Migrating Web Application Sessions in Mobile Computing

G. Canfora°, G. Di Santo°*, G. Venturi°*, E. Zimeo°, M. V. Zito°
*Department of Engineering – °RCOST - University of Sannio
via Traiano, 82100 – Benevento, Italy
+39 0824 305555
{canfora, giuseppe.disanto, venturi, zimeo, mariavittoria.zito}@unisannio.it

ABSTRACT
The capability to change user agent while working is starting to appear in state of the art mobile computing due to the proliferation of different kinds of devices, ranging from personal wireless devices to desktop computers, and to the consequent necessity of migrating working sessions from a device to a more apt one. Research results related to the hand-off at low level are not sufficient to solve the problem at application level. The paper presents a scheme for session hand-off in Web applications which, by exploiting a proxy-based architecture, is able to work without interventions on existing code.

Categories and Subject Descriptors
C.2.4 [**Distributed Systems**]: Client/server, Distributed applications.

General Terms:
Algorithms, Performance, Design.

Keywords:
Web applications, session hand-off, mobile computing.

1. INTRODUCTION
The capability to change the whole host used by a user may become a very central issue in next generation networked services. While current devices have a great deal of partially overlapping capabilities (mobile phones can browse the Web, PCs can act as phones, and so on), the dream of a unique, really multipurpose, device remains very far. In real, everyday, life many people use a steeply increasing number of devices because their capabilities are very tailored to the different purposes. Moreover, communication sessions are taking an increasing time in our life as a whole and are becoming longer taken singularly. This observation leads to two corollaries: (a) it is highly probable for a communication session to interfere with an activity requiring movement; (b) often is unforeseeable what exact kind of device we will find more suitable in the following of a session.

Some examples can illustrate the intermingling of the above considerations. A driver can set a path using his car navigation system to discover later that the last part of the path is in a pedestrian zone, in which he can use the cell phone but not the car navigator; an Internet user can start navigating from the PDA and needs to continue the navigation on a PC if he discovers that a clip needs a wider screen.

WWW 2005, May 10-14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

All these scenarios involve the capability to carry a session from a host to another. The paper investigates the capabilities of current and near future technologies to give answer to such scenarios, to assess user needs and technology limitations and to propose and experiment with possible solutions in the context of legacy applications. Enabling session hand-off in legacy applications allows them to compete with more modern ones due to the possibility of increasing their usability. However, legacy code owners are often afraid to incremental maintenance projects. Thus, we present an architecture and a protocol capable of enabling legacy applications to exploit a session hand-off service supplied with minimal interventions on the application.

The following sections talk about a proxy-based architecture and the proxy functionalities. Finally, we describe our implementation effort, which is currently in progress.

2. PROXY-BASED SESSION HAND-OFF
Some proposals centered on a client-based architectural scheme are presented in the literature [4]. With this approach, the code responsible to track and store session information (the session hand-off component SHOC) is close to the client.

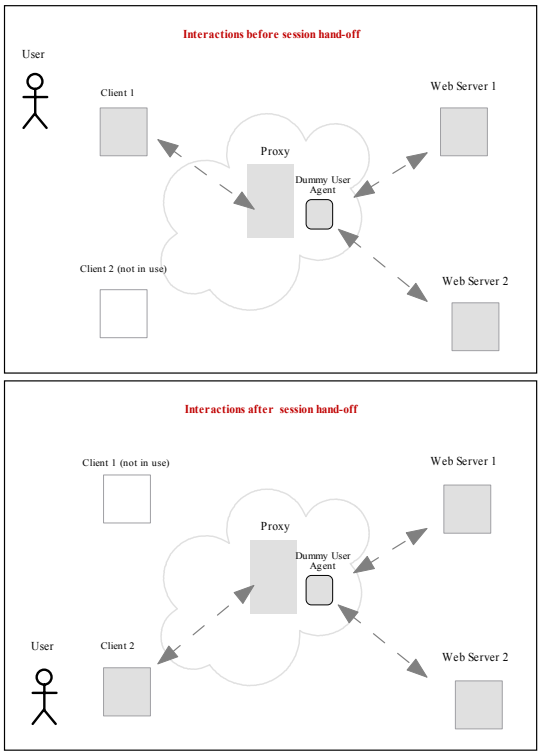

Figure 1: Session hand-off with a dummy user agent.

The proxy-based architectural scheme, instead, involves the use of an intermediate proxy, the SHOC, between client and servers. Since the interactions between client and servers cross the proxy, it is able to store all the session information even in the case in which several servers are employed for the application.

In this architecture, when a client issues a hand-off, the target client can retrieve all the information it needs from the proxy.

The proxy-based technology allows for adding the service without any intervention on both client and servers. Therefore, it can be used on a large array of existing applications.

Although HTTP is a session-less protocol, web applications based on HTTP are expected to maintain a state for a group of interactions between client and servers. The state can be handled by the server, by the client or by both. RFC-2965 [2] has introduced the Cookie technology which directly addresses session management by storing information at client side. Other session related information can be carried by requests and responses in authentication and user agent identification headers.

By adopting a proxy-based architecture, servers see a surrogate (dummy user agent) of the client on the proxy even when the client migrates from a device to another.

Figure 1 depicts a session hand-off in which the same dummy user agent is contacted by servers before and after the hand-off. To keep consistent the communication between client and server, the dummy user agent must be capable to store some data exchanged between client and server and to carry on the authentication mechanism of RFC-2617 [1] and part of the cookie management of RFC-2965.

In particular, the proxy must implement the following activities:

- *User authentication*: the proxy needs the authentication of each user in order to generate a specific dummy user agent.
- *Server authentication*: whenever a server requires an authentication, the proxy must authenticate the dummy user agent instead of the real one.
- *Cache management*: information exchanged between client and servers must be stored notwithstanding proxy caching directives and rules.
- *Cookies management*: the proxy must store cookies in order to restore them in a new user agent after a hand-off.

In our model, the hand-off happens when a new user agent authenticates itself on the proxy with any currently in use account. Since authentication data is owned by the proxy it is quite simple to re-enter authenticated sessions. However, the web session can not be completely restored since it is not aware of cookies previously exchanged by the other agent. Restoring cookies on the new client is vital for a seamless migration of the session.

3. IMPLEMENTATION

The proxy presented in this paper, called *MuffinSH*, was implemented in Java. It was built by extending a filtering system, *Muffin* [3] freely available under GNU General Public License, and supports HTTP/0.9, HTTP/1.0, HTTP/1.1 and SSL.

MuffinSH uses the filters inherited from the Muffin interfaces to implement the session hand-off. The main interfaces implemented by MuffinSH are: *RequestFilter*, *ReplyFilter*, *ContentFilter*, *HttpFilter*, and *RedirectFilter*.

The RequestFilter interface is implemented by the filters that handle request headers; ReplyFilter is used for response headers; ContentFilter processes the content of responses; HttpFilter is used to generate replies to requests that do not need of being sent to the server; RedirectFilter redirects an URL to another. By using the filters, the proxy is able to store, for each user and each domain, web content replied by a server to a client before the user hands-off the session.

The above interfaces were implemented by the following filters:

- *ProxyAuthFilter*: implements *HttpFilter*; the filter verifies whether all attained requests contain the "Proxy-Authorization" header. If it is not present or username and password are not correct, the filter requires a Proxy-Authorization with the basic authentication scheme and does not forward the request to the server. If user credentials are correct, the filter only verifies whether the authentication is issued before or after a hand-off.
- *CachedReplyFilter:* implements *HttpFilter*; when a user issues a request after a hand-off, the filter verifies whether the domain of the requested resource exists in the proxy cache and, in positive case, it creates a response to send the client the resource stored in cache.
- *ServerAuthFilter:* implements *RequestFilter* and *ReplyFilter*; it acts when a server asks the user for the authentication.
- *HeaderChangeFilter:* implements *RequestFilter*; it changes the request headers to allow the server to recognize the client with a new set of headers representing a dummy user agent.
- *CacheFilter:* implements *ContentFilter*; before a hand-off occurs, this filter acts on the content of responses characterized by a text/html content-type; it analyzes the received html tags and basing on this analysis manages the pages containing frames, the pages implementing the contained frames and the pages referred inside the frames.
- *UrlRedirectFilter:* implements *RedirectFilter*; when this filter intercepts in the request URL some identifiers previously inserted by CacheFilter, it eliminates the identifiers, stores the related information and forwards the request to the server.
- *RedFilter:* implements *RedirectFilter*; it works after the occurrence of a hand-off to perform a redirection to the URL requested from the client before the hand-off in order to re-collect all the cookies previously received by the client in response to that request.

4. REFERENCES

[1] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L. HTTP Authentication: Basic and Digest Access Authentication, RFC 2617, June 1999. http://www.rfc-editor.org/rfc/rfc2617.txt.

[2] Kristol, D., and Montulli, L. HTTP State Management Mechanism, RFC 2965, Octber 2000. http://www.rfc-editor.org/rfc/rfc2965.txt.

[3] Muffin: World Wide Web Filtering System. http://muffin.doit.org.

[4] Song, H., Chu, H., Kurakake, S. Browser Session Preservation and Migration. In Poster Session of WWW 2002, Hawai, USA. 7-11. May, 2002. pp. 2.