

# Encapsulation of the GIOP/IOP protocol in HTTP

Éric Dumas (dumas@tumbleweed.com, dumas@Linux.EU.Org)

Randy Shoup (rshoup@tumbleweed.com)

Tumbleweed Software, Redwood City, California

May 25, 1999

## 1 Introduction

This document describes the encapsulation of the GIOP protocol in the HTTP, the modifications performed in the omniORB code, the way to compile, to configure and to use it.

## 2 How is it working?

### 2.1 HTTP: a different protocol

The HTTP protocol is different of the TCP/UDP network protocol because, if you can perform asynchronous read()/write() operations using a TCP/UDP socket, you must do synchronous read()/write() for a HTTP connection. This way, it has been decided to implement a “proxy/relay” in the ORB, created as a separate thread (but you could really extract it as a separate process if you wish, and for example install it on a firewall).

### 2.2 Architecture

When data have to be sent from the client, the tcpSocketMTfactory module is involved. The implementation modify slightly the way the port number and the port are affected. In the case a HTTP connection is requiered, the destination port and IP address are modified, and a handshake packet containing the real destination ip address and port number is sent before any GIOP data is written. The following schema represent the thread actions and the various sockets connections.

```

IOP convertor HTTP

----- relay! -----<--> T1 <--->HttpConnection<-->
realConnect()| -----> |ThreadProxy      |<--> T2 <--->HttpConnection<-->
-----<--> |bind()/accept()|<--> T3 <--->HttpConnection<-->
          | -----> ... <--->HttpConnection<-->
          |          Tn <--->HttpConnection<-->
          |
          v no relaying.

```

The ThreadProxy is created only once, and will be present during all the process life. The threads  $T_1$  ( $T_2, \dots, T_n$ ) are created by the ThreadProxy once the accept() system call has returned a socket filedescriptor. It's going to read all the informations coming the ORB, *re-assembling a full GIOP packet* if necessary, and send everything as a POST HTTP operation. This thread will wait for the remote ORB answer (reply of the POST) and write the answer back in the socket.

The reason why the SocketThread assemble a full GIOP packet are:

- the ORB perform some strand stuffs we do not need in HTTP,
- if we send chunks of GIOP data, we will not get an answer on the other side of the network link, which is a big problem.
- it prevents us to keep a state on the HTTP server side, which simplify a lot the implementation.

### 2.3 How to decide to relay?

The fact to decide if we have to relay or not is decided depending of new configuration instructions in the `omniORB.cfg` file. The use of an extra *IP Mapping File* can be used too (see later) to give more flexibility to the configuration (and be able to use multiple HTTP gateways at the same time).

## 3 What has been changed, added in the standard code?

### 3.1 Modification of the existing code

All the modifications have been performed in the libomniORB2 library :

- `initFile.cc` : add a call to the IOP\_HTTP module in order to read the new config file items,
- `tcpSocketMTfactory.cc` : in the `realConnect()` method, change the destination of the socket if necessary and send the handshake,

One modification in the Gatekeeper : a new method has been added, in order to perform the security check based on a socket file descriptor instead of a tcpStrand.

### 3.2 New code

Two new files in the omniORB2 library:

- `IOP_HTTP.cc` (*new file*) : implementation of IOP over HTTP,
- `ip_mapping.cc` (*new file*) : parsing of the `ip_mapping` file.

A new library, `HttpConnection` has been added. In fact, we provide two implementations:

- `dummystub` : dummy library,
- `GPL` : a GPL version of the library. In fact, we are using our own library we cannot release to the public. However, we wrote a GPL version for the omniORB (only Unix). Feel free to improve it<sup>1</sup>!

### 3.3 Extra-Headers

An External directory has been added, in which you will find the generic interfaces for `Http` and `SSL`. Both `GPLHttpConnection` and the `Dummy` version implement the `HttpConnection` generic interface.

Some headers have been added in `include/IOP_HTTP`. They will be used in the HTTP server side in order to decode the handshake.

## 4 How to build it?

If you wish to compile omniORB with the HTTP layer, you need to define the flag `IOP_HTTP` in the `dir.mk` of the omniORB2 library (and the one located in the `sharedlib` too!). If you have a `SSL` layer<sup>2</sup> too, you need to define the flag `HAS_SSL` too.

Nothing different is required in order to compile it. When generating your binary, this to add the `HttpConnection` library.

---

<sup>1</sup>A more generic implementation could use the W3C library, <http://www.w3c.org>.

<sup>2</sup>Only the generic interface is provided here. Because of the various exportation/important rules, it sounds difficult to release this code. Have a look to `OpenSSL`, <http://www.openssl.org>.

## 5 How to configure it?

### 5.1 ProxyThread

First, let's configure the ProxyThread, by adding the following line in the `omniORB.cfg` file:

```
IOP_HTTP_BIND_PORT 5678
```

5678 is the port number the ProxyThread will be binded. If this port is already binded, it will try to use a greater port number.

### 5.2 Simple configuration: forward everything

If you wish to forward everything to a single host, just add:

```
IOP_HTTP_PORT      81
IOP_HTTP_PATH      /remoteOrb
IOP_HTTP_MIME_TYPE x-application/remoteOrb-gateway
IOP_HTTP_HOST      foo.host.org
```

- IOP\_HTTP\_PORT: HTTP port number (generally 80),
- IOP\_HTTP\_PATH: path to specify on the URL,
- IOP\_HTTP\_MIME\_TYPE: Mime-Type to specify,
- IOP\_HTTP\_HOST: Hostname of the Web server to reach.

### 5.3 Proxy configuration

If you have to go through a proxy, you can add the following parameters:

```
IOP_HTTP_REMOTE_PROXY_HOSTNAME proxy.foo.bar
IOP_HTTP_REMOTE_PROXY_PORT     8080
IOP_HTTP_REMOTE_PROXY_LOGIN    mylogin
IOP_HTTP_REMOTE_PROXY_PASSWORD mypassword
```

### 5.4 If you have SSL

Add the following parameters (the first one is only if you are using the forwarding feature, not the IP mapping).

```
IOP_HTTPS_PORT 443
IOP_HTTP_SSL_POLICY 2
IOP_HTTP_SSL_ROOTCERT_FILE /etc/trusted.txt
```

The `IOP_HTTP_SSL_POLICY` defines if HTTP has to use SSL or not. If you do not have the SSL layer, use 2, else 1.

## 5.5 File IP Mapping Feature

You can specify a filename in which are stored some “rules” in order to decide if you need to forward or not data.

```
IOP_HTTP_IP_MAPPING_FILENAME /etc/orbIpMapping.txt
```

Here is an example of this file:

```
#
# IP adress  Web site  HTTP HTTPS Path  Mime-Type      SSLStatus
#
192.240.3.2    www.a.com 80   443  /orb  x-application/orb  1
192.240.3.3    www2.a.com 80   443  /orb  x-application/orb  2
65.*.*.*      www.c.com 81   446  /orb1 x-application/orb  1
147.210.53.433 www.c.com 81   456  /orb2 x-application/orb  2
```

For example, if you want to contact an ORB component located on the 192.240.3.2 machine, HTTP packets will go through the `www.a.com` Web site. Data sent to 192.240.3.3 will be sent through `www2.a.com`.

## 6 Debugging

If you want to see if the HTTP encapsulation is working, use the commande line option `-ORBtraceLevel 5`.

## 7 Web Server Side

In our implementation, we are using `NetscapeServer`. You could very easily implement the module using the `Apache`<sup>3</sup> module feature. However, you will find in this directory an example of what has to be done in the server module.

---

<sup>3</sup><http://www.apache.org>

## **8 We want to hear you!**

This code has been released under GPL, so if you find any bug, or make any improvement, we would really appreciate to receive them or to be contacted!

Thank you.