XML Configuration Access Protocol (XCAP)

Madhuri Shinde
mss2103@columbia.edu

Introduction

As the name suggests, XCAP is a configuration access protocol. It is used to store application configuration data, in XML format and allows clients to read, write as well as modify the same configuration data.

XCAP maps XML document sub-trees and element attributes to HTTP URLs, so that these components can be directly accessed by HTTP. XCAP is a set of conventions for mapping XML documents and document components into HTTP URLs, rules for how the modification of one resource affects another, data validation constraints, and authorization policies associated with access to those resources. Because of this structure, normal HTTP primitives can be used to manipulate the data. XCAP is based heavily on ideas borrowed from the Application Configuration Access Protocol (ACAP), but it is not an extension of it, nor does it have any dependencies on it. Like ACAP, XCAP is meant to support the configuration needs for a multiplicity of applications, rather than just a single one.

Applications

In many communications applications, such as Voice over IP, instant messaging, and presence, it is necessary for network servers to access per-user information in the process of servicing a request. This per-user information resides within the network, but is managed by the end user themselves. Its management can be done through a multiplicity of access points, including the web, a wireless handset, or a PC application. Examples of per-user information are presence, authorization policy and presence lists. Presence lists are lists of users whose presence is desired by a watcher. One way to obtain presence information for the list is to subscribe to a resource which represents that list . In this case, the Resource List Server (RLS) requires access to this list in order to process a SIP SUBSCRIBE request for it. Another way to obtain presence for the users on the list is for a watcher to subscribe to each user individually. In this case, it is convenient to have a server store the list, and when the client boots, it fetches the list from the server. This would allow a user to access their resource lists from different clients.

To explain this in more detail -

SIMPLE specifications allow (SIP for Instant Messaging and Presence) allow a user (watcher) to subscribe to another user (presentity) in order to learn their presence information. In many cases a watcher would be interested in a list of presentities that is the presence list.

When wanting to subscribe to a presence list, standard procedures require watcher to create and manage subscription for each presentity in the list. For large lists the bandwidth needed for could be large particularly in case of wireless networks. An extension to SIP events framework allows a watcher to subscribe to list of resources using single subscription. Such a subscription requires a server called the Resource List Server that has a copy of the presence list that the user wishes to subscribe to. By using XCAP a client can place the presence list on the server and manipulate it as required.

Overview of Operation

Each application that makes use of XCAP specifies an application usage. This application usage defines the XML schema for the data used by the application, along with other key pieces of information. The principal task of XCAP is to allow clients to read, write, modify, create and delete pieces of that data. These operations are supported using HTTP 1.1.

An XCAP server acts as a repository for collections of XML documents. There will be documents stored for each application. Within each application, there are documents stored for each user. Each user can have a multiplicity of documents for a particular application. To access some component of one of those documents, XCAP defines an algorithm for constructing a URL that can be used to reference that component. Components refer to any element or attribute within the document. Thus, the HTTP URLs used by XCAP point to a document, or to pieces of information that are finer grained than the XML document itself. An HTTP resource which follows the naming conventions and validation constraints defined here is called an XCAP resource.

Since XCAP resources are also HTTP resources, they can be accessed using HTTP methods. Reading an XCAP resource is accomplished with HTTP GET, creating or modifying one is done with HTTP PUT, and removing one of the resources is done with an HTTP DELETE. Properties that HTTP associates with resources, such as entity tags, also apply to XCAP resources. Entity tags are particularly useful in XCAP, as they allow a number of conditional operations to be performed.

An XCAP resource is nothing but an XML document, an element within the XML document, or an attribute corresponding to an element. Each of the HTTP GET, PUT, DELETE methods can be executed on any resource belonging to a user within a particular application.

Mapping of XCAP resource to HTTP URI can be done as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xcap-caps>
  <users>
    <user1>hgs@cs.columbia.edu</user1>
    <user2>mss2103@cs.columbia.edu</user2>
  </users>
</xcap-caps>
```

The HTTP URI used to access <user1> would look as follows:

Part1	Part2	Part3
(HTTP Method)	(XCAP Root/Application(k)/user(m)/document(i)/~~/xcap-caps/users/user1)	HTTP(v)

As shown above, each request to a particular resource can be divided into three parts as shown above.

Part 1 – The HTTP method

The only methods allowed with XCAP are GET, PUT, DELETE.

Part 2 - The actual path to the resource that is traced by following the XML parent – child structure.

The tracing of the path is explained as follows:

The server retrieves the application that is being accessed by user by following XCAP Root/Application(k). As mentioned earlier, the each application and users within each application have their own directories. So the whole system is stored in a hierarchical format with the XCAP ROOT being at the top. The applications form children of this root. Within each application there are users which form child directories of the main directory corresponding to that application. Within each users directory there are XML files corresponding to that user. Within each XML file there are elements starting with the root element. Within each element there are attributes.

Thus the path above tries to access Application “k” within XCAPServer, user “m” within application k, XML document “i” corresponding to user “m”. The path followed by the server so far is called the **document selector** and as the name suggests, it access a document corresponding to a user. The “~~” is called the path separator – it separates the document selector from the node selector. Node selector is the path followed by the server after path separator. So as shown in the above request the node selector here is **/xcap-caps/users/user1.** This tells the server to start from root element of document “i”. It shoud be **<xcap-caps>**. Then within **<xcap-caps>** find element **<users>**. Then within **<users>** find **<user1>**. Once this element <user1> is obtained, operations are performed on it according to the method suggested (GET, PUT, DELETE).

Part 3 – The HTTP version

The HTTP version being used. (v) here corresponds to version, it could be 1.0 or 1.1.

For further details please refer to

http://www.ietf.org/internet-drafts/draft-ietf-simple-xcap-05.txt as well as sample request and the screen shots.

Goal of this Project

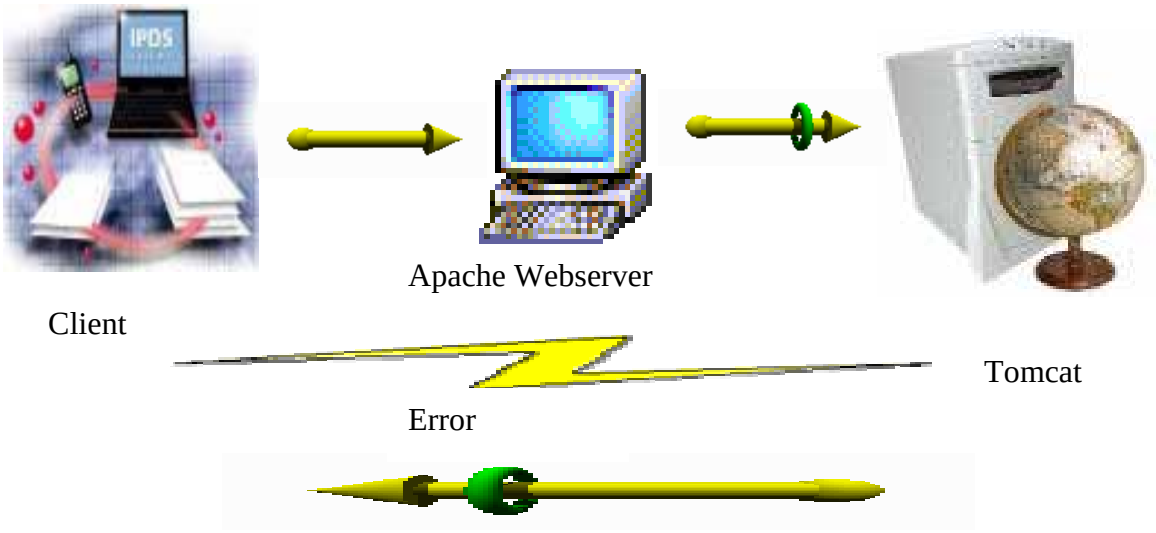
The goal of this project was to implement the XCAPServer that provides the functionalities listed below.

Server Functions

The server is required to perform functions in the following order. The order might change at times, depending on situations:

1. The server needs to check for syntax errors i.e. check if the request if formed in the right way and follows the required syntax described above. This can be done at the client side too, in order to avoid loading the server. If done at the client side, this function is performed first.
2. The specification requires the server to perform digest authentication. If the user is authenticated, then his request is carried out by the server.
3. Once authenticated, the server splits the request into various parts in order to determine the application, user, document and so on. While searching for a document using the document selector, if some component is not found, the server needs to reply with an error message. Similarly once a required document is found, further operations need to be performed. The GET, PUT, DELETE can be at document level in which case the node selector would be null. Else further parsing is needed to go upto the required node level.
4. Send the required response in the right format. Note that all the responses as well as error messages need to be in the standard HTTP format.
5. There are other features to be handled while processing requests, like checking for conditionals, e-tags, headers like content-type which are similar to processing HTTP requests.

Implementation Details



This XCAP Server was implemented using http://www.ietf.org/internet-drafts/draft-ietf-simple-xcap-03.txt that is version three of the draft released in July, 2004.

As shown in the diagram above the server was implemented using Apache Tomcat which served as a repository for the servlet that handled the entire set of functions mentioned above for the XCAP Server. A simple HTTP client available on Internet was extended for required functionality of XCAP client by GUI implementation using java Swing package. Also as shown in the diagram Apache web server sits between the client as well as Tomcat. The server is needed because Tomcat does not support digest authentication. Hence the authorization is handled by Apache server using the default .htaccess file available in Apache. If a user is in the database, he is authorized by Apache when he provides the correct authentication information irrespective of the resource that he requests. The request is then forwarded to Tomcat which handles the request using the XCAPServer servlet. Apache and Tomcat are connected by the mod_jk2 module. Access permissions are checked by Tomcat to ensure that user is trying to access a resource in his own directory. In case the user does not have access permissions to that resource, appropriate error messages are sent.

Once it is ensured that the user is accessing the permitted resources, the user request is parsed. The required document is retrieved. After that schema validation is handled. Schema validation is nothing but checking if the result of the desired request would be valid against that application's schema. Each application may choose to follow a schema which is stored as a special file in that applications directory. Also xml well formedness and data validations if any are checked. The HTTP headers like content type and conditional modifiers with etags (if any) are checked and the request is processed only if needed and if these headers are appropriate for the request. For example in case of PUT request for a document in the presence application the content type has to be application/presence+xml. Similarly if there are conditionals like if-match, if-none-match, the MD-5 digest of the document is computed and compared against the etag passed via the header. The request is carried out only if the etags match or don't match depending on the type of header (Refer to RFC 2616)

If the request needs to be processed and all permissions checks are good, the document parsing is done using DOM Parser as the documents are relatively small in size. While parsing the document, if any particular resource on the path is not found an appropriate error message is returned.

Finally, the response is sent with the appropriate headers and error messages if any.

There are a few screen shots obtained by running a few simple requests on the server at the end of the report.

Since all the screen shots could not be accommodated, here are a few types of request that can be made.

GET, PUT and DELETE requests for

1. File
2. Element
3. Attribute

A file can be searched according to an application, user as mentioned earlier. An element within a file can be searched by position or attribute or by position and attribute both (as elements could have same names and attributes).

A few sample requests are as follows:

PUT

http://warsaw.clic.cs.columbia.edu:8080/XCAPServer/servlet/edu.columbia.xcap.XCAPServer/presence/global/temp.xml/~~/presence/users/ HTTP/1.1

<users><user id="mss2103">Madhuri Shinde</user></users>

Content-type: application/xcap-el+xml

GET

`http://warsaw.clic.cs.columbia.edu:8080/XCAPServer/servlet/edu.columbia.xcap.XCAPServer/presence/global/~~/presence/users/user[1]/@id HTTP/1.1`

This application was tested by a few participants at the SIPIT 2004 and the results were found to be satisfactory.

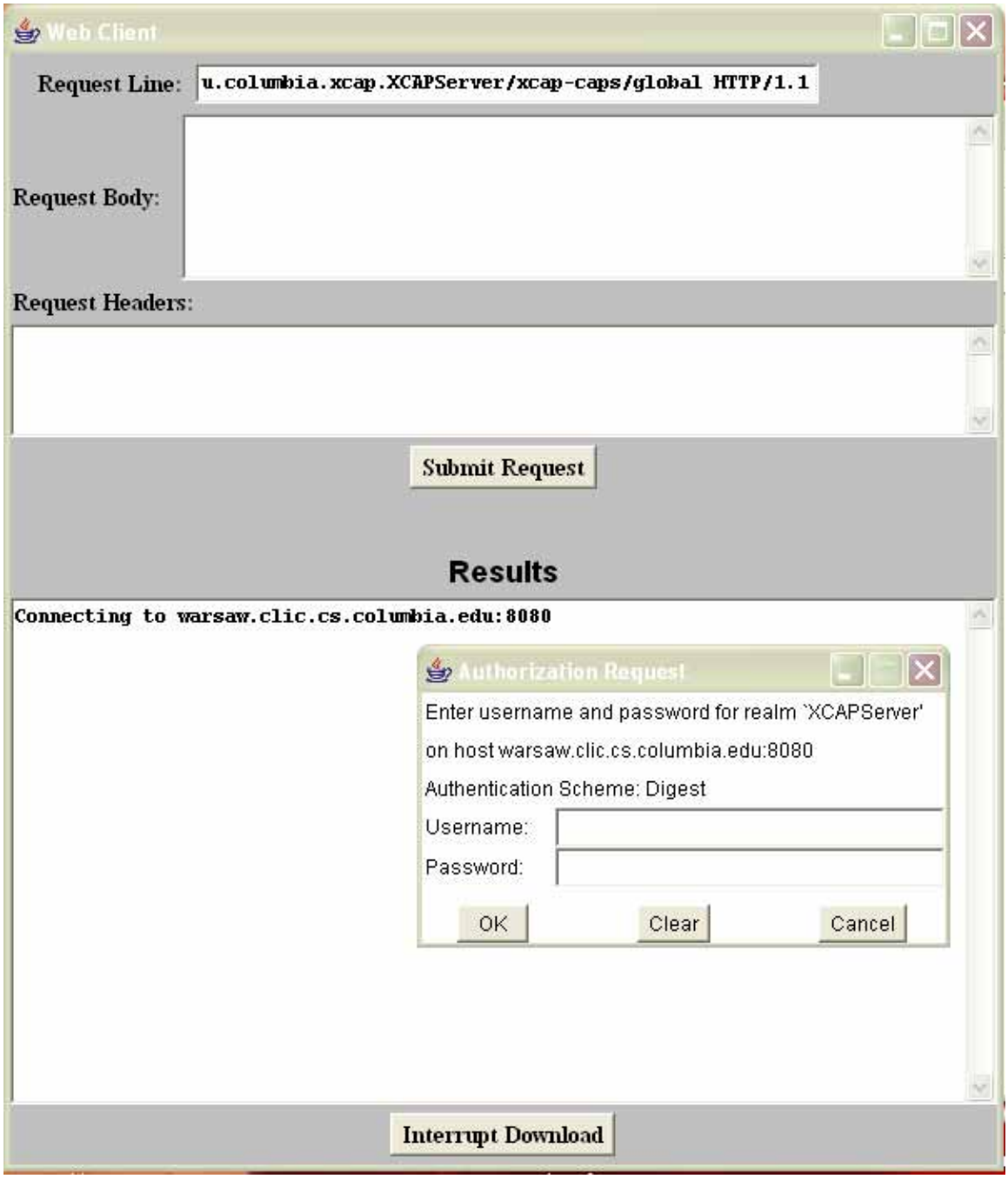


Figure 1: The XCAP Client sending a GET request. Apache Web Server requests authorization information.

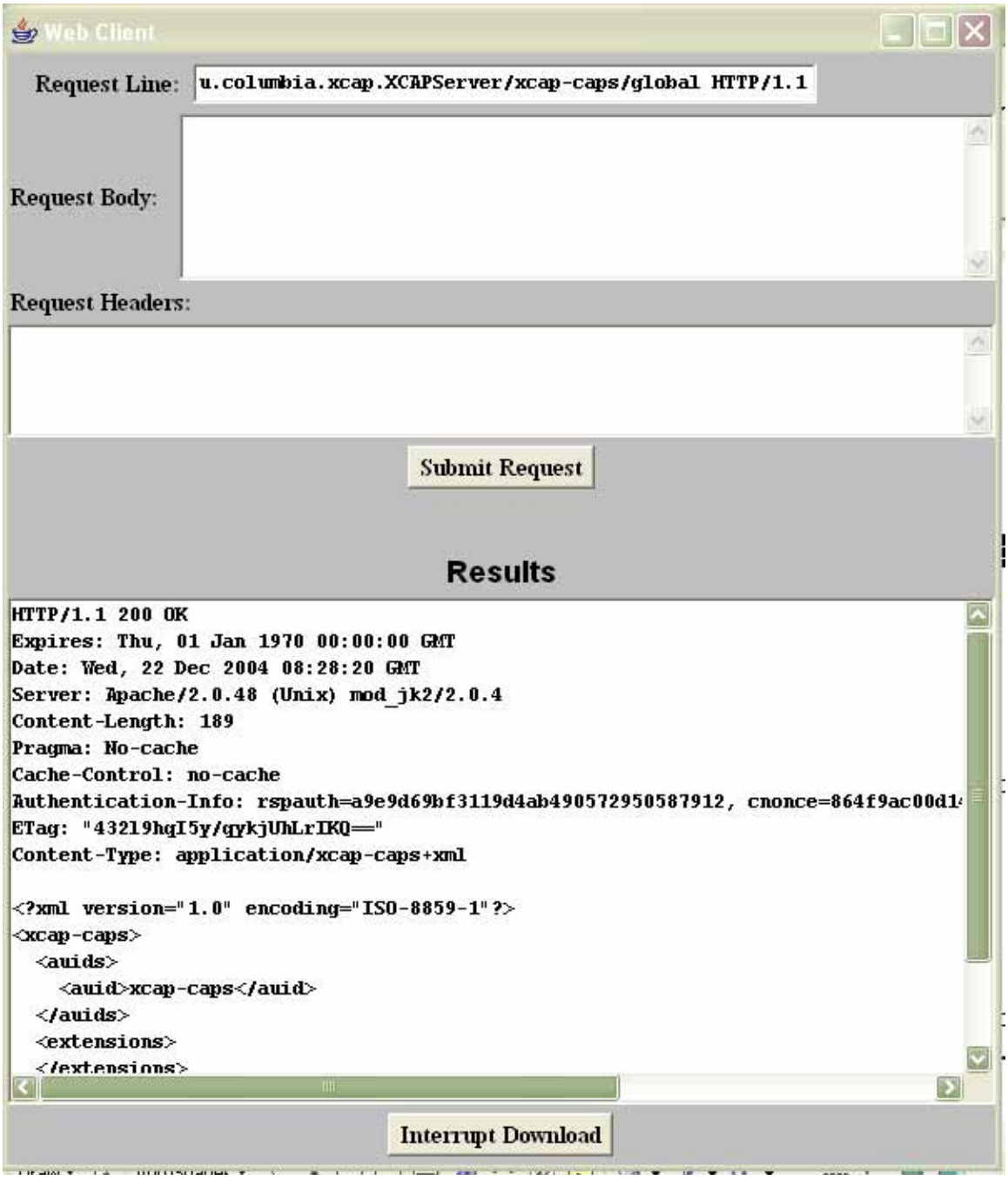


Figure 2: Results of the GET request after client sends authorization information

Figure 3: PUT request resulting into error as XML document was not sent

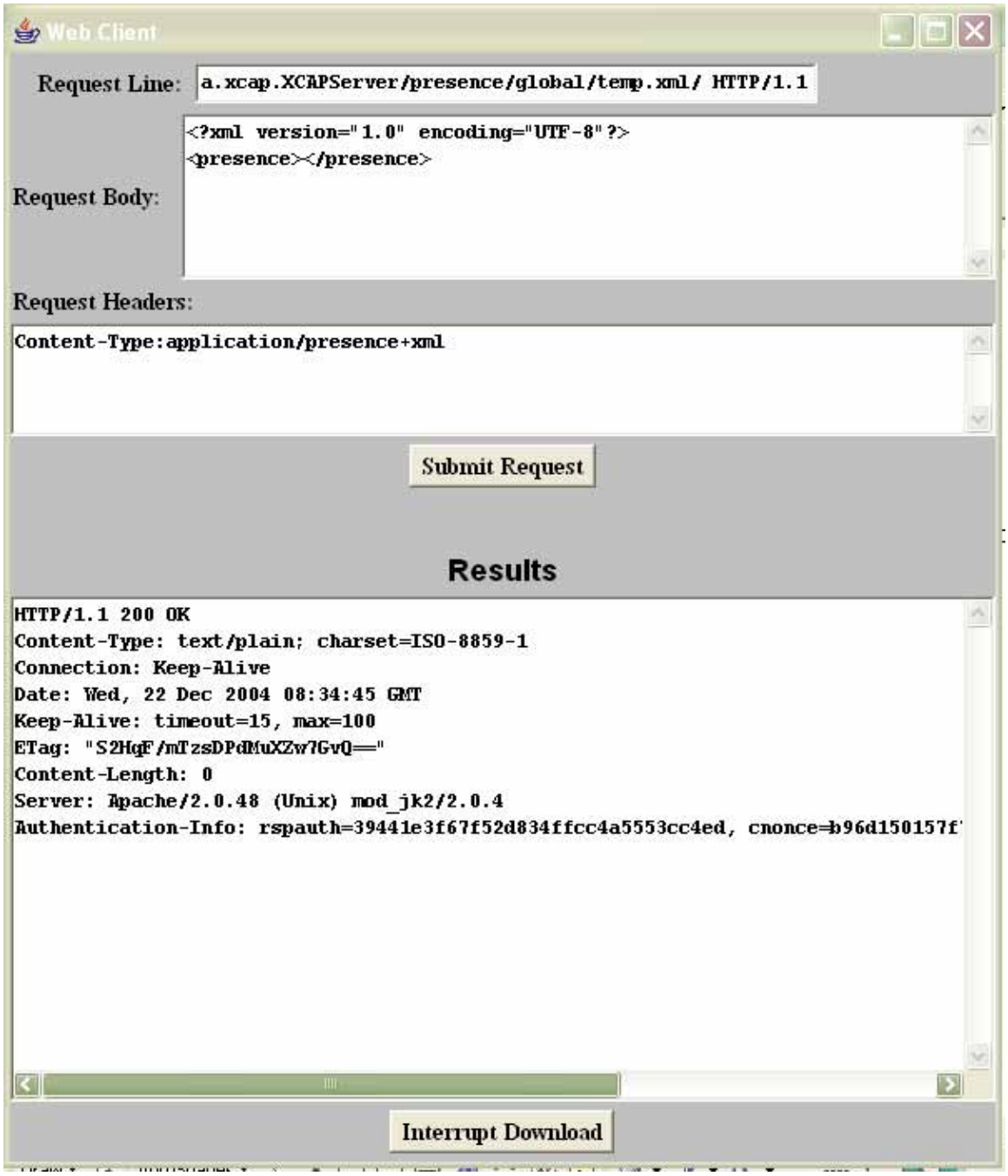


Figure 4: PUT request succeeded. Please note the content type as well as the ETag in the response that can

Figure 5: Same request tried after sending the If-Match header along with etag succeeds as Etag matches