A Client/Intercept Based System for Optimizing Wireless Access to Web Services

Irene Kilanioti, Georgia Sotiropoulou, Stathes Hadjiefthymiades

University of Athens, Department of Informatics and Telecommunications
Panepistimioupolis, Ilissia, Athens 15784, Greece
{grad0553, gsot, *shadj*}*@di.uoa.gr*

Abstract. *In this paper we discuss the introduction of Web Services in the wireless/mobile domain. The use of Web Services is gradually expanding into the mobile internet area where the population of users is rapidly increasing. Web Services offer standardized ways of creating, publishing, searching and invoking services and provide a very important platform for the development of mobile e-commerce. We identify problems related to the use of the verbose protocols of Web Services (i.e., SOAP/HTTP) over the "expensive" wireless medium. Our architecture assumes the existence of WS-aware software (client or provider) in the wireless device. We try to optimize the HTTP/SOAP stream exchanged over the wireless medium. Our effort is largely based on the IBM WebExpress design. Our measurements indicate substantial benefits for the users.*

1 Introduction

The extraordinary growth evident in the area of wireless and mobile technologies creates a dynamic environment that produces diverse wireless technologies and standards, in contrast to other areas of communications marked by a convergence toward uniformity. All this activity will bring the reality of the coming decades close to the vision of "anytime, anywhere" communications.

With the simultaneous exponential growth of the Internet, wireless users are now seeking Internet capabilities equivalent to those provided by a fixed network. Specifically, the growth of wireless telecommunications stimulated the interest for the so-called "nomadic computing"[1] [1], which aims to provide users with access to popular desktop applications, applications specially suited for mobile users and basic communication services.

Wireless/mobile computing is a very challenging research area also due to the low data rates usually available. Moreover, wireless connections suffer a high variability in terms of bandwidth, are significantly less reliable than their wired counterparts, and, could be interrupted for various reasons (e.g., handovers). Additionally, the cost for a wireless data connection is increased: the cost per byte transmitted over the

[1] The emergence of nomadic computing was also facilitated by the rapid proliferation of portable computer equipment.

wireless interface is considerably higher than in wired infrastructures. Consequently, our focus should be centered on the reduction of data volume to be exchanged over the wireless medium. The efforts should concentrate on the development of a model able to support multiple types of applications, including current and emerging TCP/IP applications and terminal emulation.[2]

During the past years a number of efforts were made to consolidate the WWW with wireless communications. Notable examples are the MobiScape Project [2] and IBM WebExpress [3][3]. Our work is mostly based on the ideas integrated in the latter solution[4].

In this paper we discuss the design, implementation and performance evaluation of a mobile computing system that optimizes access to ***Web Services*** (WS)[5] and yields substantial benefits for the end-user. To the best of our knowledge there exists no equivalent client/intercept architecture for wireless WS.

The rest of the paper is structured as follows: In Section 2, we give a brief outline of WebExpress. In Section 3 we present the proposed architecture and discuss why WSs are conducive for the integration with an intercepting architecture and how they can be incorporated in a WebExpress-like system. We emphasize on the adopted object-oriented design along with the different roles assigned to the various classes/components, the functionality of the system, the protocol reduction, and, finally, the software components used for implementation. In Section 4, we elaborate on our experimental set-up, the metrics monitored and the collected statistics, and, finally, the same Section concludes this paper.

2 Prior Work

The use of WS in the wireless domain has been discussed extensively in [6]. The architecture scenarios discussed in the referenced paper refer to the operation of the mobile device either as requestor or as service provider.[6]

The basic architectural model of WebExpress consists of the Client Side Intercept (CSI) running in the end-user mobile device and the Server Side Intercept (SSI)

---

[2] The intercept model offers this advantage as it is transparent to both the client and the server, and therefore, can be employed with any client adaptation and be insensitive to the development of a particular client/server or communications technology.

[3] A more extensive survey of such systems can be found in [4].

[4] Both systems capitalize on the WWW proxy interface and use a similar architecture (also termed "intercepting technology"). WebExpress proposes an architecture that addresses the requirements of transaction processing through the adoption of "differencing techniques".

[5] WS are a new breed of web applications following the Service Oriented Architecture (SOA) framework whose goal is to achieve loose coupling among interactive software agents [5]. A WS is a SOA with interfaces based on Internet protocols (e.g., HTTP), and messages encoded in XML (except for binary data attachment).

[6] In general, the protocols used in the wired WS world are inappropriate for the wireless medium. Therefore the wireless Internet community is pursuing their optimization. Commercial players like Microsoft and Sun introduce WS-specific features in their development frameworks and operating systems. In the recent past, optimized versions of SOAP and XML have been proposed for the wireless domain.

running in the fixed network.[7] WebExpress's differencing optimization technique is applied on responses containing dynamic content (e.g., CGI output). Dynamic responses are not stored in cache but maintained as base objects: A common base object, associated with the resource, is created in both the CSI and SSI. Subsequent references to the resource will trigger, at the SSI, the computation of differences between its present form and the common base object. Such differences are transmitted over the radio interface. The CSI reconstitutes the referenced object and delivers it to the browser.[8]

3 Proposed Architecture

Description: At the top of the WS message stack is a mechanism for envelope extensions called SOAP headers.[9]

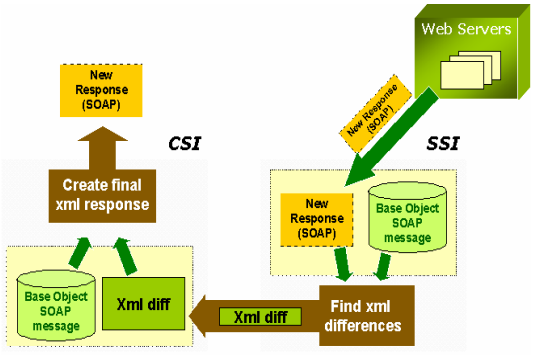

Fig. 1. Client/Intercept Architecture for WS

WS are the evolution of component-based systems such as DCOM, CORBA, RMI and EJB, that, also support distributed application logic. WS technology is considered more advantageous to these systems because it uses HTTP to be firewall friendly and payload agnostic, employs the more widely adopted XML scheme, uses the pervasive Internet concept of URLs to address object identification and offers more than a promise for interoperability as it exploits the openness of specific Internet technologies to address many of the interoperability issues of the previous systems.

---

[7] The CSI acts as a local proxy agent. The SSI is responsible for reconstituting a URL compatible data request and forwarding it to the designated origin server. On the reverse direction, the CSI reconstitutes the WWW stream and delivers it to the browser. Both the CSI and SSI incorporate caching mechanisms.

[8] HTML streams representing query responses usually contain a lot of unchanging formatting information (e.g., graphics, headers and footers). Responses from the same script are usually differentiated only by alphanumeric information.

[9] With SOAP headers, orthogonal extensions such as digital signatures can be associated with the body of the message contained in the SOAP envelope.

Due to the above reasons and especially due to the HTTP/SOAP binding and use of URLs for identification, an optimizing mechanism for access to RPC-oriented[10] WS can be easily implemented with a Client/Intercept based system. In our proposal, there is an obvious analogy to the Client/Intercept model applied for the WWW. Instead of simple HTTP, now a SOAP call is packaged as the body of an HTTP request. In that sense, the differentiating engine is applied on the exchanged message, which is XML-encoded, and base objects are stored SOAP messages that essentially comprise a response for a WS, as depicted in Fig.1. In relation to the work in [6], our scheme can handle both a mobile device hosted WS client and server.

Object-Oriented Design: In this section we discuss also the object-oriented design of our implementation along with the functionality of constituent components.[11] **CSI Objects** *Client:* (CSI) Establishes the connection with the SSI and the browser, and initializes all other objects used. For each request received from the browser, it creates a thread (a ClientThread) to serve that request. *ClientThread:* Used to serve a specific request, and terminated as soon as this request is fulfilled. Many instances of this object, each of them serving a different request, may exist simultaneously. *Demultiplexor:* Resides on the CSI side and has the responsibility of the demultiplexing of the packages that arrive from the SSI and are destined to a specific browser. **SSI objects** *Server:* (SSI) One instance of this object is initially created, and is waiting for incoming connections from the CSI. For each incoming request, a new ServerThread is created to serve that request. *ServerThread:* Operating as a thread, this object is used to serve an incoming request from the CSI. It is generated by the Server object, and all its instances operate on the original threads of the SSI, and not on copies. We should stress out that the same applies for the ClientThread objects, which operate on the original objects of the Client. **Common objects** *BaseObjEntry:* Represents an entry for a base object. These entries are stored in the BaseObjHTable and include information such as: the URL for the base object, the filename that holds the data (xml code) for that base object, the CRC value of the object, and information about the time of storage and last access of this Base Object. *BaseObjHTable:* It is a hash table of BaseObjEntries, being used to store information about a specific base object entry. The key for the hash function is the URL of the base object. *BaseObj:* Manipulates the file system of the differencing subsystem. The files stored in a specific folder, are the base objects. The BaseObj controls the access to these files and performs a folder cleanup whenever the total size of files is greater than 80% of the maximum space specified by the administrator. Files are removed, until the total size of the cache is less than 60% of the maximum disk allotment.

Protocol Reduction: Towards the reduce of redundant volume exchanged, the techniques adopted are: *Virtual sockets:* The system minimizes overhead caused by opening and terminating multiple TCP connections, with the establishment of a single TCP connection between CSI and SSI used for all HTTP communication (i.e., requests and responses). The TCP connection is persistent across different transactions and alleviates the deficiency of HTTP's operation on top of TCP, which

[10] WS should be RPC-oriented, so that they apply to the case in which WebExpress has to deal with responses containing dynamic content.

[11] Since there are many similarities in the functionality of the CSI and the SSI, some of the following objects are used by both components.

has much more negative effects over a wireless interface. In order to support a multiplexing functionality, we implemented a *virtual sockets* mechanism[12]. On the fixed network side, the virtual socket is associated with a normal socket on the designated WS provider. Some of the established virtual sockets are used for conveying commands related to normal socket calls (e.g., open, close) as a form of in-band signaling. *HTTP header reduction:* The MIME list of the types of documents that a given browser can handle usually remains constant. Therefore, the CSI allows this header information to flow towards the SSI only in the first request after the establishment of their connection and does not transmit it continuously.[13]

Functionality: Initially[14], there is no entry for a specific WS neither in the CSI nor in the SSI. When the first request comes, the CSI forwards it to the SSI, and the latter sends it to the WS provider. When SSI receives the response, it stores[15] it (and calculates the Cyclic Redundancy Check or CRC of it, which serves as a unique identifier for the message), before forwarding it to the CSI.[16] In the same way, it is stored at the CSI before being sent to the browser. At this point, a base object has been stored for the URL of the requested WS. Each base object is identified by the base URL. Let us suppose that there comes another request for the same WS. As soon as the request is intercepted, the CSI proxy checks if it has its URL stored. If a version is found, CSI forwards the request along with a relevant indication of existence and the CRC value of that base object. In every case, a message is sent to the SSI and the SSI always communicates with the WS providers. If there is no base object with the specific CRC at the SSI, SSI notifies CSI that it has to store it. Otherwise, if the CRC of the CSI base object is identical to its counterpart at the SSI, the differencing engine

---

[12] In this mechanism, a small header that contains a virtual socket identifier prefixes data sent for a specific request. At the CSI, the virtual socket identifier is bound to a real socket at the browser. An actual TCP connection is established concurrently with the "opening" of the first virtual socket. It closes after a specific period elapses since the last virtual socket was closed. CSI and SSI are responsible for the role assignment according to the virtual socket. CSI receives requests from the browser and forwards them to the SSI. For every received request, the SSI establishes a connection with the respective WS provider and forwards the request. As soon as it receives the response, it closes the connection with the server, sends the document to the CSI via the TCP connection, but does not terminate it. CSI forwards the document to the browser, and closes the connection with the latter.

[13] Both CSI and SSI maintain this Accept List as part of the connection status information. For every request received from the browser, the CSI checks the list, to verify if it is identical with the stored one, and if so, the list is omitted from the request, but subsequently inserted by the SSI. In case there is another list in the request, SSI stores the new list.

[14] It should be mentioned here, that, in the first place, when the CSI receives an HTTP request, it checks if it is a request for a WS. The check is conducted in the following way: if the binding method between SOAP and HTTP is GET, the suffix of the URL is checked. If, on the other hand, the used method is POST, and consequently, there is a body in the transmitted message, the CSI soap proxy checks, with the help of JDOM XML parser, if the body of the received SOAP message (an XML document) is well formed. If this check is successful, it forwards the message to the SSI and the processing continues. Otherwise, an HTTP error message is created and the request is not processed any further.

[15] The storage takes place on scope of the entire SOAP message transmitted.

[16] In the framework of the synchronization schema implemented between the CSI and the SSI.

is engaged. If the files are not identical (i.e., different CRC values), SSI replaces the existing base object with the new one and notifies CSI to do the same.[17]

Implementation Details: As aforementioned, our system was implemented in Java. We used the Apache Ant and Log4j tools for building management and logging mechanisms respectively, as well as the JDOM parser for check of XML request syntactical correctness. The graphical user interface (implemented with Java Swing) allows the administrator to define operational parameters, and view the accumulated statistics along with information concerning base objects. In order to implement the differencing mechanism, we employed Microsoft XML Diff and Patch, a set of tools for comparison of two XML documents and application of the changes (patching).[18]

4 Performance assessment

The performance of the system was assessed through a series of experiments (1000 WS requests performed per experiment).[19] The 40%, 60%, 80% and 100% respectively of full storage capacity of our proxies was exploited. In every case, the LRU algorithm was enforced as soon as the capacity of stored objects became greater than 80% of the current proxy capacity. Additionally, two series of experiments were implemented: one with 10 distinct WS (randomly chosen by the Client) and one with 20 WS. The monitored statistics included, among others, the Local/Total Bytes Read Ratio, that practically indicates the efficiency of the system by denoting the ratio of bytes that the CSI delivered to the browser against bytes retrieved from the SSI by the CSI. The accumulated statistics are presented in Table 1 and Table 2.

Table 1. Experiment results for 10 available WS

Experiment No.	Base Object Size (MB)	Percentage of Proxy Capacity Used	Base Object hits	Total Bytes Read (CSI)	Total Bytes Read (SSI)	Bytes Read Ratio %
1.	25131	40%	231	4906932	6179314	79,409
2.	37696	60%	335	4387517	6477379	67,736

[17] For even more optimization, the differencing stream (i.e., the difference between the base object stored at the SSI and the answer from the server) is sent to the CSI-instead of the response from the origin server-only on the presumption that the size of the resulting stream of the differencing engine is less than a percentage (70% in our implementation) of the size of the actual xml document, which would be sent. Otherwise, the retrieved response is directly forwarded to the CSI. The differencing stream (which is also XML-encoded) is sent to the CSI. At the CSI, an entity update engine uses the differencing stream along with the stored base object of the request, in order to patch the new response (at the CSI rebasing takes place), and ultimately forward it to the browser.

[18] XML Diff is based on the Document Object Model (DOM) and detects addition, deletion and structural changes like the removal of an XML sub-tree. It produces *Xml Diff Language Diffgram*, an xml-based concrete and short output language, which can then be used to perform a patch operation using the XML Patch.

[19] There was no need to perform a large number of requests to “warm up” any cache, so the above-discussed number of trials has been considered adequate.

3.	50261	80%	448	3821632	6246173	61,184
4.	62826	100%	580	3356139	6567688	51,101

Table 2. Experiment results for 20 available WS

Experiment No.	Base Object Size (MB)	Percentage of Proxy Capacity Used	Base Object hits	Total Bytes Read (CSI)	Total Bytes Read (SSI)	Bytes Read Ratio %
1.	39070	40%	239	4007997	4816465	83,215
2.	58605	60%	355	3651059	4991831	73,141
3.	78140	80%	455	3136082	5172683	60,628
4.	97674	100%	588	2538281	5255106	48,301

It can be observed that the traffic reduction was substantial (approximately 40%), an effect that was anticipated: the number of bytes read at the CSI was more than 40% of the bytes read from the SSI. As the number of used proxy capacity increases, so does the Bytes Read Ratio[20] too, since the possibility of hitting an existing base object increases, and thus there is no need for data exchange.

Having realised a series of experiments with the purpose of evaluating the alleged benefits from the various optimisation techniques, a series of experiments that demonstrated a considerable improvement in the use of WS resources (with the indicative example of 40% gain in the volume of information flowing over the wireless interface between the CSI and the SSI), we conclude that our WS-oriented client/intercept system contributes to decreased latency, decreased bandwidth use and other efficiency benefits.

References

1. T.F.La Porta, et al., “Challenges for Nomadic Computing: Mobility Management and Wireless Communications”, ACM Journal of Nomadic Computing, Vol.1 No.1, 1996.
2. C.Baquero et al., “MobiScape: WWW Browsing under Disconnected and Semi-Connected Operation”, proceedings of the 1st Portuguese WWW National Conference, Braga, Portugal, July 1995.
3. B.C.Housel, and D.B.Lindquist, “WebExpress: A system for Optimizing Web Browsing in a Wireless Environment”, proceedings of ACM/IEEE MobiCom ’96, NY, USA, Oct. 1996.
4. S.Hadjiefthymiades, and L.Merakos, “A Survey of Web Architectures for Wireless Communication Environments”, Journal of Universal Computer Science, Vol.5, No.7, Springer Verlag, 1999.
5. Thomas Erl, "Service-oriented Architecture: A Field Guide to Integrating XML and Web Services", Prentice Hall, April 2004.
6. T.Pilioura, A.Tsalgatidou, S.Hadjiefthymiades, “Scenarios of using Web Services in M-Commerce”, ACM SIGecom Exchanges, Vol. 3, No. 4, January 2003.

[20] The ratio of bytes read at the CSI to bytes read from the SSI. It should be noted, however, that this percentage largely depends on the differencing mechanism incorporated in the system.