new object ID. Even if a document is deleted, the object ID is never used again.

The separation of documents and links has another advantage. Modification of a link requires no modification of the document. Therefore, it is possible that a user who has no write access to a certain document may attach an annotation. Even a document that resides on a CD-ROM can be annotated in this way.

As mentioned previously, Hyper-G supports typed links, which is used to implement different types of annotations. Links can also have access rights and, therefore, private links are possible.

3.3. Features

3.3.1. Search Facilities

Principally, two types of searching are possible. Every Hyper-G object has a number of associated **attributes**, which can be searched for. The most important attributes in connection with searching are title, keywords, author, creation date, and modification date. Because of the separate link server (compare 3.2.3.), storing all these attributes, complex queries (including boolean combinations of attributes) over the whole database or parts of it (collections) can be specified and searches can be performed efficiently.

Secondly, the contents of text documents can be searched by using **fulltext** queries since all text documents are automatically indexed when they are inserted into the database. The result of such a query is a ranked list of document titles with a score for each document indicating its relevance.

In both modes, the user can specify the **scope** of the search. Searches can be performed on the whole database or on certain collections (including their sub-collections) only. Of course, searching in a limited set of collections can be performed much faster.

3.3.2. Navigation

The previous explanations have shown that different navigation paradigms are implemented in Hyper-G. The organization of data in collections provide *hierarchical navigation,* the *hyperlink* concept allows navigation by following links, and the advanced *search* facilities are another tool for navigation. In addition, the bidirectional link concept (see 3.2.3.) allows the creation of a graphical view of the link structure around a given document. This feature is called a *local map,* as it shows incoming and outgoing hyperlinks of a document.

The power for navigation in the "information space" lies in the combination of these different paradigms. The user does not have to stick to one means of navigation, but may change it in a flexible way. For example, a user may issue a search to receive a list of documents and collection titles, then take a look at some documents or browse through the collection hierarchy to see 'what else' is available for a certain context, and then, when reading documents, the user may follow hyperlinks to related material.

3.3.3. Identification Modes and Access Rights

Hyper-G controls the extent to which a user is allowed to modify, insert, or retrieve data by using **access rights**. Not only user names are supported, but also group names may be specified. A user may belong to a particular *user group,* which automatically gives him/her certain access rights that are common for the group.

One disadvantage with identified users is that the information system may store information about every single command the individual user issues when using the system, which is possibly referred to as the "Big Brother" scenario. The approach taken in Hyper-G allows not only anonymous and identified users, but supports *four modes of identification.*

Anonymous	In this mode no monitoring of individual users is possible. On the other hand, no write permission is granted to anonymous users.
Anonymously Identified	Anonymous users may choose pseudonyms and passwords by which they are identified. However, the system does not know their real identity, but can recognize the user in later sessions and can restore user configurations and preferences. Write operations are only allowed for documents that are not visible to the public.
Semi-Identified	Identified users may choose a pseudonym and a password to use the system in semi-identified mode. The real identity is still known to the system but not to other users reading mail, annotations, etc. A user may use more than one pseudonym.
Identified	The identity of the user is known to the system and also to other users when looking at the "who-is-online" list, receiving mail, or reading annotations from identified users.

In connection with user identification another feature has to be mentioned. Hyper-G supports so-called **home collections**, which also work as individual entry points to the system. If a user identifies him-/herself, the client will display the user's home collection.

The user can copy references to other documents or other collections to the home collection using it as a personal "bookmarks/hotlist" for items of interest. This allows faster navigation to these documents when needed in subsequent sessions.

3.3.4. Interoperability with other Information Systems

In the design of Hyper-G, interaction with other existing information systems plays a significant role. An important aspect is that for the user, a remote databases should look like an integral part of Hyper-G preserving a consistent user interface.

The Hyper-G server is able to store "remote objects" (i.e. Hyper-G objects in the link database that are pointers to remote systems) for accessing **WWW**, **Gopher** and **WAIS** servers. This allows Hyper-G clients to contact such remote systems and presenting a "Hyper-G view" (as far as possible) of information on non-Hyper-G servers. Text documents from WWW servers are transformed into Hyper-G text documents, Gopher menus into Hyper-G collections, and WAIS queries into Hyper-G queries.

In the other direction, it is possible for Gopher and WWW client programs to connect to Hyper-G servers through gateways. For a Gopher client, the collection hierarchy is mapped into a Gopher menu. However, hyperlinks cannot be represented in the "Gopher world". When a WWW client requests a Hyper-G collection, the WWW gateway (see chapter 4.) will generate an HTML document containing a list of documents (including sub-collections) belonging to the collection with hypertext links to all these items. HTF[*] documents are converted to HTML documents. All data in a Hyper-G server can be searched for using the search menu generated by the WWW gateway.

Online sessions **(Telnet)** to remote systems are possible when necessary, although not ideal because they destroy the consistent user interface.

3.3.5. Document Cache

One problem of wide area networks is the sometimes slow access to documents because of network delay. Caching is a technique often used to improve performance of an information system. This is why in the Hyper-G system, requests from clients rather than being directly sent to document servers, are routed through a document cache [KAPPE92a]. If the document requested has previously been cached and can be found in the cache server, it is transmitted directly. Otherwise, the cache server retrieves the document from the document server, transmits it to the client and stores a copy of it in the cache. When space in the cache gets filled, the cache server removes documents following a "least-recently-used" strategy. A problem often faced in connection with caching is that the cache contains information that is out-of-date if documents have been modified. Notifying caches whenever documents get changed would cause further problems, as lists of all caches and what they have cached have to be maintained. In the design of Hyper-G this problem does not occur at all because objects receive a new object ID when they are changed. When a client requests such a document, the new version is retrieved and cached. As the old object ID is not reused, the old version in the

[*] HTF is the name of a SGML derivated markup language used by Hyper-G.

cache cannot be accessed any more and will be removed because of the "least-recently-used" strategy.

3.4. Applications

As mentioned before, Hyper-G is designed to be a *general purpose* hypermedia system. Because of its flexible design (no fixed user interface metaphor, modular software concept, extendibility, etc.) it serves not only as a testbed for the next generation of hypermedia information systems, but can actually be used for a wide variety of applications. Among these are:

- **University information systems** [KAPPE92], supporting research, teaching, communication, and administration tasks in a university environment.
- An infrastructure for **teleteaching** experiments and **Computer Aided Instruction**.
- **Information and communication systems** for large organizations.
- An infrastructure for **electronic publishing** [MAURER94a].
- A basis for the building of an **active communication/information system** [KAPPE93], including extended annotation facilities, discussion groups, e-mail, "active documents", etc.
- A tool for supporting **Computer Supported Collaborative Work**.

To pick out two examples, some additional remarks may be made on university information systems and active communication systems.

A university information system based on Hyper-G may contain general information about the university, statistical data, as well as information for administrative purposes (supported by the concept of closed user groups), electronic courseware, dictionaries, library information, etc. Hyper-G allows access to information systems of other universities in a consistent way and, therefore, an integration of information sources distributed all over the world is made possible.

The idea of an active communication and information system is to create a tool that provides not only browsing and navigation facilities, but also actively looks for relevant information and delivers it to the user. The aim is to let users specify their fields of interest in an "active" collection associated with a query. If an inserted document matches the query, such a collection may send it as e-mail to the user. "Active documents" would be an interesting way of implementing a reminder service or a personal calendar, with documents being not only text documents, of course, but all kinds of multimedia documents.

3.5. Comparison between Hyper-G and WWW

If we compare the previous statements characterising Hyper-G with the properties of the World-Wide Web, we can find several differences affecting the functionality and the way how these systems meet the requirements for a modern hypermedia information system.

Information Structuring and Navigation

First of all, it should be decided which system seems to be more appropriate to reduce disorientation of the user of an information system. This **disorientation**, also known as the *"lost in hyperspace"* syndrome, is caused by the following problems, which arise when somebody is looking for information in a large network of data sources. It is often difficult to gain an overview of the information available, to find information again, to know how much information there is on a subject and how much of it has already been seen, and to know whether everything important has been seen.

One of the differences between the two systems is how data can be organized. The WWW relies solely on the *hypertext* paradigm. Information is stored in documents containing links to other documents or to other parts of the same document. Hypertext links are the only way how information can be structured according to the node-link model. This is the reason why WWW databases are often large, flat networks, which are not well-structured.

In addition to hyperlinks, Hyper-G allows to group documents into collections (3.2.1.), which are used to build a *hierarchical structure* of the information stored. This has not only the advantage that the number of links can be reduced, but it results also in an additional navigation technique.

The node-link technique may be sufficient for small to medium amounts of data, but applied to large amounts of data it may lead to confusion and disorientation. The combination of the hypertext model and hierarchical structuring resulting in extended navigation tools makes Hyper-G more suitable to fight the "lost in hyperspace" syndrome.

Searching

As for the capability of information systems to provide searching facilities, Hyper-G offers a different approach than the WWW. Searching for attributes as well as searching in the contents of documents are fully integrated into the Hyper-G system (3.3.1.). Unlike the WWW, every document stored in a Hyper-G database can be searched for. As attributes (title, keywords, creation date, etc.) are stored in a separate database, searching for these attributes can be performed efficiently. Moreover, Hyper-G can offer sophisticated full-text search facilities because every text document is automatically indexed on insertion into the database.

Although the client server protocol used by the WWW supports searching (2.4.1.1.), the functionality heavily depends on the actual implementation of the server. Searching, especially full-text searching, is often implemented by using the script interface of the WWW server. This means that the actual search is performed by an external program and the way in which this is achieved varies from server site to server site. However, most of the sites which have a WWW server running at the moment do not provide any search engine at all, which makes navigation in the Web a lot more difficult.

The fact that attributes and links are stored separately in Hyper-G is of crucial importance for the searching facilities, because in a system where links are stored directly within documents, like in WWW, it is not possible to search the whole web for keywords. If users want to search for an item in different WWW servers, they have to specify a new query for each server.

In Hyper-G, the user may specify the scope of the search because of the additional information structure provided by collections. This functionality is missing in the WWW.

Protocol

A main difference between the two systems lies in the diverse client server protocols they use. The connectionless protocol HTTP (2.4.) can be kept very simple because no "state" has to be maintained. However, in WWW transactions, a lot of information has to be retransmitted on every request (format, authorization information, etc.) and a new connection has to be established for every transaction. The connection-oriented protocol used by Hyper-G allows the server to maintain information about the session, which has effects on user identification, user statistics, etc. Furthermore, clients can be notified immediately by the server when changes in the database are relevant for them.

Unlike WWW clients, Hyper-G clients need not connect to multiple servers. Should information from a remote server be needed, the Hyper-G server which the client is connected to fetches the information and delivers it to the client. This concept makes maintenance of user accounts and access rights easier and allows the implementation of a powerful caching mechanism.

The problem of out-of-date information in connection with **caching** is solved automatically by the assignment of new object IDs for modified documents by the Hyper-G link server (3.3.5.). Document caching in the WWW is normally done by clients, but it is also possible to use a cache mechanism built into some WWW servers, if they are running as proxy servers (see 4.7.1.). However, the use of expiration times for documents cannot really guarantee consistency of the cached data.

Multi-lingual Concept

Both WWW and Hyper-G can handle documents written in multiple languages. Although the HTTP protocol offers multi-lingual support, this is rarely used by WWW implementations at the moment. Apart from that, Hyper-G uses its cluster concept (3.2.2.) to implement multi-lingual support in a natural and much more consistent way.

In the WWW, a "chunk" of information consists of a single document. Hyper-G clusters, which can be seen as the implementation of information chunks, can consist of a number of documents, which may also be of different types.

User Identification

With regard to user identification, the WWW knows only two types of access rights: everything allowed or read-only access. Hyper-G offers different types of access rights for users and user groups in connection with four modes of identification (3.3.3.).

Links

As far as the link management is concerned, there is also a big difference between the two systems. In the WWW, all links are unidirectional and stored directly within documents. The Hyper-G link server maintains links in a separate database (3.2.3.). As links are bidirectional, this is of great importance in connection with system maintenance and extended navigation facilities (e.g. graphical map of the link structure). Moreover, links have types in Hyper-G, which allows to differentiate, depending on access rights, between private annotations, group annotations or public annotations and provides the possibility of defining other link types, such as "supporting argument", "counter argument", or "generalization", appropriate for "electronic" discussions and collaborative work.

Because of the separate link server, a Hyper-G system is able to support the system administrator to find and remove bad links. Systems like the WWW which only provide unidirectional links, are not able to tell which other documents contain links to a certain document. However, this information would be important when the document is deleted or modified, because otherwise it is impossible to guarantee database integrity, as links to non-existent or invalid documents remain in the database. Considering the link concept, Hyper-G seems to provide much better facilities relating to *automatic link maintenance*, which is particularly important in large multi-user environments.

4. Implementation of the Gateway

4.1. Goal

One of the basic features of Hyper-G as a "second generation" hypermedia system is the interoperability with already existing information systems (compare e.g. [ANDREWS94] and [KAPPE93b]). As the World-Wide Web gains more and more importance on systems connected through the Internet, it seems to be obvious that interoperability between Hyper-G and the WWW is a significant issue.

Apart from the integration of WWW databases into Hyper-G, an important aspect is the access of Hyper-G servers using WWW clients. Because of the very high distribution of WWW client programs, this seems to be an interesting way of offering Hyper-G functionality to an ever increasing number of users. To meet this requirement, a gateway should be provided which can be used together with each Hyper-G server allowing WWW browsers to connect to Hyper-G.

The development of the WWW gateway described in this chapter continues the path of an existing Hyper-G tool called *wwwgate*, which also offered Hyper-G access for WWW clients which, however, suffered from a few weaknesses. The connectionless access imposed by the HTTP protocol required establishing and ending a Hyper-G session for every individual WWW request. This resulted in

- unnecessary overhead for the Hyper-G gateway/server connection,
- inability to use those Hyper-G features that depend on a connection-oriented approach,
- difficulties in obtaining realistic usage statistics.

A comment may be made on the last point. As every single request was handled as a new session by the Hyper-G server, the number of sessions caused by wwwgate could not be compared to the number of sessions using native Hyper-G clients. The approach to count consecutive requests initiated from the same host as one single session was only an approximation to the actual number of sessions. Furthermore, this concept became more and more inaccurate as the number of requests through the gateway increased.

Apart from that, the connectionless approach did not allow the use of the distributed mode of the Hyper-G server. In the distributed mode, object IDs are assigned on a per-session basis. In contrast to the local object IDs used in the non-distributed mode, these IDs cannot be used in consecutive sessions. The use of the non-distributed mode has, of course, the limitation that only information residing on the local server can be accessed.

Since wwwgate was developed before the release of HTTP 1.0, only the simple protocol HTTP Version 0.9 was supported (see 2.4.), which, although understood by all WWW clients for back-compatibility, lacks a lot of useful features (e.g. MIME headers, method POST, etc.).

In order to overcome the difficulties mentioned above and to extend the functionality of the WWW-to-Hyper-G access, the development of a new WWW gateway became necessary.

This project has the major aim to study possibilities how certain features of Hyper-G relying on a connection-oriented client server protocol can be made accessible through the connectionless protocol used by the World-Wide Web. Design goals have been:

- Improved simulation of **Hyper-G sessions** with the ability to **distinguish users**.
- Use of the **distributed mode** of the Hyper-G server.
- Possibility of **user identification**.
- Support of the advanced **search facilities** provided by Hyper-G.
- Access to the **user status** (who-is-online list) of the Hyper-G server.
- Support of the **multi-lingual concept** of Hyper-G.
- A **user interface** in **English** and in **German**.
- Use of **HTML fill-out forms** for the user interface design.
- Support of the **HTTP** protocol version **1.0**.
- Possibility of an easy **customization** of the software and the user interface.

4.2. Basic Concept

4.2.1. Providing Connection-Oriented Access with a Connectionless Protocol

In contrast to the concept of handling every single WWW request as a separate Hyper-G session, the implementation of the new gateway described in this chapter is based on the idea of keeping up the connection between the gateway and the Hyper-G server during the processing of consecutive HTTP requests.

The gateway can therefore be seen either as a *Hyper-G client* by the Hyper-G server or as a *WWW server* by the WWW client that issues requests.

Therefore, we can distinguish two interfaces of the gateway:

- The ***WWW side*** handles a transaction between WWW client and gateway according to the HTTP protocol. That is why for every new request a new connection between the WWW client and the gateway has to be established, which is, of course, initiated by the client. This connection is immediately closed after the response to a single request has been sent.
- The ***Hyper-G side*** maintains a *session*. This means that a connection between the gateway and the Hyper-G server is established when the gateway receives the first request of a new user (i.e. a user for whom currently no Hyper-G session is being maintained). This connection is being kept up during the processing of consecutive requests from the same user and is closed at the end of a session. The problem of determining the actual end of a session is discussed later in this text.

Apart from being necessary for maintaining Hyper-G sessions, this concept has the advantage that the gateway is able to store information about the client's request and to keep this information until the next request from the same client. The gateway is maintaining a **state**, although the client/gateway protocol HTTP is actually stateless. The stored information can be for instance the object ID of the last retrieved object, the user name of identified users, information needed to distinguish requests from different users, parameters influencing the optical appearance of collection lists, etc. This information is used, for example, to determine the scope for searches, for user identification purposes, for keeping certain parameters which can be set by users, etc. If this information had not been preserved between the processing of consecutive requests, the client would have been forced somehow to transmit this information with every single request. The only way of making this possible using the HTTP protocol would have been the generation of a very long URL containing all this information, which was not considered to be a very practical solution.

However, a few questions arise in connection with this concept. At first, it has to be made clear when a Hyper-G session starts and when it ends. The first question can be answered easily, as a new Hyper-G gateway/server connection can be established as soon as the first request from a WWW client is received by the gateway. This connection between gateway and server is being used for transmitting messages in order to fulfil requests from the same WWW client during the whole session.

On the other hand, the **end of a session** is not easy to determine because a WWW client, bound by the connectionless protocol HTTP, does not know the existence of a Hyper-G session. If the user of a WWW client does not request any more information, the gateway will simply receive no more requests from that particular client. The only practicable solution to this problem is to let the gateway wait a certain period of time for requests and then, when no request is received from that particular client during this period, close the connection to the server which ends the Hyper-G session. The value of this *time-out* has to be set according to practical experience, but can be altered by the system administrator.

But what happens if the client sends a request after the session timed out? The best way to handle this is by treating the request as a first request of a new user and start a new Hyper-G session. Of course, settings stored during the previous session are lost, but this is not really a problem as it does not effect any important information and besides, this does not occur very often if the period of the time-out is sufficient. In the worst case, a user has to identify her/himself once more and the settings for the preferred language and the appearance of collection lists are set to their default values again.

Another important issue is how requests, that arrive in an arbitrary sequence at the gateway, can be distinguished in respect to their sender in order to assign them to a particular Hyper-G session. Using the internet address of the client host for user differentiation is definitely not sufficient since multiple client programs can run on the same host machine. The only possible solution is to include some kind of identifier in the URL when used in a request to the gateway. As the path part of a URL is not actually understood by the client (compare 2.3.2.), a string can be included in the URL which allows the gateway to recognize the request as a further request of a client for which a Hyper-G session is maintained. Consequently, if the identifier is missing or if it is invalid (i.e. it cannot be assigned to a current session, probably because it refers to a session that has been timed out), the gateway can assume that the request is the first request from a new user and it will therefore set up a new session. The exact format of the string identifying the session is explained in 4.2.4. in detail.

A new identifier, called *session key* in the following, has to be assigned to every new session. The only problem that still remains at this stage is how to tell the client to use a different URL containing the session key in further requests. One way of solving this problem is to make sure that every anchor that is part of a document sent as a response to the client contains a URL with a session key. Following those links, a client will automatically use the modified URL. As simple as this idea is, it does not work if the document does not contain any links. Of course, one cannot say that such documents are real hypertext documents, but in reality this happens very often, especially if existing text documents that are not originally generated as hypertext are made accessible through the WWW gateway. Other types of documents (e.g. images, films, sounds, etc.) have usually no anchors at all (at least in the World-Wide Web). If users request such documents with their first requests, the gateway will not have a chance to tell clients about the new URL to use in further requests.

A possible solution to this problem is the idea of a standard document, called *entry page*, to be sent as a response to every first request from a client. This entry page contains general information about the WWW gateway, but, what is important, only one anchor. This anchor contains a URL with a session key. Following this link, which usually looks like an "OK" button to the user, the client will use the modified URL and the gateway will send the actual response to the first request this time. To guarantee that the client will use the modified URL with all further requests, the gateway makes use of *partial URLs* (compare 2.5.3.1.). Anchors which are sent to the client contain either complete URLs including session keys or partial URLs without scheme identifier, host part, and session key. When resolving partial URLs the client will also add the session key.

Although the use of a standardized entry page is an acceptable solution, it can be improved by making use of the *redirection feature* of the HTTP 1.0 protocol (see 2.4.2.1.). When receiving the first request of a client, the gateway responds with status code 302 and provides the modified URL to be used with further requests in a response header line. The client then automatically retransmits its request, but uses the new URL this time. This solution makes the entry page obsolete. Nevertheless, the option of sending an entry page in certain cases, for example if no particular document was requested, should remain.

4.2.2. Parallel Processing of Requests

Performance considerations show that the amount of time to be spent on the processing of a request has to be taken into account for the design of the gateway software. Processing can sometimes take a few seconds. Particularly, this could be the case when the request causes a fulltext search with a large scope to be performed by the Hyper-G server. The amount of time spent on processing may vary depending on network load, server load, etc. If the gateway is processing consecutive requests from several clients, some requests might have to wait for acceptance by the gateway for a long time. The problem is not so much the amount of time the client has to wait until the response has been received from the gateway since waiting a few seconds for the completion of processing in some rare cases is acceptable for the user. But a possible problem could be the fact that the time until a connection initiated by the client has been accepted by the gateway can be too long, as clients can only try to connect for a short period of time before giving up. Otherwise it could occur that they wait unnecessarily long for a server (or gateway) that is not working.

This shows that it is important to accept a connection from a client as soon as possible. If a connection is not accepted until the previous request had been completely processed, many connections would not be accepted at all, giving the wrong impression that the gateway (and consequently the Hyper-G server) was down very often.

To avoid this problem, the task has to be split in such a way that the process which accepts connections from the "outside" is not burdened with the actual processing of the request, but instead is allowed to accept further requests while others are still being processed. Accepting requests from WWW clients has to be performed by one single process, called the ***master process*** or the ***master*** in the following, because all clients use the same port to connect to the gateway. The actual processing, though, can be split between concurrent processes, each of them processing those requests that were sent by the same client and therefore belong to the same session. Each of these processes, called ***slave processes*** or ***slaves*** in the following, is keeping up a connection to the Hyper-G server and is storing all information that is relevant for a particular session as long as requests are being received from the client.

However, a slave process cannot send its responses to the client itself because it does not keep up the connection to the outside, since a new client/gateway connection is established and closed for every single WWW transaction only by the master. The master process has to pass the request to one of the slaves somehow. Doing this directly would not be a good idea because in this case, the master process would have to wait until the slave has sent the response in order to pass it to the WWW client. The master would not be able to accept further requests while others are being processed, which returns us to the problem previously discussed. Therefore, the task of passing the request, waiting for the response, and finally returning the response to the client has to be performed by another process.

Such a process is generated by "forking" of the master process. In this way the process, which will be called ***passing process***, is able to inherit the open connection to the client from the master, then passes the request to one of the slave processes using the differentiation scheme explained in 4.2.4. to contact the right slave, and sends the response from the slave to the client. After that, the passing process exits. This makes clear that a new passing process has to be generated by the master whenever a new request from a WWW client is received. After the task of handling this request has been transferred to a child process, the master is ready to accept the next connection from a client. The child handles the request with the help of one of the slave processes, transmits the response to the client, and exits.

How the whole concept described so far is implemented on a UNIX system is described in the next section (4.2.3.).

A further advantage of having a separate process running for every Hyper-G session is the easy maintenance of all session specific data. No linked lists or similar data structures have to be used for storing session information because data is stored in a decentralized manner. When such a process is generated by forking it receives its own set of data values which can be modified only by this process without interfering with data of other sessions.

4.2.3. Process Model

The considerations previously made in 4.2.2. make clear that three types of processes are necessary for the implementation of the concept described.

The **Master Process** It is the only process started when the WWW gateway is set up, hence, it is the parent process for all other processes. Its task is to listen on the port which is assigned for WWW transactions (the standard port is number 80) for connections initiated by WWW clients. When it accepts a connection, it "forks" generating a child process, which handles the request. The master can close its connection to the client immediately after forking and is then ready to accept the next connection. As its only task is accepting connections and forking, which can be performed very fast, connections do not have to wait very long for being accepted.

The **Slave Process** Generated by the forking of the master process, the child determines by examining the first line of the request from the client whether the request belongs to a session already in existence or whether it is the first request from a new user (see differentiation scheme explained in 4.2.4.). If the child recognizes a first request, it works as a slave process. At first, it chooses a port number to be used for further requests. This port number is used as a part of the session key included in the response. After that, it opens a connection to the Hyper-G server, starts a session, reads the request from the WWW client using the connection inherited from its parent, and sends a response which is either a standard entry page containing a link with the new URL or a redirection message (compare 4.2.1.). In either case the client is informed in this way to use the new URL, which includes the session key, for further requests. After that, the slave closes the connection to the outside and waits for connections on the port whose number has been previously chosen. When it has accepted a connection initiated by a passing process, it reads the request, processes it (usually by transforming it into a Hyper-G request for the server), receives the response from the Hyper-G server, transforms the response again if necessary, sends it to the passing process, and closes the connection to the passing process. By waiting for the next connection on its assigned port the whole procedure starts again. Only when the slave has been waiting for a certain amount of time exceeding the time-out period, it exits closing the connection to the server, which automatically ends the Hyper-G session.

The **Passing Process** The first action a child of the master process takes is to examine the URL of the request from the WWW client. If a session key is found, the child will try to contact the corresponding slave process. If this fails, or if the slave does not accept the request for what reason ever, the child will become a slave process and will process the request by itself. If a slave process is ready to accept the request, the child will work as a passing process. This means that it will read the request from the client using the connection inherited by the master, pass it to the slave, wait for the response, send the response to the WWW client and exits closing the connection to the outside.

All this shows us that the *"lifetimes"* of the three types of processes are different. The master process stays alive as long as the WWW gateway is working. Without the master process no new connections can be accepted from WWW clients. Of course, there is only one master and it is the parent of all other processes.

The lifetime of a slave is the duration of a Hyper-G session. The slave, by using an internal timer, also determines the end of a session after a fixed period of inactivity. Many slaves can work concurrently, each responsible for one Hyper-G session. If the master exits or is killed for some reason, all children can process their current requests independently from the master and stay alive keeping up the connections to the Hyper-G server until their time-out is reached. This is an interesting feature, because when a new master process is started by the system administrator or a controlling process within the time limit of the slaves, new passing processes generated by the master on receiving requests from clients can contact those slaves, as if nothing had ever happened.

Each passing process is alive as long as a WWW transaction takes place. After sending the response to the client they exit.

The following **example** shall illustrate what has been described so far. It shows what typically happens as far as process interaction is concerned when a WWW gateway receives several requests from a number of WWW clients.

When a WWW-to-Hyper-G gateway has been started, only one process, the master, exists. The master is waiting on port number 80 (this is the default for WWW connections) for incoming requests. As you can see in figure 4.1, the first client (W3Client A) tries to connect to the gateway.

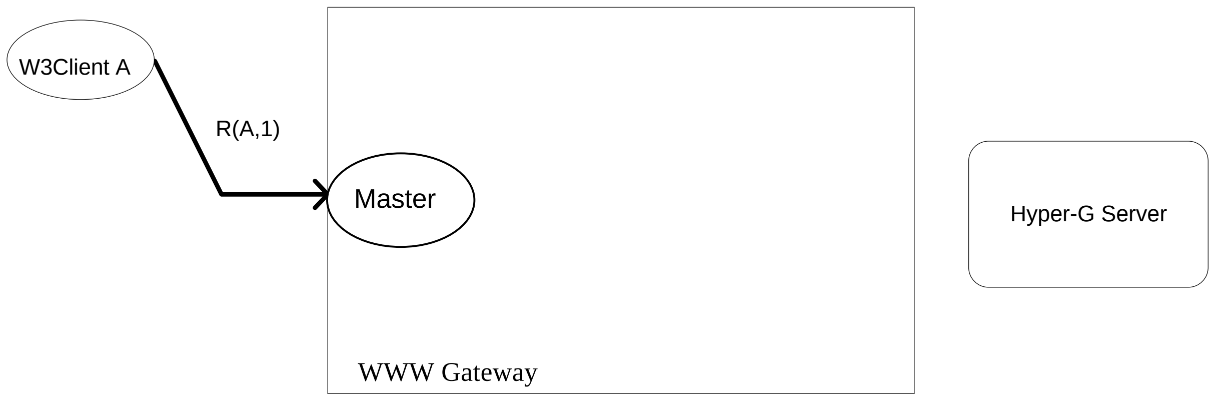


Figure 4.1

After accepting the connection, the Master forks generating a child process. Immediately after that, the master closes its connection to W3Client A and is then ready to accept other connections. The child examines the URL of the first request R(A,1) and does not find any session key in it. That is why the child works as the first slave process Slave A. It establishes the connection (A) to the Hyper-G server, assigns a new session key, and sends the response (usually a redirection message) to the client using the connection inherited from the master (see figure 4.2).

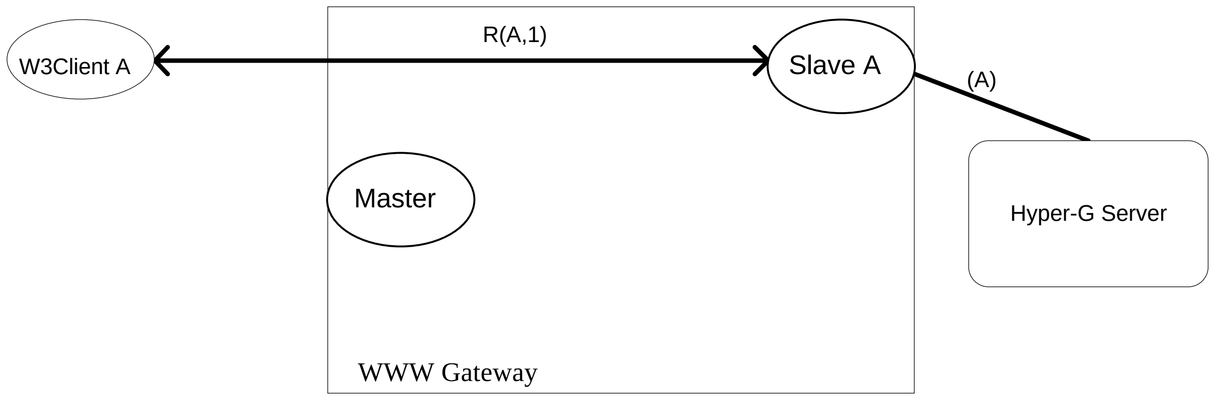


Figure 4.2

In the meantime the master has accepted a connection from W3Client B (see figure 4.3).

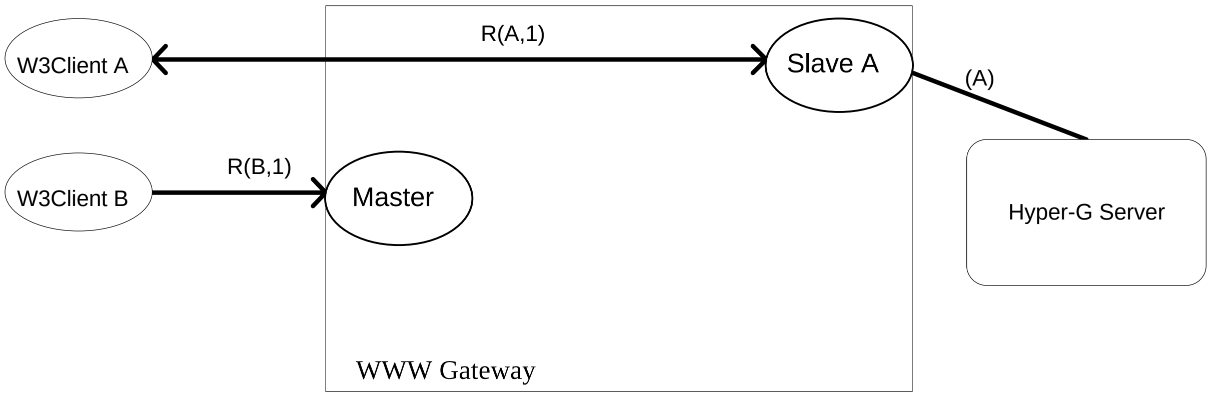


Figure 4.3

The same as what was shown in figure 4.2 for R(A,1) happens with request R(B,1). Another child process works as Slave B (see figure 4.4) starting a new Hyper-G session for W3Client B. You can also see that Slave A has closed the connection to W3Client A in the meantime after sending the response to R(A,1) and that W3Client A then tries to connect to the master in order to send its next request R(A,2).

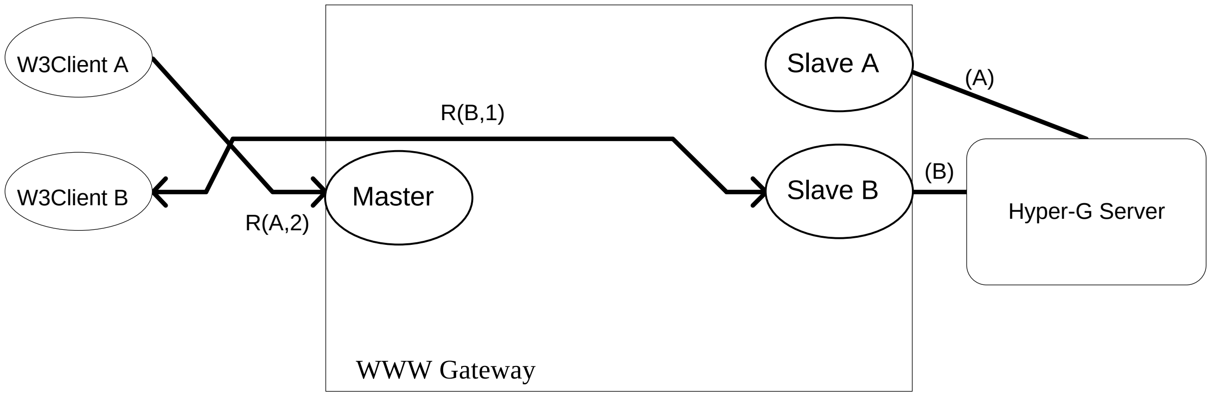


Figure 4.4

This causes the master to generate another child process (see figure 4.5). This child finds out that the session key contained in the URL of R(A,2) refers to an existing slave process. Therefore, it decides to play the role of a passing process (P A2) for this request and initiates a connection to Slave A. In this way R(A,2) is passed to Slave A, which processes the request. This is usually done by transforming the request into a request for the Hyper-G server which is connected to Slave A by connection (A). Sometimes the request can be processed by Slave A without the help of the Hyper-G server. In both cases, the response is sent back to W3Client A via the process P A2. In the meantime, the master has accepted a connection from another client (W3Client C).

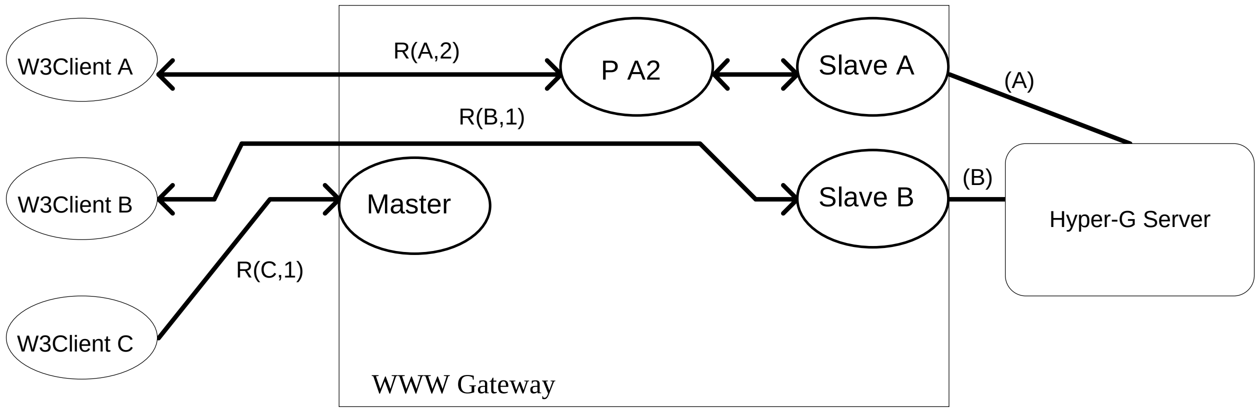


Figure 4.5

What you can see in figure 4.6 is that the master has generated another child, which is now processing the first request R(C,1) from W3Client C as a new slave process Slave C. While this takes place, the other two slaves have completed processing of requests R(A,2) and R(B,1) respectively, and now both of them are waiting for connections from passing processes to be generated by the master. You can also see that W3Client B tries to connect to the master again.

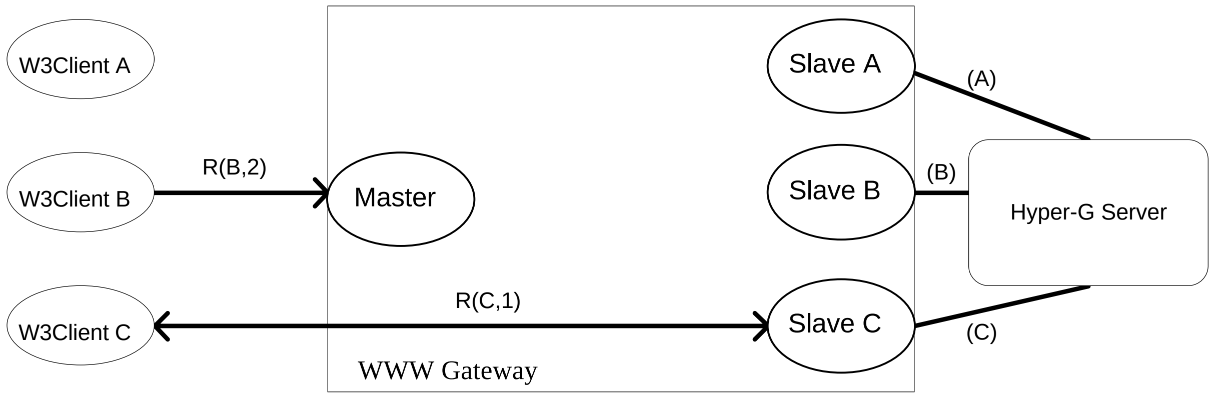


Figure 4.6

Now let us skip a few steps and take a look at the gateway after several requests from W3Client B and W3Client C have been processed. What can be seen in figure 4.7 is that currently requests R(B,7) and R(C,9) are being processed by their corresponding slaves and passing processes. W3Client A, however, has not sent any request for a certain period of time exceeding the time-out limit of Slave A. That is why Slave A exits closing the connection (A) to the Hyper-G server.

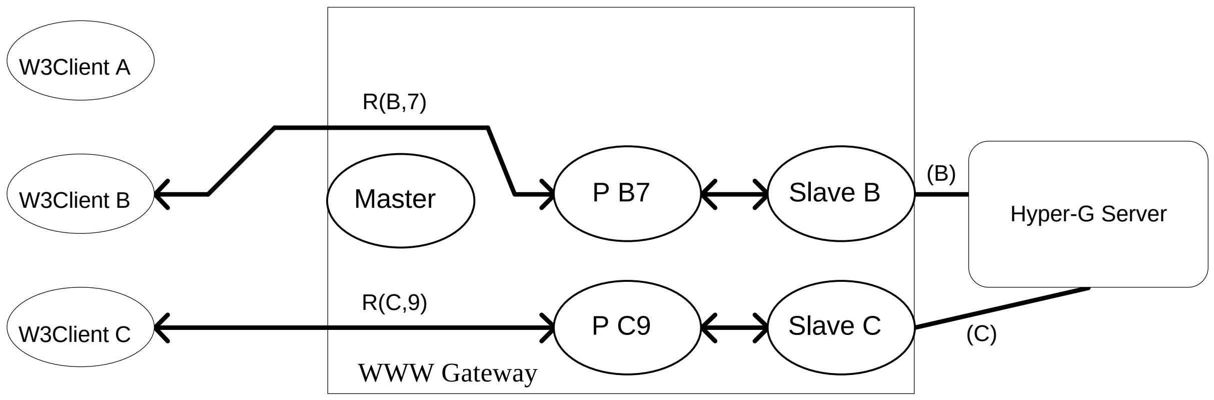


Figure 4.7

If W3Client A came along once more with another request, a new session would have to be established by a new slave process.

4.2.4. Differentiation of Users

The whole concept of using the connectionless client/server protocol HTTP to allow a connection-oriented access to Hyper-G servers, as explained in 4.2.1., is based on the idea of providing some mechanism by which requests arriving in an arbitrary order can be distinguished with regard to the session they belong to. The differentiation scheme used in the implementation of the WWW gateway is explained in the following.

At first, an identifier is defined which has to be part of the URL of all further requests from the same WWW client. This identifier, called **session key**, has to be assigned at the processing of the first request from a particular client. It is a string representation of a 32 bit hexadecimal number and consists of a 16 bit *port number* and a 16 bit *random number*. The port number refers to the port at which the slave process that maintains the particular Hyper-G session listens for connections from passing processes.

The order in which the 8 characters appear in the 32 bit hexadecimal number is as follows. The 3., 4., 5., and 6. character form the port number and the 1., 2., 7., and 8. character form the random number.

Example:

The port number 43981_{dec} (= $ABCD_{hex}$) together with the random number 4660_{dec} (= 1234_{hex}) result in the

Session Key:	1	2	A	B	C	D	3	4

The reason for the port number is obvious. It is used by the passing process to connect to the right slave process. The random number provides an additional means of preventing a request from being sent to the wrong slave.

Whenever a connection from a WWW client is accepted by the master, the task of handling the request is transferred to a newly generated child process. This child examines the first line of the request in order to find a session key in the URL. If one can be found, the process extracts the port number and uses it to contact the slave process listening on that port. If this is successful, the internet address of the client host connected to the process and the session key are sent to the slave, which compares both with the corresponding values stored. Only if the session key matches the key assigned when opening the session *and* the internet address of the client host matches the internet address of the host which issued the first request of this session, the slave will respond with "OK". In this case, the child process now keeping up two connections, one to the client and one to the slave, works as *passing process* for this request, sends the request to the slave, and exits after the response from the slave has been returned to the client.

In all other cases, the child process assumes that the request has to be treated as a first request of a new user and therefore, it decides to work as a new *slave process*. Those cases are:

1. No session key can be found in the URL.
2. Although a session key is present, no process is listening on the port referred to by the port number.

3. A slave can be contacted, but it returns "Unknown user" because either the internet address or the session key or both did not match the values significant for the session maintained by the slave.

In the first case, it is obvious that the request is the first one issued by a particular client as the gateway has not yet had the chance to pack a session key into the URL.

The second case usually happens when the time-out limit of the session referenced by the session key was exceeded. The gateway simply solves this problem by starting a new session. This has the advantage that in most cases users do not even notice that a new session has been established for them. Also, if someone uses a URL which was added to a "hotlist" some time ago and which therefore contains a session key that does not, of course, correspond to an active session any more, a new session will be established, which is evidently the right reaction (compare alternative concept 4.3.).

Finally, the third event should occur only in rare cases. However, it could happen that a client uses a URL containing a session key from which a port number is extracted that is already in use by an active session, although the session had been opened for another client.

As soon as a child process finds out that it has to work as a *slave* and it has to process the request by itself since the request does not belong to any active Hyper-G session, one of the first actions it takes is to assign a session key to the newly opened Hyper-G session by asking the operating system for a new port number and by calculating a random number. Together with the internet address of the client host, this session key is stored by the slave as well as it is packed into the path part of a URL which is returned to the WWW client in either a redirection message (see 2.4.2.1. HTTP status code 302) or a standard entry page with a link to this URL. This causes the client to use the modified URL containing the session key with its next request. The exact formation of such URLs produced by the gateway is described in 4.2.5.

A remark may be made on **security considerations** concerning the differentiation mechanism. As previously stated, it is possible that in some rare cases the session key packed in a URL of a request contains a port number that leads to a slave process which is actually keeping up a session for a different client. If this should happen, it does not at all represent a problem because it is highly unlikely that both 16 bit random numbers are also identical and if this should ever be the case, it is practically impossible that both requests were issued by different clients running on the *same* host machine.

4.2.5. URL Formation

This section answers the question how objects stored in a Hyper-G database can be addressed by URLs in order to be retrieved by WWW clients via the gateway. These URLs are, of course, generated by the gateway and inserted into anchors before documents are being sent to clients. But there are also some URLs that do not refer to any document in the Hyper-G database. These URLs are defined to be used for interaction with the gateway, often without causing any transaction between the gateway and the Hyper-G server.

URLs generated by the gateway can be divided into *three categories*.

- URLs that address documents or links stored in the Hyper-G database.
- URLs that refer to components of the gateway's user interface (e.g. inline icons, online help pages, search form, options form, who-is-online list, etc.).
- URLs that cause certain actions to be taken by the gateway, often but not always causing a transaction between the gateway and the Hyper-G server (e.g. user identification, performing a search, setting the preferred language, modifying the appearance of collection lists, etc.).

URLs of all categories have a common host part, which consists of the domain name of the gateway host and an optional port number. If the port number is omitted, the default WWW port (number 80) is assumed. The scheme specifier is usually "http:" (see 2.3. for a general description of the URL syntax).

What is different is the *path part*. It can either consist of a session key and an identifier or just an identifier. Whether there is a session key present or not, depends on the differentiation mechanism explained in 4.2.4. The session key is only needed to distinguish clients. It is actually removed from the URL after the right slave process has been contacted. Which document or action is addressed by the rest of the path, the identifier, has nothing to do with the session key. If it is present, though, it always comes as the first part of the path, followed by a slash '/' to separate it from the identifier. This is necessary in order to use partial URLs, in which case everything except the identifier is omitted and the client has to fill up the rest with the session key included.

A typical complete URL has the format:

```
http://gatewayhost/sessionkey/identifier
```

Depending on the category previously mentioned, the identifier can either be

- a **document identifier**,
- a **gateway file identifier** or
- an **action identifier**.

<u>Document Identifiers</u>

Document identifiers consist of a *document type letter* followed by either a *document name* or a *document number*. Document names are only used when referring to "named" Hyper-G objects, which are usually collections only. Document numbers are the string representation of either the global object ID of a Hyper-G object (see [KAPPE92a]) or the session dependent object ID (used for anchor objects only).

The document type letter is one of the following capital letters:

C	for collections (and consequently also for clusters)
T	for text documents (usually stored in HTF format and converted into HTML)
M	for "multimedia" documents (images, sounds, films, postscript documents, etc.)
A	for anchor objects

Examples:	`Cabout`	identifier of a collection with name "about"
	`C0x811b9908_0x00066fc8`	identifies a cluster
	`T0x811b9908_0x000cb096`	identifies a text document

<u>Gateway File Identifiers</u>

User interface elements (especially forms) provided by the gateway are often based on a special set of files in HTML format (see 4.6.1.). These files are not contained in the Hyper-G database. They only belong to the gateway. When a client requests for example a search form or an online help page, the gateway reads a particular file, replaces some place holders that are sometimes contained in it with certain values, and sends the file to the client.

There are two types of file identifiers. The first one refers to the files of the gateway in HTML format. Such a gateway file identifier corresponds to the name of the HTML file in question. Actually, there exist at least two versions of each HTML file, an English and a German version. They have the same name but are in different directories. Which of them is sent to the client depends on the setting of the preferred language by the user (compare 4.5.6.3.).

The second type refers to inline icons provided by the gateway (e.g. document type icons in collection lists, command icons, etc.). In this case, the gateway file identifier consists of a capital "I" followed by the name of the corresponding image file.

Examples:	`options.html`	identifies the file containing the options form
	`Icluster.gif`	identifies the file cluster.gif containing the document type icon for clusters

There is only one restriction that applies to file identifiers for the gateway's HTML files. They must not start with either "C", "T", "M", "A" or "I" in order to distinguish them from other types of identifiers. The safest solution, also in respect to future extensions, is to let them start with a lowercase letter.

Action Identifiers

They are used in ACTION attributes of fill-out forms and therefore, URLs containing such identifiers are sent to the gateway only with HTTP method POST, as this is the method that has to be used for submitting forms to the gateway.

An action identifier is one of the following five strings:

```
identify
status
searchit
language
display
```

They can be seen as **gateway commands**. When used in the URL of a POST operation, the gateway will interpret the data body of the request as a list of name/value pairs for the corresponding command. The individual meaning of the commands is explained in 4.6.2.

Non-Hyper-G URLs

In addition, the gateway can handle URLs that refer to a remote document that is no Hyper-G document at all. The host part of such a URL does not correspond to the name of the gateway host. Such URLs are sent to the gateway when it is used as a proxy gateway (see 4.7.2.). In such cases, the gateway uses the Hyper-G document cache server to retrieve those documents.

4.3. Comparison with an Alternative Concept

It was mentioned in 4.2.1. that some mechanism had to be found to differentiate requests arriving in an arbitrary order in respect to their sending clients, in order to assign them to a particular Hyper-G session. Including some kind of identifier into the URL of every HTTP request is the only possible solution. However, including a session key into the path part of a URL, as described in 4.2.4., is only one way of solving this problem.

Another possibility would be to make use of the fact that every slave process (see 4.2.3.) is listening for connections at a different port. The idea is to let clients **directly connect to slaves** using these port numbers. This can be achieved, of course, if these port numbers are put into the host part of the URLs used with further requests. The same possibilities as discussed for the other concept can be applied here as well in order to tell the client to use a different URL. Either the new URL is placed into an anchor of a standard entry page, or the redirection feature of the HTTP protocol is used.

Example: A newly generated slave process gets the port number 7893 from the operating system. Therefore, the slave tells the client to use the following URL with the next request.

```
http://domainname:7893/
```

This approach has the advantage that no passing processes are needed which makes it easier to implement. In other words, the task of directing requests to the right slave process is performed by the operating system.

The process model is similar to the one described in 4.2.3. For both concepts there is a single master process, which listens on the standard port of the gateway for connections from WWW clients. When the master accepts a connection, it forks generating a child process, closes its connection to the client and is then ready to listen again. The child process inherits the connection from the master and processes the first request using this connection.

The difference in process interaction between the two concepts is that in the alternative concept the client sends its next request directly to the child process because of the different port number in the URL. The master process receives only the first connection of a particular client. Therefore, the child processes are all slave processes. No passing process is needed for keeping up a connection to the client during the processing of one HTTP request and no differentiation mechanism for requests has to be implemented, as requests from different clients do not enter through the same port but are already directed to the right slave processes by using different ports.

Although this concept is in fact simpler and therefore easier to be implemented, it has one significant disadvantage. **It does not allow** to keep a URL obtained from the WWW gateway in a **"hotlist"**.

The reason for that is simple. A request which was issued after the time-out limit for its session (see 4.2.1.) had been exceeded cannot be accepted because the process which the client wants to connect to does not exist any more. Therefore, such URLs can only be used temporarily. If a user wanted to use a "timed-out" URL, the port number would have to be deleted from the host part (or edited if the gateway does not use the standard WWW port) in order to contact the master process. In general, this is not considered to be acceptable for users.

In contrast to this concept, the other mechanism using a session key in the path part of the URL is able to handle "timed-out" URLs, as all client connections are accepted by one single process, the master, at first. The session key is removed from such a URL when it is recognized that the corresponding session does not exist any more. The request will then be treated as if it was the first request for a new session. This is in fact the only way of dealing with this properly, because if somebody uses a URL kept in a hotlist to connect to the gateway, the user actually intends to start a new session. Although the session key contained in the URL is changing from one session to another, you will always access the same object, identified by the rest of the URL.

In addition to the problem of URLs stored in hotlists, another drawback of the alternative concept of using port numbers as a means of differentiating requests has to be mentioned. As the port number is a 16 bit number the mechanism is not so secure as the use of a 32 bit session key. Although this could be compensated for by using an additional session key (e.g. a random number) as a part of the path, the question arises what a slave process should do with a request when the random numbers do not correspond to each other or the client host differs from the host name stored at the first request of the session.

The master process has no control over this request, as it is directly sent to one of the slaves. This is why the master cannot generate a new child process to handle the request.

Either the slave process has to fork, generating a child to process this request, or the slave must refuse to process the request, which is not a very "user-friendly" solution. A third possibility would be to use the redirect feature of the HTTP protocol to direct the client to the "main port" of the gateway, on which the master is listening.

Because of the disadvantages of this concept, it was decided to base the implementation of the current WWW gateway on the improved concept, described in 4.2.

4.4. Structure and Functionality of the Program

The software of the WWW gateway was written in the C++ programming language on UNIX workstations. The source code mainly consists of

- the **main module**,
- the code for the **class WWWSlave** and
- a number of **utility functions**.

Among some existing Hyper-G libraries used by the gateway program there are a SGML parser that converts HTF into HTML texts and an application library, also used by the Hyper-G terminal viewer, which serves as a program interface for Hyper-G client-server communication (e.g. insertion of anchors into documents using information from the link database).

4.4.1. The Main Module

The main module contains the functionality of the master process and of the passing process (see 4.2.3.). At first, it handles command line parameters, environment variables, and program defaults. Then it listens on a connection whose port has either been opened by another tool called hgbindport, or has been given on the command line. If a system port number is used for WWW transactions, which is the usual case, either hgbindport has to be used, or the gateway has to be started by a superuser. The Hyper-G tool hgbindport opens a system port number, for example the default HTTP port 80, and assigns it to file descriptor number 3, which is then used by the gateway to attach the connection buffer.

The main task of the master process is to listen for connections of WWW clients and to fork whenever a connection has been accepted. The main module also contains some code of the newly generated child process. First of all, an internal timer is set which is used to terminate the child process after a certain period of inactivity. Then the first line of the request is read in order to decide whether the request is a normal request to the gateway or a proxy request for a remote resource. A proxy request is processed by an instance of WWWSlave, which requests the document from the Hyper-G document cache server, and then the child process exits.

When a normal request has been received, it has to be decided whether an existing process is ready to process the request, or the child process has to process it by itself. In the first case, the request is passed to the running process, the response is read and sent to the client and then the child process exits.

In the second case, an instance of WWWSlave is generated, which processes the request and then waits for further requests.

4.4.2. The Class WWWSlave

The class WWWSlave contains all data and code necessary to process requests from a single WWW client, to open a session to the Hyper-G server for this client, and to keep the "state" of this session during consecutive HTTP requests.

The constructor of WWWSlave assigns a new port number and a new session key for this session. The port number will be used by passing processes to connect to the slave process at further requests. An instance of HgStub, a class that is able to pass objects between Hyper-G clients and servers, is generated. A new Hyper-G session is opened and the user is logged in with user name "www-anonymous".

The class WWWSlave has two public functions, which are defined as follows:

```
void WWWSlave::firstRequest(rpcbuf& connection, const RString&
firstline, const RString& method, const RString& url, const
RString& protocol)
```

```
void WWWSlave::nextRequest()
```

The function `firstRequest` is called when the newly generated slave process has found out that it has to process the request by itself. The first parameter is the connection to the WWW client. As the first line of the request has already been read and analyzed by this process, the different parts of the first line (HTTP method, URL, protocol specifier) are given as parameters. After processing the request and sending the response back to the WWW client, the connection is closed.

In contrast, the function `nextRequest` is called after the first connection has been closed. It waits for a connection on the port which was assigned by the constructor for this session, reads a request, processes it, sends the response back, and closes the connection. This function is called in an "endless" loop, which can only be interrupted by the timer.

A request is processed in the following way. After reading the whole request including the header lines, a distinction is made whether or not it is a first request from a new user.

In case of a first request, either a redirection status code or an entry page is sent depending on the URL and on the options set at program start (see 4.6.3.). Because of some clients that do not support redirection, a "minimal" entry page is sent even if a redirection code is used. This entry page is normally ignored by clients that understand the redirection code.

If the request is not the first one for this particular session, the next point to look at is the HTTP method. Methods GET and POST are supported.

In case of method **GET**, the type of document to transmit is recognized by the first letter of the identifier in the URL (see 4.2.5.). The following "types" have to be distinguished: Collection lists, text documents, "multimedia" documents, anchors, icons, and HTML files.

A *collection list* is generated by fetching the corresponding Hyper-G objects and generating an HTML representation of a collection with links to all its members. Document type icons are included into the HTML text as inline images and an optional collection head document is sent before the actual list.

Clusters are special types of collections. The HTML representation of a cluster depends on the number of its documents and their types. A list of all documents that are no texts and that do not have the presentation hint attribute "Hidden" is made. This list looks like an ordinary collection list. After that, the function handling clusters tries to choose a single text document that is written in the user's preferred language and sends its HTML representation.

Documents of other types (e.g. *images, sounds, films, 3D scenes, HM-Card documents, postscript documents, compressed postscript documents*) are sent preceded by a MIME header corresponding to their types. For other "*generic*" documents MIME content type "`*/*`" is used, which usually causes WWW viewers to simply save such documents to disk.

Anchor URLs are resolved by fetching the target document and processing it depending on its type. If the target is a remote document (e.g. WWW, gopher, telnet, or ftp) a redirection status code is sent in order to direct the client to the remote server. For clients that do not support redirection, a short HTML text containing a link to the remote resource is sent in the message. Clients that understand the redirection code ignore this text.

The gateway assumes that identifiers in the URL of the request that do not start with either "C", "T", "M", "A", or "I" refer to *HTML files* which are only used by the gateway. These files contain most of the components of the gateway's "user interface" (see 4.6.1.). When such files are requested, one of the utility functions of the gateway replaces all occurrences of place holders (see 4.6.1.1.) with their current values before sending the contents of the files to the client.

If method **POST** was used in the request to the gateway, the message body has to be the contents of a submitted form. Depending on the identifier in the URL, which has to be one of the gateway commands described in 4.6.2., it is decided which action to take.

4.4.3. Implementation Details

The gateway program stores some **meta-information**, in particular the title, the ID, and the URL of the last retrieved collection or text document in an instance of the class Document_Info. One reason why this information is needed is that it allows retransmission of the last document retrieved. This is used, for example, after a successful user identification operation. The last text or collection transmitted will be retrieved again and users will be able to see the effect of their action at once. A collection list will possibly contain more items in identified mode. After a change of the preferred language has been performed, the last text or collection retrieved, if brought up again, will probably be written in a different language this time.

Another case, where meta-information about the last collection retrieved is needed, is searching. A user of a native Hyper-G client would be able to select a current collection to search in. The gateway program simulates this by specifying that the last collection retrieved is always the current collection, which determines the scope of the search.

Finally, a remark should be made on the **log file** into which the gateway program writes certain information about request from WWW clients. Each request causes one single line to be written in the file whose name may be specified on the command line or in an environment variable (see 4.6.3.). The format of such a line is:

```
DATE/TIME KEY HOSTNAME MESSAGE
```

`DATE/TIME` is the date and time when the log file entry is made.

`KEY` is the session key identifying the session which the request belongs to.

`HOSTNAME` is either the domain name or the internet address (if the domain name could not be found out) of the client's host.

`MESSAGE` is a string that describes the type of the request, the action taken by the gateway, and whether or not processing has been successful. The string also contains, depending on the type of operation, either the title of the document retrieved, the user name (identification), the name of the client program (session start), or the query string (searching).

4.5. The User's View of the Gateway

This section describes the way how information is presented to the user of a WWW client when accessing resources via the gateway. Moreover, it is explained how the user can interact with the WWW gateway (and consequently with the Hyper-G server) and how the gateway allows to use certain features of the Hyper-G system, such as searching, identification, etc.

What is shown in the following can be seen as the *"user interface"* of the gateway, although the actual appearance of information received from the gateway is, of course, under the control of the client program. Despite the fact that the gateway is actually a server program from the user's view, it is responsible for generating the interface to the Hyper-G system. To fulfil this task the gateway uses the possibilities that HTML can provide, such as fill-out forms, inline images, "click-able" icons, and so forth. It is clear, however, that the design of the "user interface" is restricted to features offered by HTML.

It has to be kept in mind that the optical appearance of this interface obviously differs from one client program to another. This is why the following description can only be seen as a general guideline for the user. However, current WWW clients do not differ very much in how they implement the most important features of HTML as far as they can rely on similar hardware capabilities.

A description of the "user interface" cannot be made completely independent from the client program that actually presents the interface to the user. Although the WWW gateway also works together with more text based clients, such as for example Lynx*, the following description sometimes implies a graphical WWW client like Mosaic [ANDREESSEN93] or Netscape+. It is not assumed that this is any restriction, as such programs are in wide use at the moment. This does not mean, though, that the gateway does not work together with other clients as well.

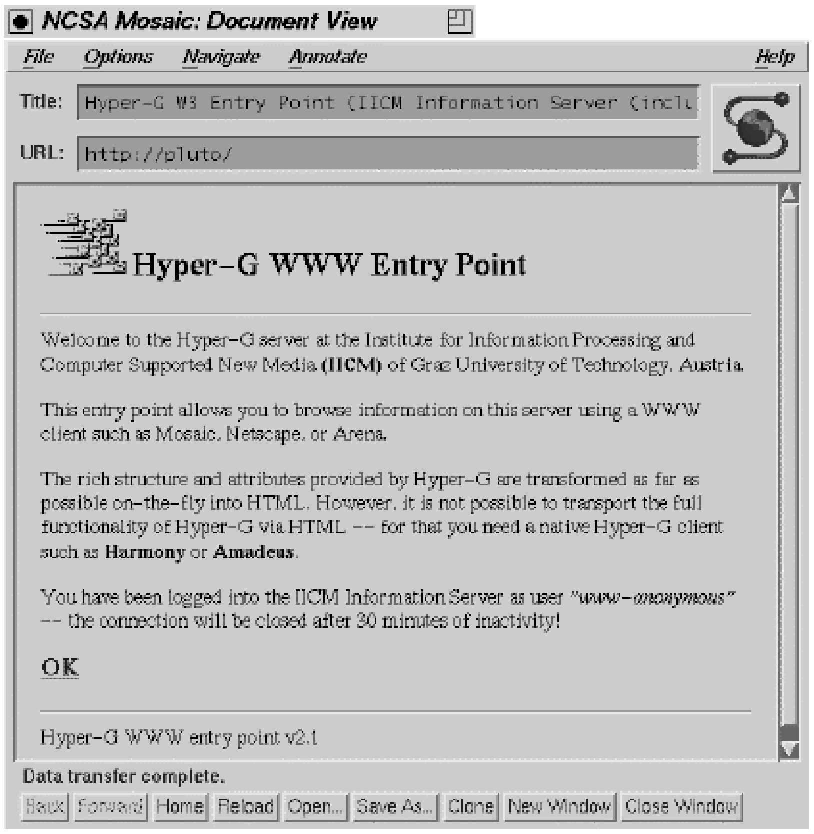
All the following "screenshots" are obtained by sending requests to WWW gateways of particular Hyper-G servers using WWW clients like Mosaic or Netscape. The screenshots represent the "user interface" originally implemented for the WWW gateway, but they can only serve as examples because the files that are the source of these pages may be modified for customization (see 4.6.). Hence, it is possible that these pages look totally different at other Hyper-G sites.

4.5.1. Entry Page

Whether or not an entry page is transmitted as a response to the first request of a particular user to the gateway depends on the options which the gateway has been set up with. If the default setting is used when starting the gateway, an entry page is sent whenever a user requests no particular document. This is the case when the path of the URL is either "/", "/ROOT", "/home", or "/home.html".

* Lynx is a hypertext client developed at The University of Kansas by Montulli L., Grobe M., and Rezac C. Information can be obtained by anonymous ftp at `ftp2.cc.ukansas.edu` in `pub/lynx`

+ Netscape is a WWW browser made by Netscape Communications Cooporations. Most of the information available about Netscape is "hyperlinked" to its home page `http://home.mcom.com/`


NCSA Mosaic: Document View
File Options Navigate Annotate Help
Title: Hyper-G W3 Entry Point (IICM Information Server (inclu
URL: http://pluto/
Data transfer complete.
Back Forward Home Reload Open... Save As... Clone New Window Close Window

Hyper-G WWW Entry Point

Welcome to the Hyper-G server at the Institute for Information Processing and Computer Supported New Media **(IICM)** of Graz University of Technology, Austria.

This entry point allows you to browse information on this server using a WWW client such as Mosaic, Netscape, or Arena.

The rich structure and attributes provided by Hyper-G are transformed as far as possible on-the-fly into HTML. However, it is not possible to transport the full functionality of Hyper-G via HTML -- for that you need a native Hyper-G client such as **Harmony** or **Amadeus**.

You have been logged into the IICM Information Server as user *"www-anonymous"* -- the connection will be closed after 30 minutes of inactivity!

OK

Hyper-G WWW entry point v2.1

Figure 4.8: Entry page (displayed with Mosaic)

As you can see in figure 4.8, the entry page can be used to provide the user with important information concerning the WWW gateway of the particular Hyper-G server whose name is also given on this page.

The link that is followed when clicking on "OK" leads to the document that was originally requested. The URL of this link contains a session key by which the gateway identifies the newly opened Hyper-G session for that particular user. When the client program does not support the HTTP redirect mechanism, an entry page is obligatory because this is the only possibility of telling the client to use the session key in all further requests.

4.5.2. Menu Line

At the top of most pages received from the gateway, a menu line separated by a horizontal ruler from the rest of the page is displayed. This menu line usually contains 4 "click-able" icons and the name of the Hyper-G user for whom the current session is maintained. This can be seen at the top of figure 4.9.

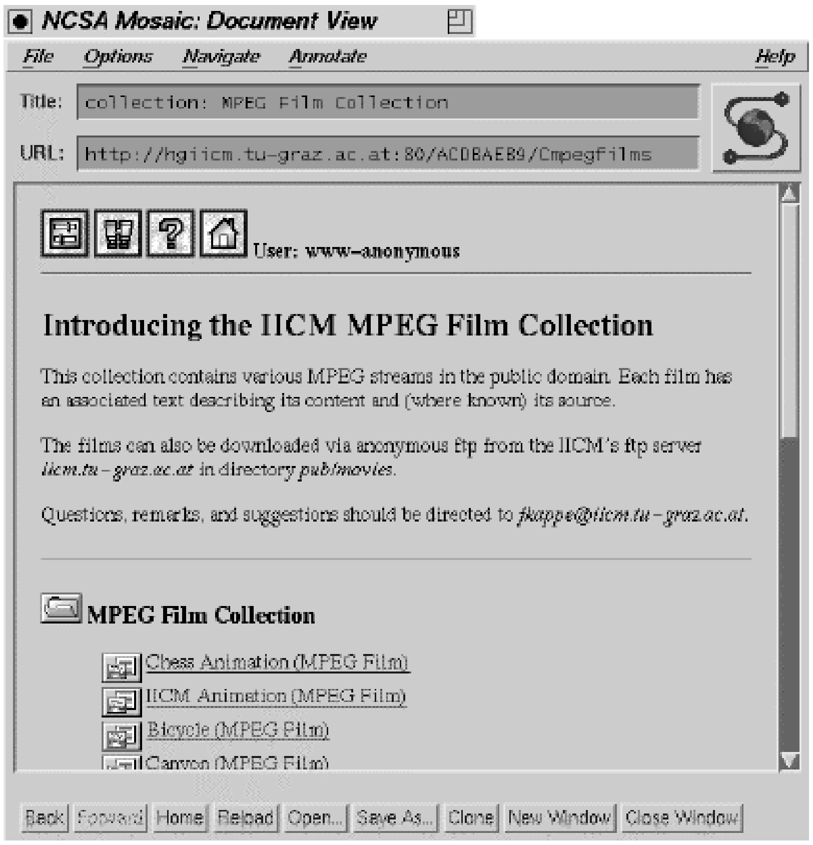
The icons have the following meaning:

"Get Options Menu" (see 4.5.6.)

"Get Search Form" (see 4.5.5.)

"Get Online Help" (see 4.5.7.)

"Go to your Home Collection" This displays either the user's personal home collection or the default home collection for anonymous users, depending on the identification status (see 4.5.6.1.)



collection: MPEG Film Collection

http://hgiicm.tu-graz.ac.at:80/AC0BAEB9/Cmpegfilms

User: www-anonymous

Introducing the IICM MPEG Film Collection

This collection contains various MPEG streams in the public domain. Each film has an associated text describing its content and (where known) its source.

The films can also be downloaded via anonymous ftp from the IICM's ftp server *iicm.tu-graz.ac.at* in directory *pub/movies*.

Questions, remarks, and suggestions should be directed to *fkappe@iicm.tu-graz.ac.at*.

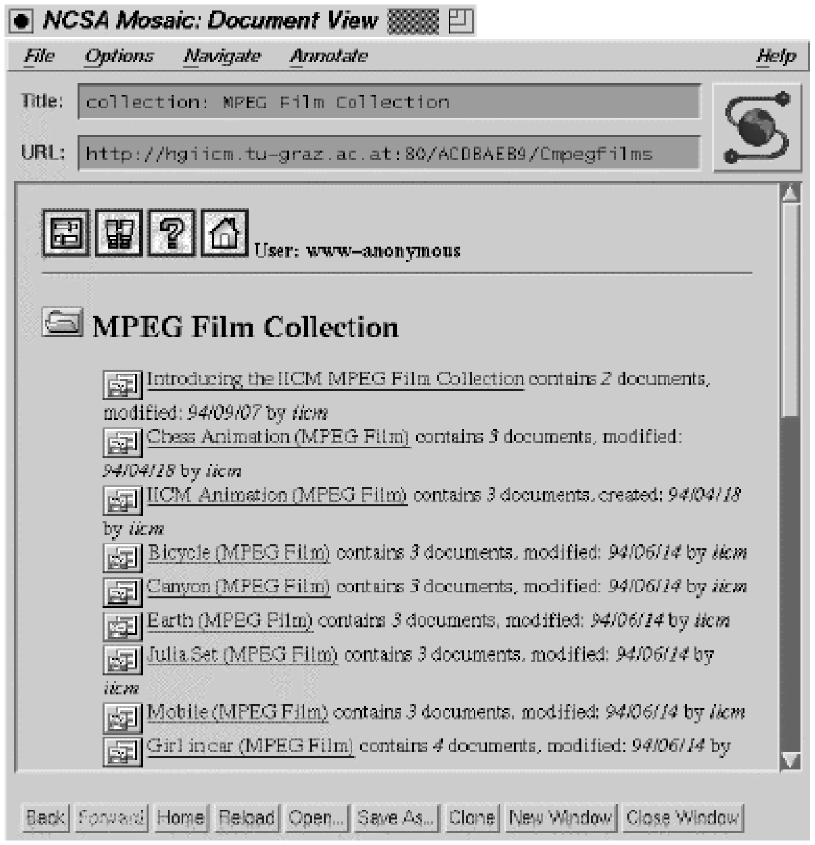
MPEG Film Collection

Chess Animation (MPEG Film)
IICM Animation (MPEG Film)
Bicycle (MPEG Film)
Canyon (MPEG Film)

Figure 4.9: Collection list (displayed with Mosaic)

4.5.3. Collection Lists

The collection list displays members of the current level of Hyper-G collection, together with some optional additional attribute information. Each entry on this list represents one member of the collection and contains a **document type icon** and a link to that particular document (see figures 4.9. and 4.10.).



Title: collection: MPEG Film Collection

URL: http://hgiicm.tu-graz.ac.at:80/ACDBAEB9/CmpegFilms

User: www-anonymous

MPEG Film Collection

Introducing the IICM MPEG Film Collection contains *2* documents, modified: *94/09/07* by *iicm*
Chess Animation (MPEG Film) contains *3* documents, modified: *94/04/18* by *iicm*
IICM Animation (MPEG Film) contains *3* documents, created: *94/04/18* by *iicm*
Bicycle (MPEG Film) contains *3* documents, modified: *94/06/14* by *iicm*
Canyon (MPEG Film) contains *3* documents, modified: *94/06/14* by *iicm*
Earth (MPEG Film) contains *3* documents, modified: *94/06/14* by *iicm*
Julia Set (MPEG Film) contains *3* documents, modified: *94/06/14* by *iicm*
Mobile (MPEG Film) contains *3* documents, modified: *94/06/14* by *iicm*
Girl in car (MPEG Film) contains *4* documents, modified: *94/06/14* by

Figure 4.10: Collection list (displayed with Mosaic)

The amount of **attribute information** (author, date, size, etc.) displayed for each member can be changed by modifying the display setting in the options menu (see 4.5.6.). The default is to display no attributes.

The user can also toggle the display of the **"collection head"**. When on, the collection head document (typically a description of the collection) is displayed before the actual collection list. When off, the collection head document appears in the collection list like any other document. In figures 4.9. and 4.10. the same collection list is shown. The difference is that in figure 4.9. the collection head is displayed, but no attribute information is given. In contrast, figure 4.10. does not show the collection head, but attributes are displayed with every member of the collection.

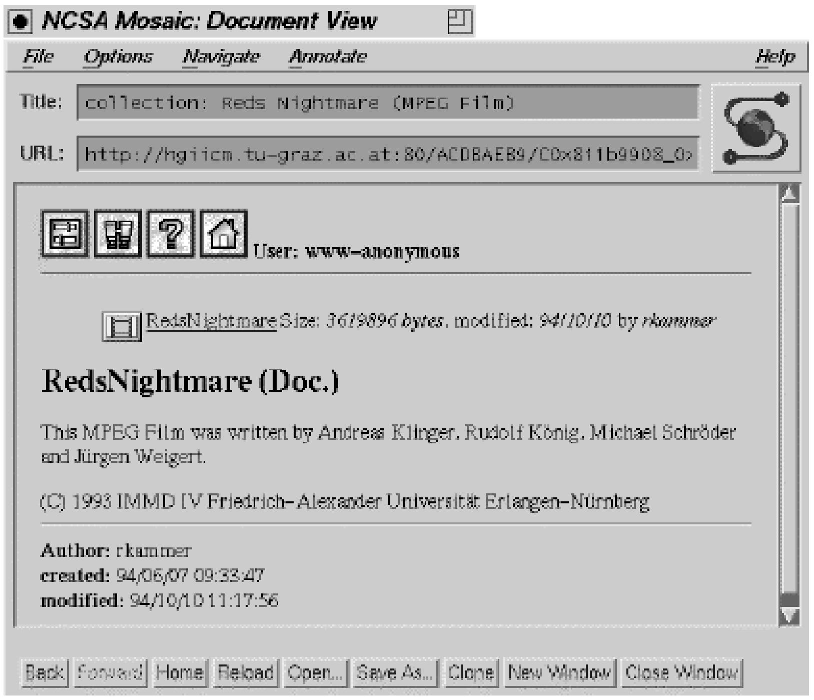
It has to be mentioned that the number of members displayed can differ from the number of objects actually in the collection. This depends on the user's identification status (see 4.5.6.1.).

4.5.4. Documents

The way how documents are presented to the user depends, of course, on their type.

Text documents, collection lists, and clusters have a menu line (see 4.5.2.) displayed at the top and some attributes such as author, creation date, and modification date displayed on the bottom, both separated by a horizontal ruler from the actual document (see figure 4.11.)

As **clusters** usually contain more than one document, they are treated in a special way. All documents contained in a cluster that are no text documents are listed with their title and type icon as in a collection list (actually a cluster is a special collection). This can be seen in the middle of figure 4.11. They can be visualized (or heard, in case of an audio document) when the user follows the corresponding links. Among text documents, however, the gateway tries to find one which is written in the preferred language of the user (see 4.5.6.3.). Only this one is displayed right after the list of links to documents of other types. If there is no such text, any other text document contained in the cluster will be used. All other text documents are not visualized.



Title: collection: Reds Nightmare (MPEG Film)

URL: http://hgiicm.tu-graz.ac.at:80/AC0BAEB9/C0x811b9908_0>

User: www-anonymous

RedsNightmare Size: *3619896 bytes*, modified: *94/10/10* by *rkammer*

RedsNightmare (Doc.)

This MPEG Film was written by Andreas Klinger, Rudolf König, Michael Schröder and Jürgen Weigert.

(C) 1993 IMMD IV Friedrich-Alexander Universität Erlangen-Nürnberg

Author: rkammer
created: 94/06/07 09:33:47
modified: 94/10/10 11:17:56

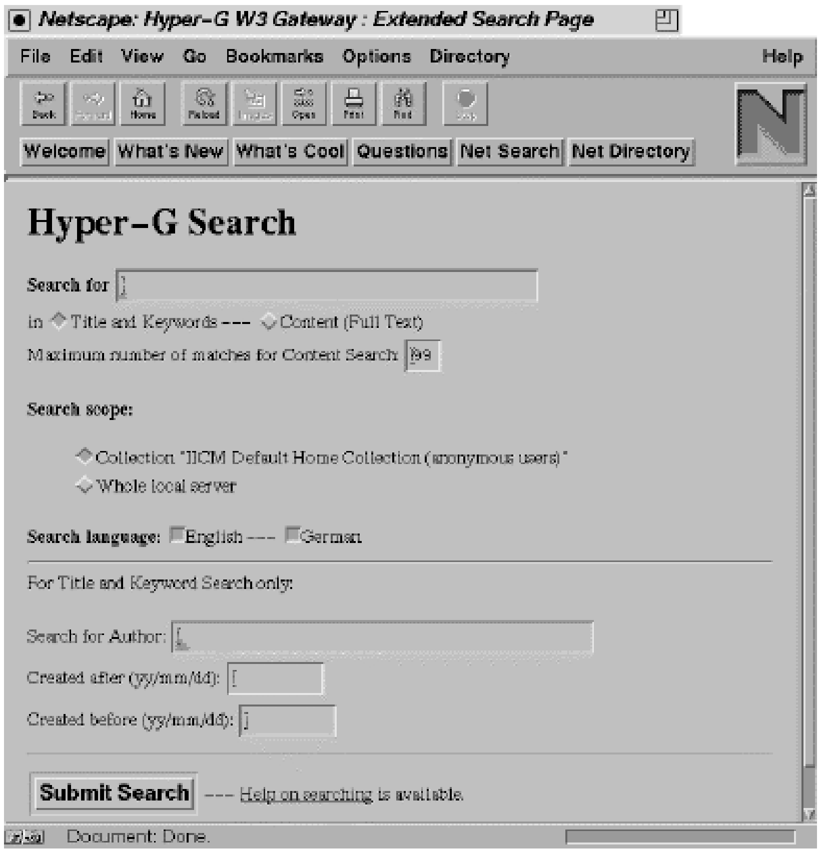
Figure 4.11: Cluster (displayed with Mosaic)

Documents of other types such as images, sounds, films, 3D scenes, postscript documents, compressed postscript documents, HM-Card documents, and other documents of "generic" types are send to the WWW client preceded by a MIME header indicating the content type. Depending on the capability for handling these documents, the client will either deal with them itself, or start another application to do the job.

4.5.5. Searching

The gateway uses HTML fill-out forms to provide a user interface for searches in Hyper-G databases. Both a *simple search form* and an *extended search form* (figure 4.12.) are available. The simple form can be accessed from the icon in the menu line. A link to the extended search form is contained in the simple form. Both search forms allow the user to select between two types of searching:

- **Title and Keyword Search**
- **Content Search (Full Text)**


Netscape: Hyper-G W3 Gateway : Extended Search Page
File Edit View Go Bookmarks Options Directory Help
Back Forward Home Reload Images Open Print Find Stop
Welcome What's New What's Cool Questions Net Search Net Directory

Hyper-G Search

Search for

in ◇ Title and Keywords --- ◇ Content (Full Text)

Maximum number of matches for Content Search: 99

Search scope:

◇ Collection "IICM Default Home Collection (anonymous users)"
◇ Whole local server

Search language: ☐ English --- ☐ German

For Title and Keyword Search only:

Search for Author:

Created after (yy/mm/dd):

Created before (yy/mm/dd):

Submit Search --- Help on searching is available.

Document: Done.

Figure 4.12: Extended search form (displayed with Netscape)

Both forms allow the user to specify a search query which is then submitted to the Hyper-G server. The search query consists of words separated by operators.

The OR operator may be specified as: `|`, `or`, `oder` (case-insensitive).
The AND operator can be: `and`, `und` (case-insensitive).
A space is interpreted as an AND operator for Title and Keyword Search, and as an OR operator for Content Search.

Search terms may be truncated with an asterisk and brackets may be used to form the query.

Examples:

```
hyper*
(IICM) | (IHM)
```

Operators like `or`, `and`, etc. always have to be separated from other words by spaces even when brackets are used (e.g. "`(IICM) or (IHM)`" and not "`(IICM)or(IHM)`").

The **scope** of the search can be the whole server or may be focused on a particular collection (including its sub-collections), which does not have to reside on the same Hyper-G server. The collection to search in is always the last retrieved collection. Its name is displayed in the search form.

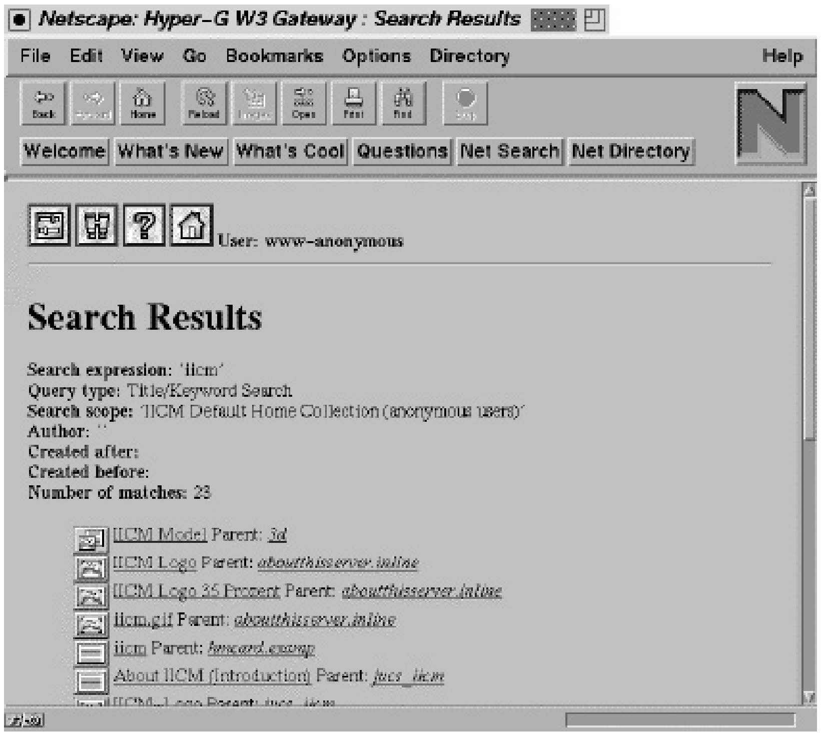
The extended search form provides the following additional features:

- The *number of matching objects* returned by Content Search can be limited (the result list is ranked by score).
- Additional *attributes* such as 'author' or 'creation date' can be specified for Title and Keyword Search.
- The *language(s)* within which to search can be specified (either English, German or both).

Result List

The search result list displays the same attributes as for members of the collection list (see 4.5.3.). In addition, a link to one of the parent collections of each object is displayed.

Moreover, in case of Content Search, the *score* of each document is displayed.



User: www-anonymous

Search Results

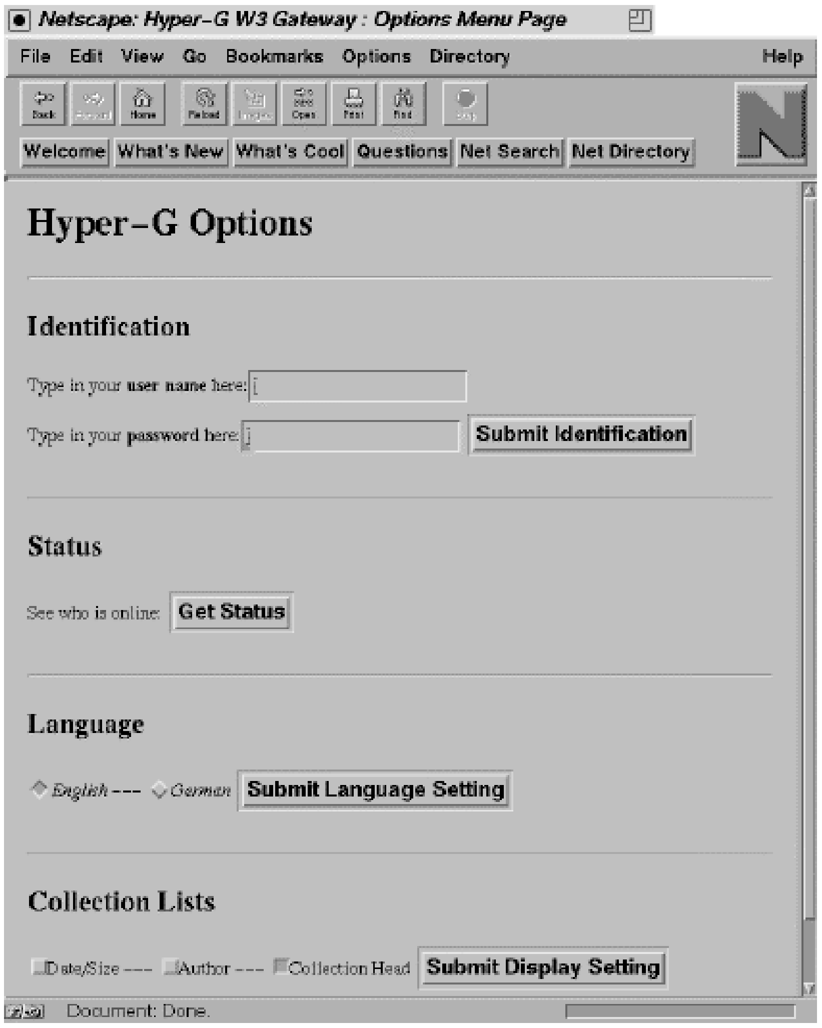
Search expression: 'iicm'
Query type: Title/Keyword Search
Search scope: 'IICM Default Home Collection (anonymous users)'
Author: ''
Created after:
Created before:
Number of matches: 23

IICM Model Parent: *3d*
IICM Logo Parent: *aboutthisserver.inline*
IICM Logo 35 Prozent Parent: *aboutthisserver.inline*
iicm.gif Parent: *aboutthisserver.inline*
iicm Parent: *lmcard.examp*
About IICM (Introduction) Parent: *jucs_iicm*

Figure 4.13: Result of a Title/Keyword Search (displayed with Netscape)

4.5.6. Options Menu

This page containing four forms separated by horizontal rulers (see figure 4.14.) can be accessed through the options icon in the menu line (see 4.5.2.).


Netscape: Hyper–G W3 Gateway : Options Menu Page
File Edit View Go Bookmarks Options Directory Help
Back Forward Home Reload Images Open Print Find Stop
Welcome What's New What's Cool Questions Net Search Net Directory
Submit Identification
Get Status
Submit Language Setting
Submit Display Setting
Document: Done.

Hyper–G Options

Identification

Type in your **user name** here:

Type in your **password** here:

Status

See who is online:

Language

English --- *German*

Collection Lists

Date/Size --- Author --- Collection Head

Figure 4.14: Options (displayed with Netscape)

4.5.6.1. User Identification

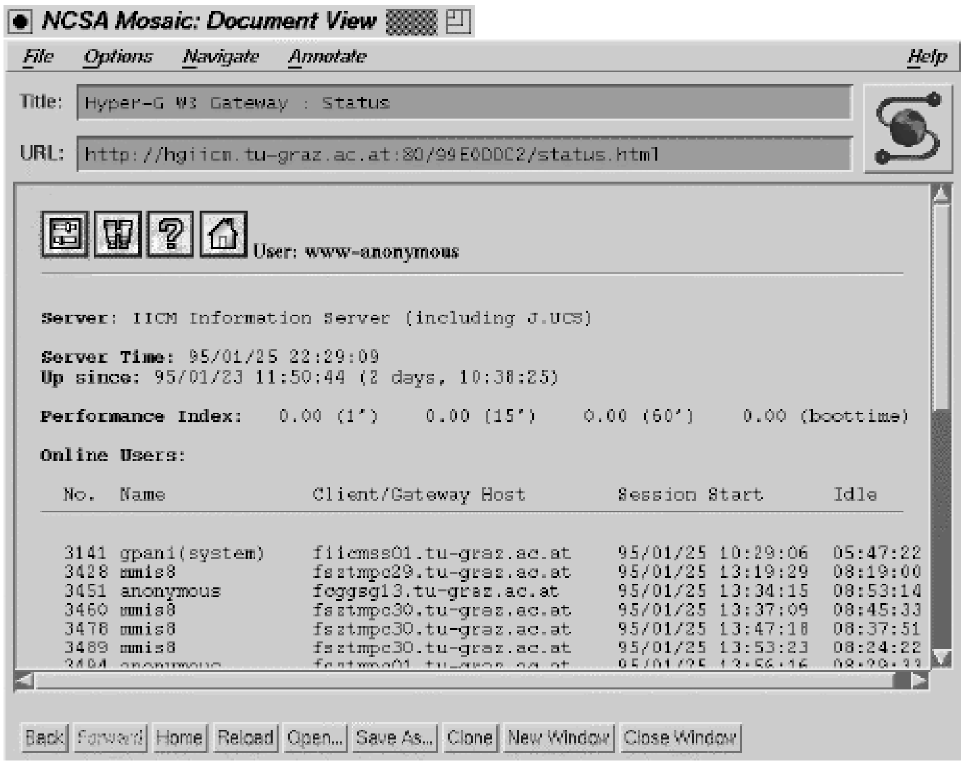
Users may identify themselves to the Hyper-G system by typing in their *name* and *password* in the identification form, which can be found in the options menu.

Assuming that a user has an account on the Hyper-G server the gateway is connected to, he/she is then logged in as an identified Hyper-G user, which can be verified by looking at the 'who-is-online-list' (see 4.5.6.2.).

Identification extends the user's access rights. For example, searches often result in more matches and some of the collection lists have possibly more entries if the user is identified.

4.5.6.2. Status

By clicking on the "Get Status" button, a page containing statistics of the Hyper-G server to which the gateway is connected followed by a "who-is-online list" appears (figure 4.15).


NCSA Mosaic: Document View
File
Options
Navigate
Annotate
Help
Title: Hyper-G W3 Gateway : Status
URL: http://hgiicm.tu-graz.ac.at:80/99E000C2/status.html
Back
Forward
Home
Reload
Open...
Save As...
Clone
New Window
Close Window

```
User: www-anonymous

Server: IICM Information Server (including J.UCS)

Server Time: 95/01/25 22:29:09
Up since: 95/01/23 11:50:44 (2 days, 10:38:25)

Performance Index:   0.00 (1')    0.00 (15')    0.00 (60')    0.00 (boottime)

Online Users:

  No.  Name             Client/Gateway Host         Session Start        Idle

  3141 gpani(system)    fiicmss01.tu-graz.ac.at     95/01/25 10:29:06   05:47:22
  3428 mmis8            fsztmpc29.tu-graz.ac.at     95/01/25 13:19:29   08:19:00
  3451 anonymous        feggsg13.tu-graz.ac.at      95/01/25 13:34:15   08:53:14
  3460 mmis8            fsztmpc30.tu-graz.ac.at     95/01/25 13:37:09   08:45:33
  3478 mmis8            fsztmpc30.tu-graz.ac.at     95/01/25 13:47:18   08:37:51
  3489 mmis8            fsztmpc30.tu-graz.ac.at     95/01/25 13:53:23   08:24:22
```

Figure 4.15: Status page (displayed with Mosaic)

This list contains all currently active Hyper-G sessions for this particular server. Each line starts with a session number, followed by the user name, the name of the client host or the gateway host (as the WWW gateway is a Hyper-G client from the server's point of view), session start date/time, and idle time. Users with name "www-anonymous" in figure 4.15 are anonymous users connected to the Hyper-G server via the WWW gateway.

4.5.6.3. Preferred Language

To support the multilingual concept of Hyper-G, the gateway provides a form within the options menu (figure 4.14.) which allows users to change their preferred language. Radio buttons can be either set to *English* or *German* as the preferred language.

This has two effects:

- The Hyper-G server will try to fetch documents in the preferred language, if available in the database.
- The gateway presents its user interface (forms, menus, search results, messages, online help etc.) in the preferred language.

An important remark has to be made at this point. Changing the preferred language does not, of course, affect documents already loaded. Accessing such a document (or collection list) through the "BACK" feature of the client typically does not reload the document.

4.5.6.4. Display Setting

The amount of attribute information displayed for each member of a collection list (see 4.5.3.) can be changed.

Possible attributes are:

for *collections:*	number of sub-documents, author;
for *clusters:*	number of sub-documents, modification date (if present, otherwise creation date), author;
for *all other documents:*	size, modification date (if present, otherwise creation date), author (not for remote documents).

The options menu (figure 4.14.) contains a form with three toggle buttons. With the first one the displaying of the *author's name* can be switched on or off. The second one controls the displaying of *creation/modification date* and *size/number of sub-documents.*

The display of the *"collection head"* can also be toggled. When on, the collection head document, if there is one present in the collection, will be displayed before the actual collection list. When off, the collection head document will appear in the collection list like any other member of the collection.

4.5.7. Online Help

Help on using the WWW gateway to Hyper-G can be accessed through the help icon of the menu line (see 4.5.2.). The help page displayed contains links to further pages which cover the topics: identification, searching, collection lists, and language setting.

All online help pages are available in English and in German. Which version is displayed depends on the preferred language set by the user in the options menu (see 4.5.6.3.).

4.6. Customization

4.6.1. Use of Files for Building the User Interface

The gateway provides the user of a WWW client with an interface to the Hyper-G system. It is clear that this user interface has to consist of elements provided by HTML in order to be sent to WWW clients. The decision to put most of the elements which make up the user interface into a number of HTML files rather than to "hard-code" the whole functionality has some remarkable advantages.

- Large parts of the user interface can be modified by simply editing text files. This reduces the number of changes that have to be made in the gateway program significantly.
- Customizations can be made for example by system administrators in order to adapt the user interface of the gateway to individual needs of different sites running a Hyper-G server together with a WWW gateway.
- Support of Hyper-G's multi-lingual concept can be achieved easily for the user interface by providing a separate set of files for each language. If a new language has to be added, only very small changes will have to be made in the gateway program.

The idea is that whenever the gateway needs a certain part of the user interface (e.g. a form or a menu line with click-able icons), it should read the corresponding file and send it to the client.

However, the problem that arises is that some pieces of information can only be provided by the gateway program at run-time and therefore cannot already be contained in the file (for instance, the name of the current collection). This is why a set of **place holders** to be used within the files have been specified. These place holders are recognized by the gateway program when it reads the file. The gateway then replaces the place holders with their current values and sends them to the client.

4.6.1.1. Specification of Place Holders

The following place holders, if contained in HTML files of the gateway, are replaced with their current values before the contents of those files are sent to the client.

All these place holders consist of a string with two percentage signs at the beginning and at the end.

`%%include:%%`	Include file If `%%include:` is found in the file, it will be replaced with the content of an includefile. The name of the includefile must be given just after `%%include:` on the same line without quotes (e.g. `%%include:includefilename.html%%`) If the includefile cannot be found, the gateway program will proceed with the "parent"-file.
`%%date%%`	Local date of gateway
`%%time%%`	Local time of gateway
`%%my_host%%`	Host name of gateway
`%%hgserver%%`	Name of Hyper-G Server (Server String)
`%%user%%`	User Name
`%%key%%`	Session Key
`%%port%%`	WWW Port

`%%url%%`	URL of client's request to the gateway without user key and reduced (only path part) if local URL. A leading '/' is removed from reduced URLs.
`%%url_with_key%%`	URL of client's request to the gateway with user key inserted (can be full url or path part starting with '/')
`%%obj_type%%`	Type of object
`%%obj_url%%`	URL of object
`%%obj_title%%`	Title of object
`%%obj_author%%`	Author of object
`%%obj_rights%%`	Access rights for object
`%%obj_parent%%`	Parent of object
`%%obj_subdocs%%`	Number of sub-documents of object (collection)
`%%obj_created%%`	Creation time and date of object
`%%obj_modified%%`	Modification time and date of object
`%%obj_expires%%`	Expiration time and date of object
`%%obj_descript%%`	Description of object
`%%last_coll_url%%`	URL of last retrieved collection
`%%last_coll_title%%`	Title of last retrieved collection
`%%x0%% .. %%x10%%`	Sub-string of status-string This place holder is used only in the file `status.html` (see 4.6.1.2.). The gateway replaces each occurrence with a certain value belonging to the user statistics. The order (x0 to x10) is as follows: server string; local time of server; "up since", days, hours, minutes, seconds; number of hits in the last minute, quarter of an hour, hour; total number of hits since up.
`%%who%%`	Who-is-online list

`%%s0%% .. %%s6%%`	Sub-string of string-argument value This is used in various files. The meaning differs from one file to another (see 4.6.1.2.). In `ident_ok.html %%s0%%` stands for the old user name. In `ident_fail.html %%s0%%` stands for the user name typed in. In `result_head.html` the order (s0 to s6) is as follows: query string, collection title, search type, author, "date after", "date before", number of matches.
`%%disp_date%%`	Collection list: display date/size This will be replaced with CHECKED, if the display of date and size in collection lists is switched on.
`%%disp_author%%`	Collection list: display author This will be replaced with CHECKED, if the display of author in collection lists is switched on.
`%%disp_collhead%%`	Collection list: display collection head This will be replaced with CHECKED, if the display of collection heads in collection lists is switched on.

A few comments may be made on some of the place holders.

The place holders `%%disp_date%%, %%disp_author%%,` and `%%disp_collhead%%` allow the gateway to specify the current settings for some checkboxes contained in a form.

All place holders starting with `%%obj_` refer to the current document, either texts or collections. There is a special feature for the this category of place holders. If their corresponding values which should replace them are empty, the whole line from the file will be skipped (not sent to the client). This allows dynamic displaying of certain values together with their descriptive texts from the HTML-file. If a value is empty, the descriptive text will not be displayed either (This is used, for example, in the file `footer.html` - see 4.6.1.2.). It has to be mentioned that the first %%obj_...%% place holder of a line which would be replaced with an empty value will cause a skipping of the rest of the line. Therefore, not more than one %%obj_...%% place holder should usually be put on the same line.

However, it is possible to force the line to be transmitted in any case (no line skipping), if a "!" is given before the first character of the place holder's name (e.g.: %%!obj_title%%).

4.6.1.2. Description of Files

All files used by the gateway except for the SGML parser's style sheet and the DTD are kept in a directory called HTML directory and its sub-directories.

The parser looks in a directory called sgml for the files

- `htf_2_html.sty` and
- `htf_2_html.dtd`.

Conversion of HTF documents into HTML documents is controlled by the contents of these two files.

The actual names and paths of the HTML and the sgml directories can be specified as command line parameters, the name of the HTML directory can also be given in an environment variable (see 4.6.3.).

There are two categories of files in the HTML directory. The first one are files that do not have any language dependent contents. These are mainly image files representing icons used by the gateway (menu icons, document type icons, logo). They are kept directly in the HTML directory.

The second category are HTML files containing various parts of the user interface of the gateway, especially fill-out forms. There exists a separate set of these files for each supported language. They reside in sub-directories of the HTML directory. The names of the sub-directories correspond to the language codes. Currently there are an English and a German version residing in the sub-directories "en" and "ge" respectively.

When attempting to open one of these files for reading, the gateway follows a certain strategy. At first it looks for the file in the sub-directory that corresponds to the preferred language. If the file cannot be found there, the gateway looks in the sub-directory "en". If the file is not present there either, a last attempt is made in the HTML directory. Failing all this, the gateway sends some information instead that is "hard-coded" into the program, whenever possible. For instance, if the file `footer.html` cannot be found, the gateway will at least send a </BODY> tag.

This is one of the measures that allow the gateway to offer as much functionality as possible even if it has not been set up properly.

What follows is a list of files currently used by the gateway. Moreover, short descriptions either of particular files or at least of groups of files are given. The names of files that are underlined must not be changed because these file names are directly known by the gateway program. All other file names appear only in anchors within other files or are "included" in other files using the `%%include:` place holder (see 4.6.1.1.). Therefore, the names may be changed if they are also changed within the "calling" files.

There is a couple of files containing image data. The first four of them are menu icons. The rest are document type icons appearing in collection lists.

`HELP8.GIF`
`HOME8.GIF`
`OPTS8.GIF`
`SEAR8.GIF`
`audio.gif`
`cluster.gif`
`coll_clos.gif`
`coll_open.gif`
`film.gif`
`ftp.xbm`
`gopher_index.xbm`
`hglogo1.xbm`
`image.gif`
`telnet.gif`
`text.gif`

There is a separate set of the following HTML files for each language supported. Of course, the file names have to be the same for each set.

`entry.html` This is the entry page which can be sent as response to the first request of a client (see 4.5.1.). Whether it is sent or not, depends on the parameters the gateway has been started with (see 4.6.3.). The text contained can be edited and different logos can be included as inline images in order to generate individual entry pages for different sites running a WWW gateway

and a Hyper-G server. Important is that it contains one anchor whose URL has to be generated dynamically by the gateway using place holders (see 4.6.1.1.). This URL should be the same as in the first request except that this time a new session key is included in the path part. This can be achieved, for example, if the URL contained in the anchor is:
`%%url_with_key%%`
The entry page can also be used to inform the user of important changes. Especially, if a URL is going to be closed in the near future, a message telling the user about the new location can be included in the entry page. Finally, when the URL is not valid any more, it is recommended to set up the gateway with command line option "`-e always`" and perhaps "`-t 1`". This forces the gateway to respond with the entry page independently of the request it has received. The anchor should be removed from the page. Hence, users have no chance to proceed sending requests using this URL, but they can be informed through the entry page about a new URL to use.

`footer.html` This is sent at the end of each document and collection list. It usually contains some place holders that are dynamically replaced by some attribute information (e.g. author, creation date, etc.) about the document or collection list.

`header.html` This is sent at the beginning of each document and collection list. It contains the HEAD section of an HTML document (compare 2.5.3.1.). The `<BASE>` tag gives the actual URL of the document using the `%%obj_url%%` place holder. This is often necessary if the gateway answers to a request with a document whose URL is different from the URL of the request. This happens, for instance, when submitting forms with the POST operation.
The file usually "includes" the file `menuline.html` using the `%%include:` place holder in order to have the menu line on top of each document.

`help.html` This file is sent when users click on the help icon in the menu line. It represents the main online help page with links to the following four help pages. It can also contain links to other help information available from the Hyper-G server.

`help_coll_lists.html`	Help on collection lists.
`help_identification.html`	Help on user identification.
`help_language.html`	Help on language setting.
`help_search.html`	Help on searching.
`ident_fail.html`	This file contains a message telling the user that an attempt to identify himself/herself to the Hyper-G server has failed. It also provides the user with a form in which name and password can be typed in order to retry identification.
`ident_ok.html`	This represents a message to the user telling that his/her attempt for identification has been successful. The user also gets the possibility to follow a link to the previously accessed document or collection or to go to the user's personal home collection.
`menuline.html`	This file contains `<IMG>` tags to the inline menu icons as well as the `%%user%%` place holder to form the menu line, which is often "included" in other files.
`nothing_found.html`	A message is contained in this file telling users that their previous query did not result in any matches. The query is shown again for verification.
`options.html`	This represents the options menu which contains four fill-out forms (see 4.5.6.).
`result_head.html`	This file is sent at the beginning of a search result. It shows the query which the user specified and informs about the number of matches.
`search.html`	The extended search form is contained in this file (see 4.5.5.).
`search_simple.html`	This represents the simple search form (see 4.5.5.)
`status.html`	This file contains a lot of place holders with special meanings. The gateway program replaces them with certain values from the server statistics. Moreover, a current "who-is-online" list is generated. Whenever a user requests this file, the gateway answers with up-to-date server statistics and a new who-is-online list.

To sum up, it can be said that all of the files may be modified to a certain extend. This may, of course, sometimes reduce the functionality of the gateway (especially if anchors are removed). Care has to be taken when some of the `%%s0%% .. %%s6%% and %%x0%% .. %%x10%%` place holders as well as some of the anchors containing place holders are being modified or removed, as this could cause unexpected effects.

On the other hand, this open concept allows an easy adaptation of the user interface. For instance, if somebody wants an entry field to specify a search query on the bottom of every document or collection list, this can easily be achieved by putting a fill-out form into the file `footer.html`.

Another possibility for a modification would be to put an inline image anchor into the file `menuline.html`. This anchor would point, for example, to some kind of logo.

4.6.2. Gateway Commands

The WWW gateway understands the five "commands": *`identify, status, searchit, language,`* and *`display.`* These commands are used as identifiers in the path part of URLs when clients submit fill-out forms generated by the gateway. In other words, the ACTION attribute of a form is the gateway's URL with an identifier that is one of the commands previously listed (see 4.2.5. for an explanation on identifiers in URLs to be used with the gateway). All these fill-out forms have to be submitted to the gateway <u>with method POST</u>.

Whenever a request is sent to the gateway with method POST, the gateway checks whether or not the URL contains one of the five commands. If it does, the data body of the request is interpreted as a string containing name/value pairs of the form: `name1=value1&name2=value2&name3=value3...` (compare 2.5.4.)

The contents of this string depends on the individual command. For each command a different set of name/value pairs is defined. What follows is a description of all gateway commands together with the meaning of their name/value pairs.

<u>`identify`</u>

A request with this command is used for identification of a user to the Hyper-G system (compare 4.5.6.1.). The message body of the POST operation has to contain the two fields (name/value pairs) `username` and `password`.

Example: `username=`*`uname`*
`password=`*`secretpassw`*

In this example, *`uname`* and *`secretpassw`* are the user's Hyper-G user name and password. If they correspond to a valid account on the Hyper-G server, the user will be logged in as an identified user with probably extended access rights and the gateway responds by sending the file `ident_ok.html`. If the identification fails, the user will stay anonymous and the gateway will respond with the contents of the file `ident_fail.html` (see 4.6.1.2.).

status

If a request has a `status` identifier, the gateway sends a page containing statistics of the Hyper-G server which the client talks to via the gateway followed by a "who-is-online list" (see 4.5.6.2.). The same happens when the file `status.html` (see 4.6.1.2.) is requested with method GET. The reason why there is also a command `status` is that it can be used in a form which contains a "Get Status" button.
There is no name/value pair defined for this command. Therefore, the message body of such a request is empty (i.e. content length = 0).

searchit

This command is used whenever search queries are transmitted to the gateway (compare 4.5.5.). The data body contains the fields of the query. A number of name/value pairs is defined for this purpose.

`searchexpr=`*`query`*

`query` is the term to search for.

`type=title/key` or `type=content`

This specifies the type of search. The default value is `title/key.`

`maxftdocs=`*`no_docs`*

The number of documents returned by fulltext search can be limited to *`no_docs`*`.` The result list is ranked by score. The default is no limitation, but a maximum of 99 seems to be appropriate.

`scope=`*`collection`* or `scope=all`

`collection` is the name of the collection to search in. If the value is `all` or

the field is omitted, the search will be performed in the rootcollection of the Hyper-G server.

`language=en` and/or `language=ge`

More than one of these fields are possible. Searching is only performed among documents written in one of these languages. The values have to be the language codes. The default is both English and German texts.

`authorexpr=`*`author`*

This specifies if the search should be directed to documents of one author, whose user name is given in *`author`*. The default are all authors.

`dateafter=`*`adate`*

If *`adate`* is specified, the Hyper-G server will only search for documents that were created after *`adate`*. The date format is `yy/mm/dd.`

`datebefore=bdate`

If *`bdate`* is specified, the Hyper-G server will only search for documents that were created before *`bdate`*. The date format is the same as with *`adate.`*

After receiving the result of the search from the Hyper-G server, the gateway responds either with a result list preceded by the contents of the file `result_head.html` or with the file `nothing_found.html`.

`language`

This command is used to change the preferred language for the individual session (see 4.5.6.3.). As only English and German is currently supported by the gateway, one of the following two name/value pairs may be present in the data body.

```
language=en
language=ge
```

The default language is English. The language setting has an effect on both the user interface language and the language of text documents.

`display`

The appearance of collection lists can be modified by sending a request with the display command. The data body may contain the following fields.

`date/size=yes`

If this name/value pair is present, additional attribute information is displayed for each member in a collection list. For members that are collections or clusters the number of sub-documents is displayed. For all other documents the size in bytes is displayed.

`author=yes`

This field, if present, switches on the display of the author of each member in a collection list except for remote documents.

`collhead=yes`

If this field is present in the data body, the collection head document, usually a description of the collection, is displayed before the actual collection list.

If one of the previous name/value pairs is not present in the data body, the corresponding display is switched off.

In concluding, it has to be mentioned that it is, of course, possible to create new fill-out form layouts other than those provided with the gateway (e.g. 4.5.5. - search form or 4.5.6. - options). However, all forms have to use at least a subset of the commands and fields previously described in order to be processed by the gateway.

4.6.3. Commandlineparameters and Environmentvariables

When a WWW gateway is started, it has to be provided with some information about its environment (e.g. the Hyper-G server to connect to, the WWW port number, etc.). A set of command line parameters has been defined for this purpose. Moreover, they allow to control certain functions of the gateway. Alternatively, for most of these settings UNIX environment variables may be used. However, if parameters are given on the command line, they override the values of environment variables. If neither command line parameters are given, nor environment variables are set, certain program defaults are active.

In the following, all command line parameters are described together with their corresponding program defaults. If environment variables are defined for the same purpose, they are also given in this list. All environment variables defined for the gateway start with "`WWW_`". The gateway also uses the environment variable "`HYPERG_HOME`", which is used by other Hyper-G tools as well.

`-h`

This displays a short help text, lists default values of parameters and then the program exits. If environment variables are set, their values will be used instead of program defaults.

`-v`

This switch turns on extensive debugging output.

`-r` ***`hghost`***

`hghost` is the host name of the Hyper-G server which the gateway has to connect to. The default is `hyperg`.

`-p` ***`hgport`***

`hgport` specifies the number of the port on which the Hyper-G server is listening for connections from Hyper-G clients (and therefore for connections from the gateway). The default is `418`.

`-m` ***`myname`***

`myname` is the name of the gateway host. The default is the name of the host machine.

`-t` ***`timeout`*** **`WWW_TIMEOUT=`*****`timeout`***

If a gateway child process stays inactive for *`timeout`* minutes, it exits closing the connection to the Hyper-G server and therefore, its session ends. The default is 30 minutes.

`-d` ***`htmldir`*** **`WWW_HTMLHOME=`*****`htmldir`***

The name of the directory where the gateway looks for its files can be specified in *`htmldir`*. The language dependent files have to be stored in sub-directories of *`htmldir`* (see 4.6.1.2.). The default is `$HOME/HTML`.

`-l` ***`logfile`***

`logfile` is the name of a file which all log information is written to. The default file name is `/dev/null`, which means that no log information is written in this case.

`-S` ***`SGMLdir`***

The name of the directory where the SGML parser can find the HTF-HTML style sheet and the DTD can be specified in *`SGMLdir`*. If parameter `-S` is omitted, the gateway assumes that the name of the directory is `$HYPERG_HOME/sgml`. If the environment variable `HYPERG_HOME` is not set either, the program default is `/usr/local/Hyper-G/sgml/`.

`-n` **`WWW_COLLHEAD=off`**

This sets the default setting for displaying the collection head document to "off" for a new user. In this case, the collection head appears in the collection list like any other document. The user can change this setting in the options menu. If parameter `-n` is omitted or `WWW_COLLHEAD=on` is given, the collection head document is displayed before the actual collection list, which is also the program default.

`-s` **`WWW_DATE_SIZE=on`**

This turns the default setting for displaying the date and the size of objects in collection lists to "on". The user can change this setting in the options menu. The default is "off".

`-a` **`WWW_AUTHOR=on`**

This sets the default setting for displaying the author of objects in collection lists to "on". It can be changed by users in the options menu. By default, the author's name is not displayed.

`-c` ***`language`*** **`WWW_COUNTRY=`*****`language`***

The preferred language at the start of a session may be given here. *`language`* can be `en` for English, `ge` for German or `detect` for country detection. When accessing the gateway from Austria or Germany the default language is automatically set to German in this case. If parameter and environment variable are omitted, the default language at the beginning of a session is English.

`-e` ***`entry`*** **`WWW_ENTRY=`*****`entry`***

This specifies whether or not an entry page is sent as a response to the first request of a new user. *`entry`* can be `always`, `never` or `home`. `always` causes the gateway to send an entry page at every new session. The redirection mechanism is not used in this case. `never` causes the use of HTTP status code 302 for redirection. If `home` is given, the gateway sends an entry page if the path part of the URL in the first request is either '`/`', '`home`', '`home.html`' or '`ROOT`' and uses redirection else. The program default is the same as if parameter `-e home` was given.

`wwwport` **`WWW_PORT=`*`wwwport`***

The port on which the gateway listens for HTTP requests can be specified here. If the command line parameter is used, it has to be the last parameter on the line. The default is the WWW standard port number 80. Since this is a system port number, either the superuser has to start the gateway, or the Hyper-G tool hgbindport is used.

`WWW_ERRORS=`*`filename`*

If this environment variable is set, all error output generated by the gateway will be written to the file whose name is *`filename`*. There is no command line parameter defined for this purpose. If this environment variable is not set, error output is written on standard error.

4.7. Proxy Gateway Support

4.7.1. Proxy Server Concept

A proxy server is a special type of server or gateway that usually runs on a *firewall* machine. A proxy server, proxy for short, provides access to resources on the Internet for people on closed subnets who can only access the outside world through a single machine for security reasons. Requests from clients inside the firewall are forwarded by the proxy to the remote server outside the firewall. After reading the response from the remote server the proxy sends it back to the client.

As the same proxy is usually used by all the clients within a given subnet, it is possible for the proxy server to do document **caching**. To let the proxy do the caching is much more efficient than the caching mechanisms built in the individual client programs because only a single copy of a particular document is cached. Furthermore, the proxy server can use much more complex algorithms to predict which documents are worth caching for a long time and which not as the proxy can base its statistics about how often documents are referenced by multiple clients on a larger sample size.

Currently, not all WWW servers offer proxy capabilities. CERN's httpd was the first WWW server which had a fully built-in proxy support together with document caching (see [LUOTONEN94] for detailed information about proxies).

There is only one difference in the protocol when using a WWW server as a proxy. When normal HTTP requests are made by clients, the server gets only the path part of the URL, since the protocol specifier "`http:`" and the host part are obvious. When a request is sent to a proxy server, however, the full URL has to be specified because the proxy needs this information to make the actual request to the remote server.

There is not much extra functionality necessary for clients when they run in proxy mode. The only difference is that they have to send all their requests to a particular proxy server instead of sending requests directly to remote servers. This functionality is, for example, implemented in CERN's library of common code. Hence, all clients developed by using this library automatically can be set up for proxy support. For this purpose a set of environment variables, one for each protocol, have been defined.

- `http_proxy`
- `ftp_proxy`
- `gopher_proxy`
- `wais_proxy`

These environment variables have to be set to the URL pointing to the proxy server. This causes the client to send all requests to that proxy server and to specify the full URL of the resource requested instead of specifying just the path portion on the first line of the HTTP request.

4.7.2. Using the WWW Gateway as a Proxy Gateway

As the Hyper-G server has its own document cache server that is able to retrieve and cache non-Hyper-G documents as well (compare 3.3.5.), the idea came up to add some functionality to the WWW gateway in order to build a proxy server for WWW documents.

At first, the gateway needs to distinguish requests for resources within Hyper-G from requests that normally would go directly to some remote server. The normal situation is when a requesting client is not set up in proxy mode. This is illustrated in figure 4.16.

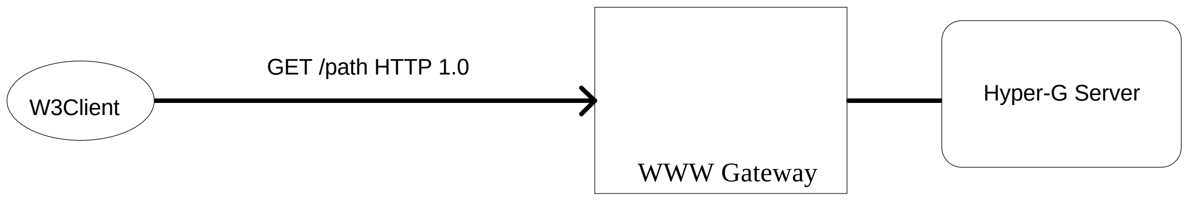


Figure 4.16

The gateway only gets the path portion of the URL, since its host name and the protocol specifier are obvious. The request is processed, usually by getting some document from the Hyper-G server and sending the converted document to the WWW client.

When a client is set up in proxy mode, however, the situation is slightly different. If the client has to use the WWW gateway as its proxy server, it will send every request to the gateway independently of its URL. Moreover, it will use full URLs with every request. If the URL references a resource on a remote non-Hyper-G server, we have a situation like in figure 4.17.

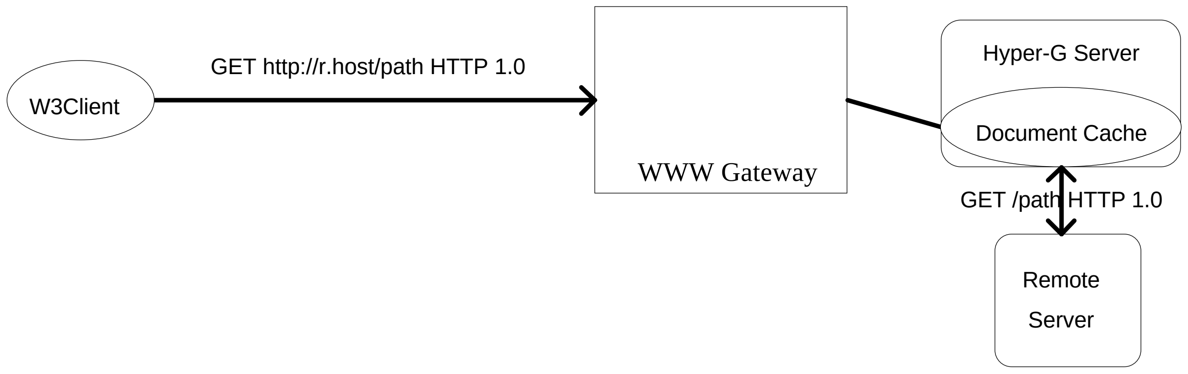


Figure 4.17

The request is passed to the document cache server, which retrieves the document from a remote server like a normal client. It sends the document to the gateway and stores a copy of it in its cache. The gateway passes the document to the WWW client.

When the same document is requested once more and the document cache server still has the copy in its cache, no request to the remote server will be necessary this time. The gateway will receive the document much faster.

The approach taken to implement proxy functionality into the WWW gateway results in a program that is not only able to handle requests for information within Hyper-G but also requests for remote non-Hyper-G documents.

However, for retrieving remote documents, the concept of keeping up a Hyper-G session for each WWW client (see 4.2.) is not regarded as ideal because with remote non-Hyper-G documents there is no possibility to tell the client to use a session key with its requests. Therefore, it has been decided to treat proxy requests differently. Unlike normal Hyper-G requests, proxy requests do not cause a slave process to wait for further requests from the same client. A gateway process that handles a proxy request exits after it has sent the response to the client. When we look at the process model described in 4.2.3., the functionality of handling proxy requests can easily be added to that of the passing process, as this is the type of process that is generated on every new request and that exits after the request has been processed.

The only problem is how to distinguish proxy requests from normal Hyper-G requests, as they should not be directed to the document cache but have to be converted into client server transactions between the gateway and the local Hyper-G server or sometimes they are even processed by the gateway itself (e.g. request for forms, icons, etc.). It is not sufficient for the gateway to base the decision solely on the fact whether it receives a full or partial URL because a client which has been set up in proxy mode always sends full URLs. This is why the gateway has to check the host portion of the URL whether or not it refers to the gateway's host and port number. If it does not, the request is treated as a proxy request and the gateway passes it to the document cache server. If the internet address in the URL correspond to the internet address of the gateway's host and the port number in the URL correspond to the port number used for WWW transactions by the gateway, the protocol and host part is stripped off from the URL and the request is handled as a normal Hyper-G request.

Proxy functionality has been added to the WWW gateway in order to find out if the gateway together with the document cache of the Hyper-G server can work as a proxy server. However, it has to be mentioned that at the moment this works only with HTML documents that do not contain fill-forms because the document cache server does not support HTTP 1.0 so far. Therefore it is not possible for the gateway to pass request headers or to use method POST. Also, the use of other protocols such as Gopher or ftp would require some extensions and modifications in the document cache server.

Bibliography

[ANDREESSEN93] Andreessen M.:
NCSA Mosaic Technical Summary, May 93,
anonymous ftp ftp.ncsa.uiuc.edu /Web/Mosaic/Papers/mosaic.ps.Z

[ANDREWS94] Andrews K., Kappe F., Maurer H., Schmaranz K.:
On Second Generation Hypermedia Systems, J.UCS, Vol. 0, No 0, Nov 94

[ANDREWS95] Andrews K., Kappe F., Maurer H.:
The Hyper-G Network Information System, Information Processing and Management, 95, Special Issue: Selected Proceedings of the Workshop on Distributed Multimedia Systems, Graz, Austria, Nov. 94.

[BERNERS-LEE92] Berners-Lee T., Cailliau R., Groff J.-F., Pollermann B.
World-Wide Web: An Information Infrastructure for High-Energy Physics, presented at "Software Engineering, Artificial Inteligence and Expert Sytems for High Energy and Nuclear Physics" at La Londe-les-Maures, France, Jan. 1992

[BERNERS-LEE92a] Berners-Lee T., Cailliau R., Groff J.-F., Pollermann B.
World-Wide Web: The Information Universe, Electronic Networking: Research, Application and Policy, Vol 1 No.2, Meckler, Westport CT, Spring 92

[BERNERS-LEE93] Berners-Lee T.:
Hypertext Transfer Protocol: A Stateless Search, Retrieve and Manipulation Protocol, Internet draft, Nov. 1993,
anonymous ftp ftp.w3.org pub/www/doc/http-spec.ps

[BERNERS-LEE93a] Berners-Lee T., Connolly D.:
Hypertext Markup Language: A representation of Textual Information and Metainformation for Retrieval and Interchange, Internet draft, Jul. 1993,
anonymous ftp ftp.w3.org pub/www/doc/html-spec.ps

[BERNERS-LEE94] Berners-Lee T.:
Uniform Resource Locators: A unifying syntax for the expression of names and addresses of objects on the network, Internet draft, Jan. 1994, anonymous ftp ftp.w3.org pub/www/doc/draft-ietf-uri-url-03.txt.ps

[BERNERS-LEE94a] Berners-Lee T., Cailliau R., Loutonen A., Frystyk H., Secret A.:
The World-Wide Web, Communications of the ACM, Vol 37, No 8, Aug. 94, p. 76-82

[BINA94] Bina E.:
Mosaic for X version 2.0 Fill-Out Form Support, http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html

[BORENSTEIN92] Borenstein N., Freed N.:
MIME Multipurpose Internet Mail Extensions, Internet RFC 1341, Jun. 92, anonymous ftp nic.ddn.mil rfc/rfc1341.txt

[CAILLIAU95] Cailliau R.:
About WWW, Information Processing and Management, 95, Special Issue: Selected Proceedings of the Workshop on Distributed Multimedia Systems, Graz, Austria, Nov. 94.

[CROCKER82] Crocker D.:
Standard for the format of Internet text messages, Internet RFC 822, Aug. 82, anonymous ftp nic.ddn.mil rfc/rfc822.txt

[GOLDFARB90] Goldfarb C.:
The SGML Handbook, Oxford university Press, 90

[HORTON83] Horton M.:
Standard for Interchange of USENET Messages, Internet RFC 850, Jun. 83, anonymous ftp nic.ddn.mil rfc/rfc850.txt

[HUGHES93] Hughes K.:
Entering the World-Wide Web: A Guide to Cyberspace, Honolulu Community College, Sep. 93, http://www.hcc.hawaii.edu/guide/www.guide.html

[KAHLE92] Kahle B., Morris H., Davis F., Tiene K., Hart C., Palmer R.:
Wide Area information Servers: An Executive Information system for Unstructured Files, Electronic Networking: Research, Applications and policy, Vol 2, No 1, Spring 92, p. 59-68

[KAPPE91] Kappe F., Maurer H, Tomek I.:
Hyper-G: Specification of Requirements, IIG Report 284, IIG, Graz University of Technology, Apr. 91

[KAPPE91a] Kappe F.:
Aspects of a Modern Multi-Media Information System, PhD thesis Graz University of Technology, Jun. 91, also available as IIG Report 308, IIG, Graz University of Technology, Jun. 91, and by anonymous ftp from iicm.tu-graz.ac.at in pub/Hyper-G/doc

[KAPPE92] Kappe F., Maurer H, Sherbakov N.:
Hyper-G: A Universal Hypermedia System, IIG Report 333, IIG, Graz University of Technology, Mar. 92

[KAPPE92a] Kappe F., Pani G.:
The Architecture of A Massively Distributed Hypermedia System, IIG Report 341, IIG, Graz University of Technology, Sep. 92

[KAPPE93] Kappe F., Maurer H.:
From Hypertext to Active Communication/Information Systems, IIG Report 363, IIG, Graz University of Technology, May 93

[KAPPE93a] Kappe F., Maurer H.:
Hyper-G: A Large Universal Hypermedia System and Some Spin-Offs, IIG Report 364, IIG, Graz University of Technology, May 93

[KAPPE93b] Kappe F., Andrews K., Faschingbauer J., Gaisbauer M., Pichler M., Schipflinger J.:
Hyper-G: A New Tool for Distributed Hypermedia,
anonymous ftp ftp.iicm.tu-graz.ac.at pub/Hyper-G/papers/report388.ps

[KROL94] Krol E.:
The Whole Internet, User's Guide & Catalog, Second Edition, O'Reilly & Associates, Inc., Apr. 94

[LINDNER94] Lindner P. (Editor):
Internet Gopher User's Guide, University of Minnesota, Apr. 94,
anonymous ftp boombox.micro.umn.edu
pub/gopher/docs/GopherGuide_Jan12-94.A4.ps

[LUOTONEN94] Luotonen A., Altis K.:
World-Wide Web Proxies, Apr. 94,
http://info.cern.ch/hypertext/WWW/Proxies/

[MAURER90] Maurer H., Tomek I.:
Some Aspects of Hypermedia Systems and their Treatment in Hyper-G,
Wirtschaftsinformatik, Vol 32, No 2, Apr. 90, p. 187-196

[MAURER94] Maurer H., Philpott A., Sherbakov N.:
Hypermedia Systems without Links, Journal of Microcomputer applications,
Vol 17, No 4, Oct. 94, p.321-332

[MAURER94a] Maurer H., Schmaranz K.:
J.UCS - The Next Generation in Electronic Journal Publishing, Vol 0, No 0,
Nov 94

[MOCKAPETRIS87] Mockapetris P.:
Domain names - Concepts and Facilities, Internet RFC 1034,Nov. 87
anonymous ftp nic.ddn.mil rfc/rfc1341.txt

[RAGGETT94] Raggett D., Lie H., Frystyk H., Hallam-Baker P.:
Arena: W3o's HTML3 browser, home page,
http://info.cern.ch/hypertext/WWW/Arena