# Object-Stacking in the World-Wide Web

Yasushi Shinjo, Satoshi Zakimi, Seiki Kyan

Department of Information Engineering
University of the Ryukyus
Nishihara, Okinawa 903-01, Japan
Phone: +81 98 895 2221 Ext. 3266
Fax: +81 98 895 2688
E-Mail: yas@ie.u-ryukyu.ac.jp
URL: http://bottom.ie.u-ryukyu.ac.jp/~yas/

**Abstract**

*Object-stacking is a model for structuring object based systems. The main feature of object-stacking is that layers of objects with a uniform interface are constructed, and the functions of these objects are integrated. The effectiveness of object-stacking has been shown for file systems of distributed operating systems. This paper presents the application of the object-stacking model to the World-Wide Web, an information exploring/providing system on the Internet. Object-stacking gives powerful tools to information providers who use the World-Wide Web. This paper describes the implementation method of object-stacking in the World-Wide Web. This paper proposes the new concepts of demand stacking, virtual stackable objects, and pointer swizzling in the World-Wide Web.*

## 1 Introduction

*Object-stacking* is a model for structuring object-based systems[10][11]. The main feature of object-stacking is that layers of objects with a uniform interface are constructed, and the functions of these objects are integrated. Object-stacking can be used in object-based systems in which the client-server model of interprocess communication is satisfied. In this paper, we describe the use of object-stacking in the World-Wide Web.

*The World-Wide Web* (W3) is an information exploring/providing system on the Internet based on hypermedia [1]. W3 provides still image data, video data, audio data as well as text data.

W3 has many good user interface programs, such as Mosaic, Netscape, etc. However, W3 provides a few tools to the information providers. On the provider side of W3, maintaining consistency among multiple views of a single data is one of the biggest problems. Suppose that there is a large image data. It is desired to provide a small view of the original large data for the users who accesses through a slow communication line by a personal computer with a monochrome display. The small view is created by reducing its colors and scaling its size. It is hard to maintain several views consistent for data which is updated frequently.

In Japan and other countries, the multiple view problem arises in character encoding. For example, Japanese is usually expressed by using thousands of *Kanji characters* (Chinese characters). Usually, Japanese kanji is encoded by using one of the following three coding methods:

(1) JIS (Japan Industrial Standard) kanji code. This is based on an ISO standard [7].
(2) Shift-JIS kanji code by Microsoft.
(3) EUC (Extended Unix Code) for Japanese.

JIS and EUC are used in UNIX applications. Shift-JIS is used in personal computers, such as Macintosh and IBM PC/AT compatibles with MS-DOS or MS-Windows. It is hard for information providers to maintain these three views consistently.

W3 is used to distribute information widely. It can be used to share information within a closed group of people. To accomplish this, many efforts are made on authentication and encryption.

These problems have been solved in file systems by using object-stacking [10][11]. Object-stacking is proposed for providing multiple view, first [11]. In file systems, kanji code transformation and image conversion are done by object-stacking. Encryption and compression are popular applications of object-stacking.

In this paper, we describe the implementation method of object-stacking in W3. We have transferred some techniques developed in file systems into W3. We also developed new techniques, such as *demand stacking*, *virtual stackable objects*, and *pointer swizzling* in W3. We present these new techniques in depth in this paper. And, we compare our techniques with others.

*1995 International Workshop on Object-Orientation in Operating Systems (I-WOOOS'95)*
*University of Lund, Sweden, August 14-15th, 1995.*

## 2 Object-stacking in file systems

The key idea of object-stacking is *to stack objects* that have a *uniform interface*. *To stack* Object b1 (an upper object) *on* Object a1 (a lower object) means that Object b1 holds the *id* (identifier) of Object a1, and Object b1 calls Object a1 to implement its functions. The *interface* of an object is the set of its exported procedures. To choose objects flexibly, they must have a *uniform interface*. If interfaces of objects are not uniform, the objects that can be stacked on an object are restricted.

Figure 1 shows stacking of three file objects. We define *a bottom object* as an object that holds no object id. In Figure 1, Object a1 is a bottom object. We define *a stackable object* as an object that holds an id of a lower object which interface is equivalent to that of itself, and it calls the lower object to implement its function. In Figure 1, Object b1 and Object c1 are stackable objects.

In Figure 1, each object provides the following functions, respectively.

c1: Sort.
b1: Translate upper case characters to lower case ones.
a1: Store data.

These functions are integrated into the top object, Object c1. That is, an object is created which has the functions to store lower-case sorted data.

In Figure 1, when each stackable object accepts a read request from its client (including its upper object), the stackable object sends a read request to the corresponding lower object. Next, the stackable object performs the appropriate filter operation (sorting or translation) on the result that is returned from the lower object. Finally, the stackable object returns the filtered data to the client.

In Figure 1, the ids of Object c1, b1 and a1 are registered into a directory object. These registered ids can be regarded as three *views* of a single object. In Figure 1, the following views are provided:

(1) The original view.
(2) The lower-case view.
(3) The sorted lower-case view.

Moreover, multiple view can be implemented by stacking several upper objects on a single lower object [10].

In file systems based on object-stacking, file objects or directory objects are stacked, and their high-level functions are integrated. In object-stacking, each object provides a simple function, such as filtering, indirection, caching, masking, or grouping. By stacking such objects, their functions are integrated and can be used together.

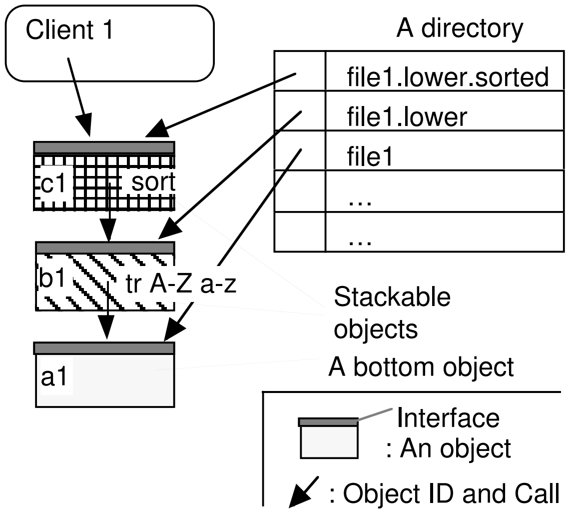


Figure 1: Integrating the functions of objects by stacking in a file system.

### 2.1 Server-stacking and server-cascading

In object-stacking, layers are constructed from objects. Similarly, layers can be constructed from servers. We refer this method as *server-stacking*. Server-stacking is implemented in the Ficus system [3]. In Ficus, the interface of NFS vnode is used as a uniform interface.

Server-stacking is a special and protected case of object-stacking. Protection means prevention of creation of illegal stacks of objects [10].

In this paper, we describe both object-stacking and server-stacking in the World-Wide Web. We use the term "a *stackable server*" for denoting both a server of stackable objects in object stacking and a server in server-stacking.

In the Spring system, layers of file servers are constructed for implementing a distributed file system and an attribute-caching file system [4]. In the implementation of those file systems, a single server provides two types of objects which are related internally. We call this technique *server-cascading*. Server-cascading is a special and protected case of *object-cascading* [9]. We are now implementing a new window system based on object-cascading.

### 2.2 Object-stacking and delegation

The mechanism of object-stacking is similar to *delegation* [5][13]. Delegation is a mechanism to implement shared behavior in object-oriented systems.

In delegation, behaviors of objects are shared *implicitly*. If variables and methods are not defined in an object, its prototype is searched automatically. In delegation, there is a special variable, named *self*, which holds the object that receives a message first from a client. These two mechanisms make delegation powerful as inheritance.

In object-stacking, an object can use functions of its lower objects by calling the lower objects *explicitly*. In object-stacking, explicit procedure invocation is needed to forward received messages to lower objects. The methods that are not defined in an object are not forwarded to its lower objects, but they are rejected. In delegation, to disable automatic invocation of prototypes, it is needed to redefine the methods in the object.

# 3 Object-stacking in the World-Wide Web

*The World-Wide Web* (*W3*) is a wide-area information exploring/providing system based on hypermedia. In this section, we show an overview of the mechanism of W3, first. Second, we present several stackable objects in W3. Finally, we discuss the W3-Proxy server as a stackable server in W3.

## 3.1 The overview of the W3 mechanism

W3 has come to stand for the following definitions:

**(1) HTML (HyperText Markup Language):** HTML is a markup language used to create hypertext resources in W3. To express hyperlinks to other resources, HTML uses URL below.

**(2) URL (Uniform Resource Locator):** URL is a compact representation of the location and the access method for a resource available on the Internet. URL used in W3 has the following form:

`protocol://host:port/dir/filename`

The "`protocol`" indicates a communication protocol[1]. Usually, the protocol is "`http`" which will be described below. URL can also express other protocols. The "`host`" and the "`port`" are the host name of the server of the resource and the port number of the server in TCP/IP. If the port number is omitted, the default value of the protocol is used. The following string "`/dir/filename`" is the string that is used by the server to distinguish resources in the server. Usually, this string means a relative path name from the working directory of the server in a file system.

**(3) HTTP (HyperText Transfer Protocol):** HTTP is a generic stateless communication protocol on TCP/IP [1]. HTTP is based on the client-server model of interprocess communication [12]. HTTP defines interaction between clients and servers. In HTTP, the `GET` request is the most frequently used request. To get a resource, a client sends a `GET` request with a URL as an argument to a server. When the server receives the `GET` request, the server searches for the resource which is pointed to by the URL, and sends the content of the resource. In HTTP, the following requests are also defined: `POST` for creation of resources, `PUT` for updating resources, `DELETE` for deleting resources. In HTTP, a connection of TCP/IP is established every time a single resource is transferred.

URL has two forms: the full form and the partial form. *The full form* of URL consists of a protocol, a host name, an optional port number, and a path name of a file as described above. *The partial form* of URL is a path name of a file. For example, consider that a resource described in HTML is denoted by the URL "`http://host1/port1/dir1/file1.html`". If this resource contains the URL "`file2.html`" in a partial form, the URL "`file2.html`" points to the resource denoted by the full URL "`http://host1/port1/dir1/file2.html`". Note that the full form and the partial form are also called *the absolute form* and *the relative form*, respectively.

The following is an example interaction between a client and a server of HTTP. First, suppose that the client needs a resource denoted by the next URL:

`http://host1:port1/Welcome.html`

The client analyzes this URL, and knows that the server on `host1` is waiting for requests at `port1`. So, the client opens a TCP/IP connection to the server, and sends the following request in HTTP.

`GET /Welcome.html HTTP/1.0`

"`GET`" is the request type, "`/Welcome.html`" is the URL of the document in the partial form, and "`HTTP/1.0`" is a version of HTTP.

When the server accepts the above request, the server opens the file "`Welcome.html`" in the working directory, and reads the contents of the file. Then, the server sends the contents of the file to the client through the TCP/IP connection.

Figure 2 shows an example of a hypertext resource in HTML returned by a server. The first line is called *a status line* in HTTP. This line indicates the status of the response. In this example, the status is OK. The next line means that the contents of the response body are in HTML. The next empty line is the delimiter of the response header and the response body.

[1] In URL definition, the term "scheme" is used insted of the term "protocol". For example, "scheme" includes "file" for local files.

```
HTTP/1.0 200 Document follows
Content-Type: text/html

<HEAD><TITLE>University of the Ryukyus W3
Home Page.  </TITLE></HEAD>
<BODY>
<IMG SRC="title.gif">
<H2>
Welcome to W3 server at the University of
the Ryukyus.
</H2>
If you want to change to Japanese,
<A HREF="Welcome-j.html">click here.  </A>
<A HREF="./UR_history/history-e.html">About
the University.</A>
</BODY>
```

Figure 2: A W3 resource in HTML.

In the response body, `<IMG SRC="title.gif">` means that the content of the image file "`title.gif`" should be expanded there. The region from `<A>` to `</A>` is called an *anchor*. An anchor means the starting point of a hyperlink (hyper reference, `HREF`) to another resource.

When the client application receives this resource, the client application formats the resource, and displays the formatted resource to the user. For example, the client application expands inline images, and changes the fonts for anchors.

Besides HTTP, other protocols, such as FTP, Gopher, and WAIS are supported in W3 [1]. In this paper, we deal with only HTTP.

Our research focuses on support for information providers of W3. In this paper, the term "user" means a human who uses a client application.

### 3.2 Stackable objects in W3

To implement object-stacking, we have to find objects and a uniform interface. We can regard the resources that are denoted by URLs and transferred via HTTP as objects. As objects with a uniform interface, we use the objects that can accept the GET request of HTTP. Now, we define the following terms:

- *An HTTP object* is an object which is transferred via HTTP.
- *A text object* is an HTTP object which content type is text.
- *An HTML object* is a text object which is described in HTML.
- *An image object* is an HTTP object which content type is image.

As the uniform interface in W3 is defined, stackable objects in W3 are defined inferentially. *A stackable HTTP object* holds another HTTP object as a lower object. When it receives a `GET` request from a client, it sends a `GET` request to its lower object. Then, it receives a response message from the lower object, performs an operation on the response message, and sends a response message to the client.

Figure 3 shows two stacks of HTTP objects. The objects in Figure 3 provide the following functions, respectively:

(c1, c2) Reduce the sizes for image objects. Do nothing for other types of objects.

(b1, b2) Translate kanji code into Shift-JIS for text objects. Do nothing for other types of objects.

(a1, a2) Store data.

These functions are integrated into the top objects, Object c1 and c2. That is, a shift-JIS text object and a small image object are created.

Figure 4 shows the effects of the stackable objects. This hard copy was made on Macintosh which supports shift-JIS kanji code. On the left of Figure 4, a large image and broken kanji characters are displayed. These correspond to Object a1 and a2 in Figure 3. On the right of Figure 4, a small image and correct kanji characters are displayed. These correspond to Object c1 and c2 in Figure 3.

We have implemented the following stackable HTTP objects:

(1) Image conversion objects. An image conversion stackable object converts a lower image object using one image format to an image object with a different image format. This object is also used for scaling and changing the number of colors. This stackable object helps a user who accesses through a slow line, such as an international line or a home telephone line, to reduce the size of an image object. This also helps an information provider who uses an image scanner. She/he can save images in a scanner-specific format in spool directories. Simultaneously, people in the Internet can view those images in a standard format, such as GIF (Graphic Image Format). These objects are implemented by using the `convert` command in the ImageMagic package.

(2) Kanji filter objects. A kanji filter stackable object translates the kanji code of its lower HTTP object into specified one. As described in Section 1, there are three popular encoding methods for Japanese. By using these objects, an information provider can provide three views of a single text data which includes kanji characters.

(3) Encryption/decryption objects. These are used to share information within a closed group of people who are distributed in the Internet. By using these objects, information can be shared safely over insecure networks.

(4) General-purpose stackable HTTP objects which invoke UNIX filter commands. By using these objects,

filter commands in UNIX are translated into stackable HTTP objects.

(5) M4 macro objects. These are instances of the above objects. These objects invoke the m4 macro language processor of UNIX.

We have a plan to implement the following stackable HTTP objects:

(1) Kanji-to-Romaji translation objects. These objects translate documents with kanji characters into that with alphabet characters. These objects help Japanese learners and native Japanese people who are using a client application which cannot display kanji characters.

(2) Read aloud objects. These objects transform text data into audio data. These objects will use Kanji-to-Romaji translation objects for preprocessing.

(3) Scheduling/congestion control objects. These objects observe the usage of network resources and the loads of servers, and schedule the requests and control speeds of transfer.

### 3.3 W3-Proxy is a stackable HTTP server

W3-Proxy of CERN is an HTTP server typically running on a firewall machine, providing with access to the outside world for people inside the firewall [6]. It has a facility of caching of resources.

In terms of ours, W3-Proxy is a stackable HTTP server. It can be used together with our stackable servers for caching and crossing firewall.

Unlike our stackable servers, W3-Proxy does not change the contents of lower objects. Our stackable server changes the contents of lower objects. This change includes rewriting of URLs in HTML objects. About rewriting of URLs, we will describe in Section 4.2 in depth.

The main supporting target is different between W3-Proxy and our stackable HTTP servers. The former supports mainly information users, and the latter supports mainly information providers.

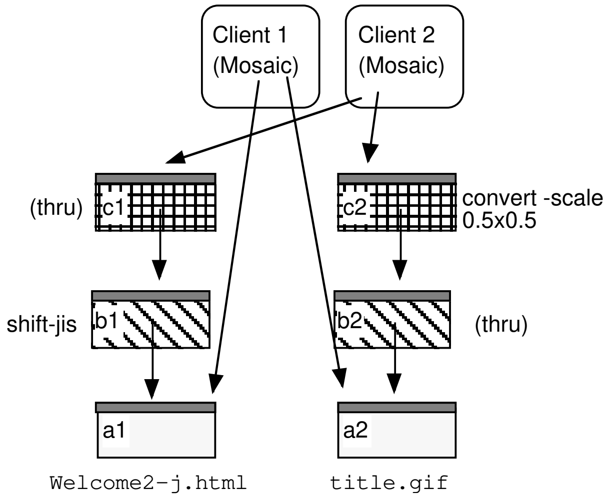


Figure 3: Integrating the functions of objects by stacking in the World-Wide Web.

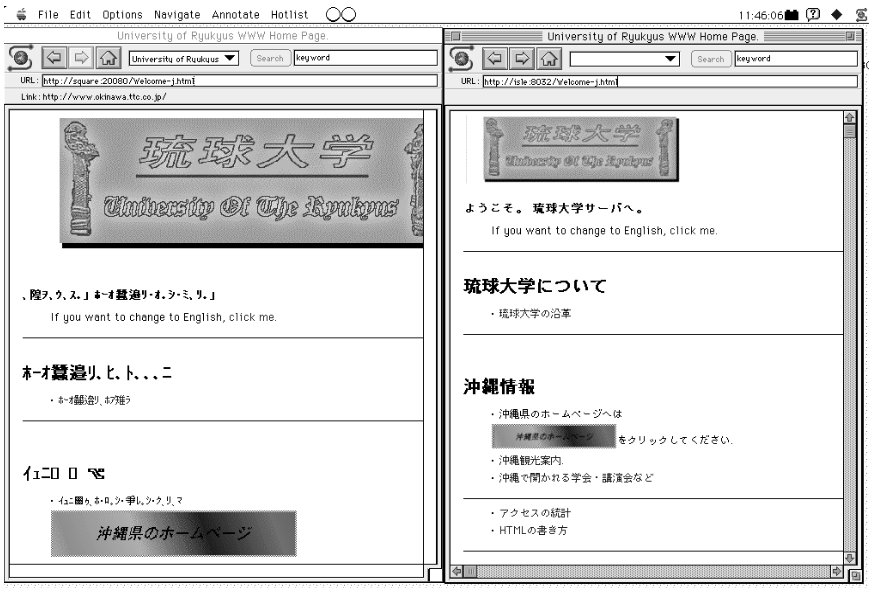


Figure 4: Formatted HTTP documents (original and filtered).

# 4 Demand stacking, virtual stackable objects and pointer swizzling

As compared with file systems, W3 has the following features:

(1) No directory object is needed. URLs, which correspond to object ids, include symbolic names, and they are human-readable.

(2) HTML objects include references to other objects.

By the first feature, object creation methods in W3 become different from that in file systems. This is discussed in Section 4.1, and two new ideas are proposed: *demand stacking* and *virtual stackable objects*. The second feature requires the rewriting of URLs in HTML objects. We call this rewriting *pointer swizzling*. We describe pointer swizzling in Section 4.2.

In this paper, we describe both object-stacking and server-stacking in W3. These can be chosen by information providers. From a technical point of view, both object-stacking and server-stacking have many common aspects, as described in Section 2.1. The differences appear in pointer swizzling in W3.

### 4.1 Demand stacking and virtual stackable objects

This subsection describes how to create stacks of objects in W3. In file systems, stacks of objects are created step by step, or duplicated from existing ones in template object bases all at once [10]. In any case, stacks of objects are constructed from bottom to top. In W3, these operations are troublesome since there are so many objects in the world. In addition, W3 has human-readable object ids, URLs. URLs should be used to create stacks.

Now, we introduce the new idea of demand stacking for stack creation in W3. *Demand stacking* means that stacking is done on demand by a client. We call a stackable object created on demand as *a virtual stackable object*. In contrast to this new stacking, we refer old stacking in file systems as *pre-stacking*. In pre-stacking, stackable objects are created by calling object creation procedures first, and then the ids of new stackable objects are fixed later. Stacks of objects are constructed from bottom to top. In demand stacking, virtual stackable objects are created by accessing with the object ids (URLs) first, and then stacks of objects are created later. Stacks of objects are constructed from top to bottom.

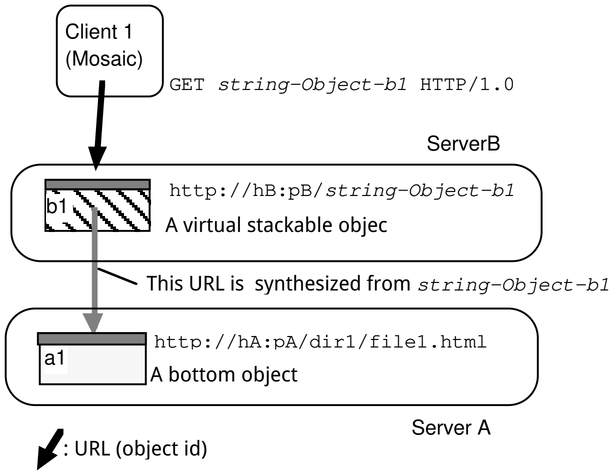


Figure 5: Creation of a virtual stackable object on demand.

An example of creation of a virtual stackable object is shown in Figure 5. In Figure 5, Object a1 is a bottom object, and it is identified by the following URL:

(a1) `http://hA:pA/dir1/file1.html`

This bottom object is managed by a regular HTTP server, named Server A on host `hA` at port `pA`.

Now, let us create a virtual stackable object, named Object b1, which holds the id of Object a1 as its lower object. Suppose that Server B, the server of Object b1, is running on host `hB`, and accepting requests at port `pB`. As Object b1 is a virtual stackable object, it is not needed to send an object creation request (`POST` in HTTP) to Server B. Instead, the following object read request causes creation of Object b1:

`GET /`*`string-Object-b1`* `HTTP/1.0`

Where "*string-Object-b1*" describes Object b1 in Server B. The full URL of Object b1 is:

(b1) `http://hB:pB/`*`string-Object-b1`*

When Server B accepts the `GET` request, Server B creates Object b1 which lower object is Object a1. Object b1 sends the `GET` request to Object a1.

In demand stacking, "*string-Object-b1*" is important. In pre-stacking, a server knows the id of a lower object before creation of a new stackable object, and the server saves the id of the lower object in a working file. When the server receives a request message to a stackable object, the server fetches the id of the lower object from the working file, and uses the fetched id to send request messages to the lower object. In demand stacking, on the other hand, the server receives the object id ("*string-Object-b1*") first. The server has to find the id of the lower object in the given object id. In the case of Object b1, the lower object is Object a1 which is identified by the URL "`http://hA:pA/dir1/file1.html`". Therefore, "*string-Object-b1*" must include enough information to synthesize the URL of Object a1.

We use the following two types of strings to identify Object b1 in Server B:

(1) A string including the full URL of Object a1.

(2) A string including the partial URL of Object a1.

The former is used in object-stacking. This type of string enables a stack creator to choose a lower object without any restriction. The latter is used in server-stacking. A creator of a stackable object has a free choice of a lower object within the Server A, but the creator cannot specify an object in another server.

In general, we use the following form for a virtual stackable object in object-stacking:

`http://hS1:pS1/options1!`*`lower-url-1`*

where the string "*lower-url-1*" is a full URL which starts with "`http://`". If this lower object is also a virtual stackable object, the URL of the virtual stackable object is as follows:

```
http://hS1:pS1/options1!\
http://hS2:pS2/options2!lower-url-2
```

The symbol '**!**' stands for the delimiter of host names in the old UUCP (UNIX-to-UNIX-CoPy) fashion. In UUCP, users have to specify all relay hosts to the final destination host from left to right, like "`relay-host1!relay-host2!...!destination-host`". In pipeline of UNIX, data flows from left to right of the symbol '**|**'. In a URL of a virtual stackable object, requests propagate from left to right of the symbol '**!**', and response data flows from right to left.

### 4.2 Pointer swizzling

Unlike file systems, HTML objects in W3 include references (URLs) to other objects. To implement object-stacking and server-stacking, these references should be rewritten consistently. We define this rewriting *pointer swizzling*. This term stands for pointer swizzling in OOOS (object-oriented operating systems) and OODB (object-oriented databases), of course [2][14].

In OOOS and OODB, pointer swizzling is a technique to speedup navigation operation. In those environments, objects are copied from the server (the object manager) into the address space of an application process. In a server, references are represented by object ids. If a copied object includes a reference to an object in memory, this reference is turned (*swizzled*) into the physical pointer to the object in memory.

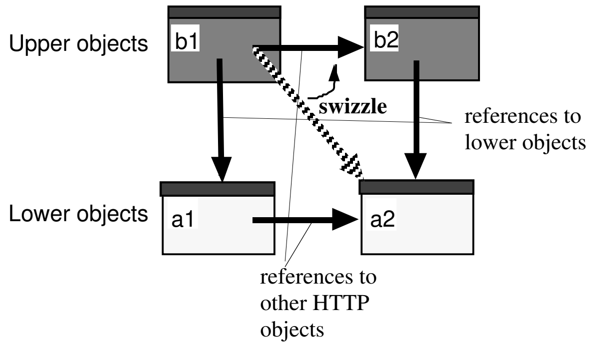


Figure 6: Pointer swizzling in W3.

In stacking HTML objects, pointer swizzling is needed as similar to OOOS and OODB. Let us consider the example in Figure 6. In Figure 6, there are two bottom HTML objects, Object a1 and Object a2. On those two objects, Object b1 and Object b2 are stacked, respectively. Object a1 includes the reference to Object a2. In other words, Object a1 includes the URL of Object a2. The corresponding reference (URL) in Object b1 should be rewritten to the URL of Object b2.

To implement pointer swizzling, we use different strategies for object-stacking and server-stacking.

#### 4.2.1 Pointer swizzling in server-stacking

As described in Section 3.1, URL has two forms: the full form and the partial form. In server-stacking, we rewrite only the partial (relative) form of URL, and we leave the full (absolute) form of URL unchanged. This strategy can be implemented easily in virtual stackable objects. Each virtual stackable object simply passes URLs in the lower HTML object to the client.

In this strategy, information providers have to describe references in the partial (relative) form of URL. Using the partial form of URL is a regular way to describe hypertext in HTML.

#### 4.2.2 Pointer swizzling in object-stacking

In object-stacking, we have to rewrite both the partial (relative) form and the full (absolute) form of URL. This is because virtual stackable objects in object-stacking always require the full form of URL to access lower objects.

The rewriting rules are as follows:

(1) For a URL in the full form, the next prefix is attached:

```
http://self_host:self_port/self_opts!
```

(2) For a URL in the partial form, the next prefix is attached:

```
http://self_host:self_port/self_opts!\
http://lower_host:lower_port/\
```

```
working_directory/
```

"self_host" and "self_port" are the host name and the port number of the server of the target virtual stackable object, respectively. "lower_host" and "lower_port" are the host name and the port number of the server of the lower object, respectively. "working_directory" is the directory name part of the URL of the target virtual stackable object.

For example, let us consider a virtual stackable object, named Object b1, which is identified by the following URL:

```
http://hB:pB/!\
http://hA:pA/dir1/file1.html
```

We name the lower object of Object b1 as Object a1. The URL of Object a1 is next:

```
http://hA:pA/dir1/file1.html
```

Suppose that Object a1 includes the following URL in the partial form:

```
HREF="file2.html"
```

This URL is rewritten into the following URL:

```
HREF="http://hB:pB/!\
http://hA:pA/dir1/file2.html"
```

As this URL denotes that the server of this object is on host hB at port pB, the following GET request to this object is directed to the server on host hB at port pB.

```
GET /!http://hA:pA/dir1/file2.html \
HTTP/1.0
```

This server can access the lower object, Object a1 by extracting the URL of Object a1 from the URL of Object b1.

Suppose that Object a1 includes the following URL in the full form:

```
HREF="http://h3:p3/dir3/file3.html"
```

This URL is rewritten into the following:

```
HREF="http://hB:pB/!\
http://h3:p3/dir3/file3.html"
```

### 4.3 W3-Proxy and pointer swizzling

In W3-Proxy, pointer swizzling is not performed. This is because W3-Proxy forces clients to send all requests to W3-Proxy. This forcing changes the old model of W3 clients. Pre-proxy clients send request messages to final destination servers directly, as described in Section 3.1. To use W3-Proxy, the programs of pre-proxy clients have been modified to send all requests to W3-Proxy. For our stackable HTTP server, in contrast, this modification is not needed because of pointer swizzling.

In W3-Proxy, all requests come to W3-Proxy. This property can be used to use object-stacking in clients' sites (information users' sites). For example, kanji code translation can be implemented in clients' sites easily.

### 4.4 M4 macro stackable HTTP server

To use object-stacking in W3, information providers have to write URLs of virtual stackable objects in HTML objects. Since the URLs of virtual stackable objects are very long in object-stacking, writing such URLs is a painful work. We solve this problem by using stackable objects that invoke the m4 macro processor of UNIX. The m4 macro processor includes the following definitions before the contents of the lower object:

```
define(`TEXT_JIS',   http://host1:port1/)
define(`IMAGE_SCALE',http://host2:port2/)
define(`IMAGE_MONO', http://host3:port3/)
define(`M4_FILTERED', 1)
```

So information providers can use the following short URL for a stackable virtual object:

```
Monochrome and JIS-code data is
<A HREF="IMAGE_MONO!TEXT_JIS!\
http://host4/dir4/file4.html">here </A>.
```

Note that information providers do not have to write long URLs in all pages. Instead, they have to write long URLs in several key pages, such as home pages. This is because other pages followed by the key pages become virtual stackable objects recursively.

## 5 Implementation

We have implemented several stackable HTTP servers. In implementation of those servers, we have transferred some techniques developed in file systems into W3. We have also developed new techniques, such the virtual stackable objects, demand stacking, and pointer swizzling described in Section 4. In this section, we show the implementation of these new techniques and compare the implementation with that in file systems.

### 5.1 The structure of stackable HTTP servers

The structure of stackable HTTP servers is similar to that of stackable file servers [10]. The differences between file systems and W3 appear in handling of object ids and resident objects in memory.

In file systems, numeric object ids are used. Data of stackable file objects is stored in a working file which is maintained by the server. When a server receives a request, the server first transfers the numeric object id into the pointer to the object in main memory. The object id is used to locate data in the working file. The object in main memory has several states and caches inside or outside the server. According to the request, the object in main memory should be locked.

In contrast to a stackable file object, a virtual stackable HTTP object is created on demand, as described in Section 4.1. A URL, which is the object id of a virtual stackable HTTP object, contains all information about the virtual stackable HTTP object, including the id of the lower object. Therefore, no fetching operation from files is needed. A virtual stackable object is lightweight, lives in a session corresponding to a single request, and contains no state to be shared among several sessions. Therefore, no mutually exclusion is needed.

As similar to file systems, we have built a filter HTTP object that invokes external filter programs.

In the current implementation of virtual stackable HTTP objects, caching is not used. Caching is one of the main techniques which are used to implement stackable file objects [10]. The reason of the use of caching is that each stackable file object has to deal with random access from clients. The cache for an entire file is created to realize non-byte filtering. In W3, in contrast, a virtual stackable HTTP object is read from the beginning to the end sequentially. Therefore, it is not needed to calculate positions to read lower objects.

### 5.2 Pointer Swizzling

We provide the following library procedures for supporting pointer swizzling described in Section 4.2.2:

(1) swizzling_prefix_make(): This procedure makes two prefix strings for swizzling. One is for the relative form of URL, and the other is for the absolute form of URL. This procedure takes the URL of the target virtual stackable object and the URL of the lower object.

(2) line_swizzle(): This procedure performs pointer swizzling in a line. This procedure works as a state transition machine.

### 5.3 Performance

We measured the performance of the stackable image conversion server, described in Section 3.2. Experimental results are shown in Table 1. In this experiment, original image data is managed by a regular HTTP server (CERN httpd 3.0pre6, libwww 2.16). The stackable image conversion server is implemented by the generic UNIX filter server. This server invokes the `convert` command in ImageMagic with options `"-scale 0.5x0.5 -monochrome"`. The regular HTTP server and the stackable image conversion server run on different machines (Sun SPARCstation Classic Compatible, 50MHz MicroSPARC I, main memory 16M Bytes).

In Table 1, `http-cat` is a simple HTTP client that takes an URL, gets a corresponding HTTP object from a server, and writes the contents to the standard output. We measured the execution times of the `http-cat` command by the time command of tcsh. The `http-cat` command is executed on another Sun Classic. `MacMosaic` of NCSA is executed on Macintosh Duo 280 (CPU 68040 33MHz without FPU, main memory 12Mbytes) with DuoDock II. We measured the execution times of reloading of image data by a stopwatch. All machines are connected with Ethernet (10M bps).

In this experiment, we use color image data which size is 230 k bytes. This image data is reduced by the `convert` command into 5 k bytes. This conversion takes 2.5 seconds by the `convert` command if it is invoked by the UNIX shell.

In Table 1, from the results of execution times of the `http-cat`, the overhead of the stackable image conversion server is about 1.6 seconds. This time is shorter than the execution time of the `convert` command. One reason for this result is that parallel processing works well among the three machines.

In the case of `MacMosaic`, although all machines are connected with a high-speed local area network, the effect of image conversion is large. This is because in this case, the reloading operation is bounded by the CPU that executes the client application, and the execution time of redisplaying for the small monochrome image data is much shorter than that for the large color image data.

Table 1: Performance of stackable image conversion server (in seconds).

| | http-cat on Sun (execution time) | MacMosaic (reloading time) |
|---|---|---|
| Original (230k bytes, color) | 1.7 | 18 |
| Filtered (5k bytes, monochrome) | 3.3 | 5 |

# 6 Related work

### 6.1 CGI (Common Gateway Interface)

A regular W3 server provides a facility, called *CGI* (*Common Gateway Interface*) which allows information providers to extend the functions of the server [1]. The term "*gateway*" means the entrance to the HTTP protocol from the other protocols.

Our stackable HTTP object that invokes an external program is similar to CGI. As compared with CGI, the main feature of our stackable HTTP object is that it must take a lower object. Our stackable HTTP object reads the contents of another HTTP object. A regular CGI program does not read another HTTP object. This means that CGI is not stackable. An HTTP object created by CGI can be used as a bottom object of our system.

### 6.2 DeleGate

DeleGate [8] is an HTTP server that provides the services of crossing firewall and caching, as similar to W3-Proxy described in Section 3.3 and Section 4.3. Unlike W3-Proxy, DeleGate can change the contents of relaying HTTP objects according to directives embedded in URLs. Besides HTTP, DeleGate supports other protocols, such as Gopher, FTP, and NNTP.

As similar to our pointer swizzling, DeleGate has a facility to rewrite URLs in response messages. Therefore, DeleGate can provide the services of crossing firewall and caching for pre-proxy clients.

DeleGate supports only conversion of text data. One of our stackable servers supports conversion of image data. For text data, one of our stackable servers invokes external filter programs and provides their filtering functions. Moreover, our stackable servers are designed as tools which can be combined.

## 7 Conclusion

In this paper, we have described object-stacking and sever-stacking in the World-Wide Web. Object-stacking and sever-stacking are models for structuring object-based systems. One of their main features is that layers of objects with a uniform interface are constructed, and the functions of these objects are integrated. In the World-Wide Web, object-stacking and server-stacking provide high-level functions, such as conversion of image, translation of character encoding methods, macro processing, conversion of media, and any filtering functions provided by filter commands in UNIX.

We have shown the implement method of object-stacking in the World-Wide Web. We have introduced the new concepts of demand stacking, virtual stackable objects, and pointer swizzling. We have implemented several stackable servers in the World-Wide Web. We made an experiment to measure performance of a stackable image conversion server. In our future work, we would like to propose a mechanism based on object-stacking and object-cascading for maintaining cache consistency in the World-Wide Web.

## References


[1] T.Berners-Lee, Robert Cailliau, A.Luotonen, H.Frystky Nielsen, and A.Secret: "The World-Wide Web", Communications of the ACM, Vol.37, No.8, pp.76-82 (1994). <http://www.w3.org/hypertext/WWW/TheProject.html>

[2] R.G.G.Cattell: "Object Data Management", Addison-Wesley (1991).

[3] R.G.Guy, J.S.Heidemann, Wai Mak, T.W.Page,Jr., G.J.Popek and D.Rothmeier: "Implementation of the Ficus replicated file system", USENIX 1990 Summer Conf., pp.63-71 (1990).

[4] Y.Khalidi and M.Nelson: "Extensible File Systems in Spring", 14th SOSP, ACM Operating System Review, Vol.27, No.5, pp.1-14 (1993).

[5] H.Lieberman: "Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems", OOPSLA'86, pp.214-223 (1986).

[6] A.Luotonenand, and K.Altis: "World-Wide Web Proxies", First International Conference on the World-Wide Web (WWW'94) (1994). <http://www.w3.org/hypertext/WWW/Proxies/>

[7] J.Murai, M.Crispinand, and E.van der Poel: "Japanese Character Encoding for Internet Messages", RFC1468 (1993). <ftp://ds.internic.net/rfc/rfc1468.txt>

[8] Y.Sato: "Development of a Protocol Mediation System DeleGate", ETL Technical Report (1994). <http://www.etl.go.jp/People/ysato/>

[9] Y.Shinjo: "The Object-Stacking and Object-Cascading Models", 6th IPSJ Computer System Symposium, Vol.94, No.10, pp.31-38 (1994).

[10] Y.Shinjo and Y.Kiyoki: "The Object-Stacking Model for Structuring Object-Based Systems", Proc. 2nd International Workshop on Object Orientation in Operating Systems (I-WOOOS'92), pp.328-340 (1992). <http://bottom.ie.u-ryukyu.ac.jp/~yas/>

[11] Y.Shinjo and Y.Kiyoki: "Multiple Views of the ReSC Distributed Operating System", Workshop on Object Orientation in Operating Systems at OOPSLA/ECOOP'90 (1990).

[12] A.S.Tanenbaum and R.Renesse, "Distributed Operating Systems", ACM Computing Surveys, Vol.17, No.4, pp.419-470 (1985).

[13] D.Ungar and Randall B.Smith: "Self: The Power of Simplicity", OOPSLA'87, pp.227-242 (1987).

[14] P.R.Willson and S.V.Kakkad: "Pointer Swizzling at Page Fault Time: Efficiency and Compatibly Supporting Huge Address Spaces on Standard Hardware", Proc. 2nd International Workshop on Object Orientation in Operating Systems (I-WOOOS'92), pp.364-377 (1992).