Building Highly Dependable Wireless Web Services

Wenbing Zhao, Cleveland State University, USA

ABSTRACT

In this paper, the authors introduce Web services technology and its applications to mobile business transactions. This paper shows that the Web services technology is a powerful tool to build next-generation e-Commerce applications for wireless mobile devices following the service-oriented architecture. Such an approach would bring significant benefits to organizations involved with e-Commerce. The authors further discuss the importance of ensuring high dependability of Web services and provide a literature review of state-of-the-art techniques that are critical to the implementation of practical and dependable wireless Web services. Finally, research on the design, implementation, and performance evaluation of a fault tolerance framework for wireless Web services are described.

Keywords: Algorithms, Distributed Systems, E-Commerce Quality Issues, Middleware, Service Interruption, Web-Based Commerce, Wireless Technologies

INTRODUCTION

Electronic commerce or e-Commerce in short, refers to the selling and buying of products and services over electronic systems, which such as the Internet (Chaudhury & Kuilboer, 2001). E-Commerce has grown to a 100 Billion per year business and it is expected to continue growing with a staggering speed (Scheleur, King, & Kinyon, 2008). This article touches on two emerging issues related to e-Commerce: (1) e-Commerce over wireless mobile devices, and (2) the use of the Web services technology for e-Commerce systems. A Web service refers to a programmatic interface accessible by other applications via the World Wide Web (W3C, 2008). The Web services technology typically refers to the set of standards that enable Web services (more details will be provided in the next section).

We expect that many next generation e-Commerce systems will be built using the Web services technology and most of them will provide support for wireless mobile devices (by providing and enabling the consumption of wireless Web services). However, there exist substantial technical challenges in materializing this vision. In this article, we focus on one of the most important of challenges, i.e., how to ensure high dependability of wireless Web services provided by such systems. The reason why high dependability becomes a major concern for e-Commerce providers and users alike is primarily due to the high financial cost associated with unreliable services. Services are expected to be highly dependable 24 by
7 and the unavailability of a service not only disrupts many end-users of the service, but the composite services built on top of the service as well. Extensive downtime would often mean massive loss of revenue, loss of employee productivity, and eventually the reputation of a business (Patternson, 2002).

This article is structured as follows. In the next section, we introduce the Web services technology and elaborate the benefits of using the Web services technology for conducting e-Commerce over mobile devices. We then layout the primary challenges for the Web services technology to support wireless mobile devices, and provide a literature review of the state-of-the-art solutions towards the objective. This is followed by the presentation of our research on dependable wireless Web services, including design, implementation and performance evaluation. Finally, we conclude the article and point out future research directions.

Wireless Web Services for E-Commerce

In this section, we first introduce the Web services technology, and then elaborate the benefits of using the Web services technology for conducting e-Commerce over wireless devices.

The Web Services Concept

The Web services technology refers to the set of standards that enable automated machine-to-machine interactions over the Web. The cornerstones of the Web services technology include eXtensible Markup Language (XML) (Bray et al., 2006), HyperText Transfer Protocol (HTTP), Simple Object Access Protocol (SOAP) (Gudgin et al., 2007), and Web Services Description Language (WSDL) (Christensen et al., 2001).

XML is designed to facilitate self-contained, structured data representation and transfer over the Internet. It is extensible because it allows users to define their own tags. The extensibility of XML makes it the essential building block for Web services. Here the term extensibility refers to the capability of introducing additional features and functionalities without disrupting existing systems.

HTTP is an application-level protocol following the client-server interaction model. HTTP is designed to share and access Web resources (i.e., hypertext objects). However, HTTP is a stateless protocol in that it does not keep track of the state across different HTTP requests. The design of HTTP not only removes unnecessary complexity, it ensures high degree of scalability as well.

As the name suggests, SOAP was originally designed to conduct remote procedure calls over the Web. It has evolved to become the main communication protocol to exchange XML documents. A SOAP message contains a SOAP Envelop and a SOAP Body. A SOAP message often contains an optional SOAP Header element and a Fault element if an error is encountered by the sender of the SOAP message.

WSDL is an XML-based language used to describe Web services. For each Web service, the corresponding WSDL document specifies the available operations, the messages involved with the operations, and a set of endpoints to reach the Web service. Due to the use of XML, WSDL is also extensible. In particular, it allows the binding of multiple different communication protocols and message formats.

To enable dynamic services publishing and discovery, a Universal Description, Discovery and Integration (UDDI) (Clement et al., 2004) service could be used. UDDI provides the standard way for Web services providers to describe their services, and for the consumers to search and discover available services. However, the experiments of offering global-scale public-domain UDDI registries were not successful, as evidenced by the shutdown of once well-known UDDI registries provided IBM, Microsoft and SAP (Krill, 2005). One reason for this failure is perhaps due to current business requirements, i.e., inter-enterprise transactions can rarely take place before the relevant legal documents are signed by the companies involved in the transactions. Nevertheless, UDDI has been gaining momentum to be used as a Web services registry within each enterprise. Thus, from an
architecture point of view, the Web services platform consists of Web services providers, Web services consumers, and UDDI registries that broker the providers and the consumers, as shown in Figure 1.

**Using the Web Services Technology for Wireless E-Commerce**

*E-Commerce over wireless mobile devices.* The total number of mobile phone subscriptions worldwide had reached 3.3 billion by November 2007, which makes the mobile phones the most pervasive technology and the most popular electronic device in the world (Reuters, 2007). As mobile devices become ubiquitous and more powerful, it is natural to expect the extension of e-Commerce over these devices (Tiwari & Buse, 2007). The demand for conducting e-Commerce over mobile devices is driven both by consumers who want to enjoy the convenience of shopping online on the go, and by business owners who want to increase the productivity of their workforce (e.g., if they have workers in the field who need to engage in e-Commerce operations on mobile devices). This demand would open many opportunities for new services and revenue sources for e-Commerce service providers. Therefore, it is essential for managers at such institutions to reconfigure organizational resources to embrace the new challenges and opportunities (Wu & Hisa, 2008).

---

**Figure 1. The Web services architecture**
with minimum assumptions on other systems with which it communicates, and therefore, this approach reduces the risk that a change in one system will force modification of other systems. Furthermore, SOA also enables the loose-coupling of different parts of the system so that some parts of the system could undergo changes (such as an upgrade of the implementation) without significantly disrupting other parts of the system. Well-defined Web services also encourage the reuse of existing services in the production of business applications (Erl, 2005). Hence, SOA brings more agility to organizations to reallocate their resources for rapid and low-cost system development and yet with potentially higher system quality. In the SOA paradigm, e-Commerce transactions are conducted in the form of consumptions of Web services (Newcomer & Lomow, 2005). The Web services targeted to wireless mobile devices are often referred to as wireless Web services (Pilioura et al., 2007).

DEPENDABLE WIRELESS WEB SERVICES: CHALLENGES AND PROPOSED SOLUTIONS

In this section, we discuss three primary challenges for building practical and dependable wireless Web services for wireless mobile device users, and for each challenge, we provide a literature review of possible solutions. (Note that these are what we believe the most important ones based on our research interest and experiences, and they are by no means complete.)

Challenge 1: Overcome the limited computing resources of mobile devices. Mobile devices have much lower CPU speed and smaller memory size comparing with their desktop counterparts, e.g., it was reported by Smykil (2007) that iPhone, one of newest mobile phones in the market, uses a 600MHz ARM CPU (probably runs at 400MHz to extend battery life) and 128MB memory. This means that the client software to run on the mobile devices has to be made as simple as possible with a small footprint (Ellis & Young, 2003).

Proposed Solutions for Challenge 1. The adoption of the Web services technology implies that SOAP must be used as the messaging protocol between the service providers and consumers. However, full support for SOAP seems to be problematic for wireless mobile devices because of their limited computing capacity. The proposed solutions so far can be roughly divided into three categories: (1) use only a subset of SOAP functionality (Ellis & Young, 2003), (2) use alternative SOAP-over-transport-protocol binding (Phan, Tari, & Bertok, 2006), and (3) use alternative SOAP encoding (Werner, Buschkam & Fischer, 2004; Apte, Deutsch, & Jain, 2005).

The most notable example of the first approach is the J2ME (short for Java 2 Platform, Micro Edition) Web services specification (Ellis & Young, 2003). This specification describes a set of lightweight APIs (application programming interfaces) to allow resource-limited devices to access Web services as pure clients, i.e., it does not allow the devices to offer Web services. Furthermore, it has no support for dynamic Web services discovery via UDDI, and it does not support the Document Object Model (Wood et al., 2000) for XML processing.

An example of the second approach is the SOAP-over-TCP binding. This binding has been provided in Apache Axis (2008) and Microsoft Web Services Enhancement (WSE) 3.0 (Fus sel, 2005). However, there does not seem to have standardization process on this binding. For small SOAP documents, SOAP-over-TCP can reduce the message sizes significantly, and therefore, achieve much shorter response time and higher throughput. For example, Phan, Tari, and Bertok (2006) reported that the message transmitted over the wire is reduced from about 700 bytes to below 500 bytes for the echoDouble request, and the corresponding response time is reduced from about 270 ms to about 170 ms.
Furthermore, a SOAP-over-UDP binding has been proposed as a Web Services Standard (Combs et al., 2004). The SOAP-over-UDP binding caters to certain types of applications that do not require the reliability provided by TCP, and/or need multicast. The use of UDP protocol not only avoids the overhead associated with TCP and HTTP, it further eliminates the connection establishment phase. Consequently, applications using SOAP-over-UDP exhibit the best performance in terms of response time and throughput, as reported in (Phan, Tari, & Bertok, 2006).

In the third approach, several XML-aware encoding mechanisms have been proposed, such as differential encoding (Werner, Buschmann, & Fischer, 2004) and WSDL-aware encoding (Apte, Deutsch, & Jain, 2005). Differential encoding attempts to reduce the SOAP message size by speculating the content of the message, deploying the speculated message at both the sender and the receiver prior to the runtime, and sending only the differences between the actual message and the speculated message. The WSDL-aware encoding attempts to reduce the information needed to be included in the message. In the best cases, these schemes can reduce the message by a factor of 10 or more. However, despite the potentially huge performance gains, we have strong reservations about using non-standard messaging protocols because this would create strong coupling, reduce interoperability, and hence risk wiping out the benefits of the Web services technology.

**Challenge 2: Overcome unreliable network connections and unpredictable online/offline behaviors of mobile devices.** The connectivity between the mobile phones and the Internet via cellular networks is still far less reliable than wired networks. Furthermore, a user of the mobile phone might turn it on or off fairly unpredictably, for example, due to low battery or other conceivable reasons. This requires the communication infrastructure for wireless Web services be always prepared for potential disconnection, and ideally, to provide continuous service to users during periods of transient network failures (Terry & Ramasubramanian, 2003).

**Proposed Solutions for Challenge 2.** Caching has been a classical technique to increase distributed systems performance and it is also a powerful tool to overcome the intermittent connection problem common to wireless mobile devices (Hadjietf-thymiades & Merakos, 2003). A cache (proxy) server is often positioned at or adjacent to the client and is used to cache the responses to the requests issued by the client. If a later request asking for the same object that has already been cached, the cached object will be returned immediately to the client without the need to send the request to the origin server.

For mobile devices, the caching technique can be extended to address the intermittent connection problem. Terry and Ramasubramanian (2003) proposed to use caching together with playback to increase the perceived availability of the Web services. The main idea is that both the requests and the responses are cached at the client. If a response object is found in the cache database for a newly issued request for a cacheable operation, the response object is returned immediately. If a request is for an update operation, such request must be sent to the server, or played back when the connectivity to the server is restored to ensure the consistency between the server and the client. Furthermore, when the client issues an update operation on certain object, any cached object from a previous query must be invalidated, again, for the purpose of cache consistency.

Recently, Liu and Deters (2007) proposed a dual-caching strategy to enhance the performance of wireless Web services. According to the dual-caching strategy, caching is performed at both the client side and the server side. Caching at the server side helps ameliorate the intermittent connection problem perceived at the server side. When the server is ready to send a response to the client, the connection to the client might be broken. In this case, the
existence of the server-side cache can be used to store the response until the connectivity is restored. The server-side cache manager can also handle duplicate requests by retrieving and sending the corresponding response objects to their clients.

One of the main concerns of caching is cache consistency. To maintain strong cache consistency, it often requires application protocol support. It is beneficial to know which operations’ results are cacheable and for how long, and what operations would invalidate a cached object. For example, HTTP has built-in support for caching. Unfortunately, the Web services platform lacks explicit support for caching. This makes caching for Web services particularly challenging.

**Challenge 3: Meet the high dependability requirements for service-oriented computing.** Service-oriented computing implies that the services must be highly dependable 24 by 7 because the unavailability of a service not only disrupts many end-users of the service, but the composite services built on top of the service. Extensive downtime could mean massive loss of revenue, loss of employee productivity, and serious damage of reputation of a business (Patternson, 2002). As reported in the 2001 Contingency Planning Management survey (2001), for most businesses, one hour of downtime would lead to a loss of $50,000 or more. Therefore, we believe that a high dependability infrastructure that addresses all three challenges listed above is urgently needed.

**Proposed Solutions for Challenge 3.** High dependability of a Web service means that it is always available to handle a client’s request correctly and in a timely fashion (Avizienis et al., 2005). To ensure high dependability, a Web service must be replicated so that the failure of one of the replicas will not bring down the entire Web service. There are two common replication styles, namely, active replication and passive replication (Powell, 1991).

**Active replication.** All replicas accept and process the same set of requests, and generate the same set of responses in the same order. For active replication, each replica is often modeled as a state machine. A request would cause a replica to transit from one state to another. To ensure the consistency of the state changes, the replicas must be deterministic or rendered deterministic, i.e., given the same set of requests in the same order, every replica must go through the same sequence of state transitions.

**Passive replication.** Only one replica (the primary) executes in response to the client’s requests. Passive replication has two variations: warm passive replication and cold passive replication. In warm passive replication, the primary periodically checkpoints its state and propagates the checkpoints to the backups. In cold passive replication, the backups are not launched until the primary is detected to have failed. It is the responsibility of the fault tolerance infrastructure to retrieve checkpoints from the primary and log them.

For highly available Web services, we have seen numerous proposed solutions (Chan, Lyu, & Malek, 2006; Fang et al., 2007; Looker, Munro, & Xu, 2005; Salas et al., 2006; Santos, Lung, & Montez, 2005; Zhao, 2007). To ensure strong replica consistency, there are primarily two common replication approaches. One is to use a group communication system that guarantees all the replicas in the same process group to receive the same set of messages in the same total order (Birman & van Renesse, 1994), such as WS-Replication (Salas et al., 2006). The alternative approach is to use a consensus algorithm (Lamport, 2001) to ensure total ordering of request messages, such as Thema (Merideth et al., 2005). (A consensus algorithm ensures that if a member in the group decides on a certain value, all other members will decide on the same value.) However, these solutions cannot be readily used for wireless Web services...
because they did not consider the challenges of overcoming the resource limitations of mobile devices (C1) and intermittent connections (C2).

**METHODOLOGY: DESIGN AND IMPLEMENTATION OF A FAULT TOLERANCE FRAMEWORK FOR WIRELESS WEB SERVICES**

In this section, we describe a lightweight fault tolerance framework for wireless Web services (referred to as LFT-WWS). To avoid being too technical, we omit the description of the fault tolerance algorithm employed in LFT-WWS. Interested readers are referred to (Lamport, 2001) for details. The framework has two main objectives: (1) it ensures high availability of the Web services by replicating the server-side components, and (2) it reduces the end-to-end response time and increases the perceived availability of the Web services at the client side by incorporating a two-way caching mechanism (Liu & Deters, 2007). Therefore, the primary organizational benefit of LFT-WWS is the enhanced dependability of the Web services provided and their usability for wireless mobile devices. In addition, comparing with similar approaches cited in the previous section, LFT-WWS has the following unique features, which will bring further organizational benefits to the Web services providers:

- It does not use any proprietary communication protocol to coordinate the server replicas, and to enable the communication between the clients and server replicas. All the messaging required for replication is defined in WSDL which is publicly visible to users. This ensures interoperability and extensibility of the Web services, which is important to any organizations.
- Our framework is lightweight in that it does not impose sophisticated replication and fault management requirement. The configuration is through a simple property file. The fault detection is incorporated in the replication mechanisms. Simpler replication management means fewer operator errors.
- The framework requires minimum changes to the Web services and their clients. On the service side, only two additional operations are introduced to retrieve and restore the service state. All other changes happen in the configuration files used by the SOAP engine.

**Implementation of the LFT-WWS Framework**

Our fault tolerance framework for Web services is built on top of two open source libraries. On the server-side, we use Sandesha2 (2008), an implementation of the Web Services Reliable Messaging (WS-ReliableMessaging) specification (Davis et al., 2007). On the client-side, we use ksoap2 (2008), a lightweight SOAP Web service client library for constrained Java environment such as J2ME applications. The architecture of the fault tolerance framework and the main server-side components are shown in Figure 2.

The main server-side components include a Replication Engine, a Total Order Invoker, an Out Handler, an In Handler, and a Sender. The Replication Engine is the core component in our framework. This component drives the execution of the replication algorithm. The Replication Engine uses its own storage to log the replication control messages, and the checkpoints.

The Total Order Invoker component is responsible to deliver requests to the replicated Web service in a total order. At each replica, the Total Order Invoker fetches the application message from the In Msg Queue, which stores all incoming application requests.

The Out handler performs preliminary processing on all outgoing messages. All outgoing messages are placed in the Out Msg Queue (i.e., the server-side cache) before they are sent out by the Sender component. The Sender knows what messages to be multicast by examining the SOAP action property included in each message.
If a multicast is needed, the Replication Engine is consulted to obtain the multicast destinations.

The In handler handles the control messages used by the replication algorithm. All incoming messages are placed in the In Msg Queue after preliminary processing, and before they are delivered to the Web service by the Total Order Invoker.

The Sender component is responsible for sending messages to their destinations through point-to-point or multicast channels. The multicast functionality is used only by the primary, and by a backup for checkpoint/view change messages. To perform the multicast, multiple threads are launched to concurrently send the same message to different destinations. Each thread is responsible to send the message to a distinct destination point-to-point.

The client-side architecture is much simpler. Its main component is a cache manager that stores both outgoing requests and incoming responses. The policy of the caching, such as the cache size and the duration of each cached message, can be dynamically configured by the client.

To enable caching of a reply message, a cache key of the corresponding request is constructed so that when a future request is issued, the cache can be searched for possible cached reply message. In our framework, we use the target operation name together with the argument set as the cache key. We assume that the Web services provider supplies metadata information regarding which operation is cacheable and what operation would invalidate a cached record.

**Performance Evaluation of the LFT-WWS Framework**

We evaluated our framework on a testbed consisting of a group of Dell SC440 servers. Each SC440 server is equipped with a 2.8GHz dual-core processor and 1GB RAM, running SuSE 10.2 Linux. To evaluate the system performance as experienced by a wireless mobile device user over a 3G cellular network (Kurose & Ross, 2005), we injected delays according to the size of each message sent in the framework to emulate the downlink (we use the value of 1Mbps) and
uplink bandwidth (we use the value of 128 kbps) offered by typical 3G service providers. The performance of our framework is assessed using a micro-benchmark application and a more realistic travel reservation application.

**Micro-benchmark results.** The micro-benchmark experiment is used to characterize the runtime overhead of our framework. The test application contains two operations: query (read-only operation) and update (read-write operation). For each operation, we experiment with the following three request/reply combinations: short/short, long/short, and short/long. The short message contains a single element (total message size over the wire is about 400 Bytes) and the long message contains 100 elements (total message size over the wire is about 3300 Bytes).

As shown in Figure 3, we compare the end-to-end latency as observed by the wireless mobile client under the following three configurations: no server replication (Configuration I), 3-way replication (to tolerate a single replica fault) (Configuration II), and 5-way replication (to tolerate 2 faulty replicas) (Configuration III). During each run, the client sends a request to the replicated Web service and waits for the corresponding reply within a loop without any “think” time between two consecutive calls for 2000 iterations, and we report the average value of the end-to-end latency. The results are summarized in Table 1.

As can be seen from Table 1, the runtime overhead of our framework is significant, but reasonable, for the update operations considering the complexity of the Paxos algorithm and the fault tolerance benefit achieved. The overhead is primarily due to the additional communication cost of the control messages involved in the Paxos algorithm to ensure the consistency of the replicas. For the query (i.e., read-only) operation, the overhead is reduced significantly with the tentative delivery optimization because of the elimination of the consensus steps. Note that the micro-benchmark results show the worst-case overhead of our framework because the operations involve virtually no work.

**Assessment using a travel reservation application.** To evaluate the performance of our framework for practical use, we developed a travel reservation application where a wireless mobile client can make reservations for flight, rental car and hotel room as part of a business activity. The application involves the following components: coordination services for the business activity, the flight reservation Web service, the rental car Web services, the hotel reservation Web service, and the mobile client. We assume that except for the mobile client, other components communicate via high-speed wired networks. To emulate the actual work of producing a quote and to finalize a reservation, we assume that each such operation takes about 100 ms. We further assume that the interactions between these components, as shown in Figure 4, conform to the Web Services Business Activity Specification (WS-BA) (Freund & Little, 2007), and an extension to WS-BA, referred to as Web Services-Business Activity-Initiator protocol (WS-BA-I) (Erven et al., 2007) that governs the interaction between the initiator of the business activity (i.e., the mobile client in our case) and the coordinator.

As can be seen from Figure 4, the travel reservation application involves two interaction styles not-seen in simple client-server applications such as the micro-benchmark test application: (1) nested invocations and (2) one-way invocations. Nested invocations, such as the one formed by messages 14 and 15 in Figure 4, are made in response to another (root-level) invocation such as request 13 in Figure 4. In essence, the Web service that issued the nested invocation has dual-roles: it acts as the server...
Figure 3. Three configurations used in the microbenchmark experiment

Table 1. Micro-benchmark results. The percentage overhead with respect to the no-replication case is also provided

<table>
<thead>
<tr>
<th>Request/Reply (No. Elements)</th>
<th>With 3-Way Replication (ms) (Configuration II)</th>
<th>With 5-Way Replication (ms) (Configuration III)</th>
<th>Without Replication (ms) (Configuration I)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Query</td>
<td>Update</td>
<td>Query</td>
</tr>
<tr>
<td>1/1</td>
<td>95.8 (12%)</td>
<td>150.3 (76%)</td>
<td>98.2 (15%)</td>
</tr>
<tr>
<td>100/1</td>
<td>277.1 (4%)</td>
<td>331.7 (24%)</td>
<td>279.5 (5%)</td>
</tr>
<tr>
<td>1/100</td>
<td>101.6 (15%)</td>
<td>153.3 (74%)</td>
<td>106.9 (21%)</td>
</tr>
</tbody>
</table>
for its clients, and also as a client for another Web service. One-way invocations, such as messages 26-31 and 36-41, are very useful to implement asynchronous communication, which is essential to achieve loose-coupling.

Because the LFT-WWS architecture described earlier in this section contains modules for both client-side and server-side mechanisms, extending it to support multi-tiered applications requires only a minor change. A sequence between two replicated Web services is mapped to a virtual sequence between two groups of replicas instead of between a single client and a group of replicas.

In our experiment, we compare the end-to-end latency as seen by a single mobile client with and without replicated Web services. Due to the limited number of nodes in our testbed, we use only 3-way replication for each replicated component. We experimented with four different configurations, as shown in Figure 5.

As can be seen from the experimental results summarized in Table 2, without replication (Configuration 1), the total end-to-end latency

Figure 4. Sequence diagram for the travel reservation test application
for the entire business activity is about 2.56 seconds. This is the sum of all 21 invocations (with 42 request and reply messages), 12 of which are from the mobile client to the Web services (including the coordination Web services). Among the 21 invocations, 4 are read-only and the remaining ones involve with update operations. When only the coordination services are replicated (Configuration II), the latency increases to about 3.09 seconds, i.e., our framework incurs approximately 21% overhead. In this case, 8 requests (for update operations) sent to the replicated coordinator is totally ordered and only one request (message 33 in Figure 4) is read-only. For other type of applications that consist of more read-only operations, the overall overhead could be much lower.

When the coordinator and the flight reservation service are both replicated (Configuration III), the total latency for the business activity reaches 3.31 seconds, a 29% overhead comparing with the latency of the no-replication configuration. Again, only 2 out of 11 invocations are for read-only operations (i.e., request messages 5 and 33).

When all Web services are replicated (Configuration IV), the total latency is about 3.75 seconds. The relative high overhead of 46% is not surprising because 17 of the 21 requests will have to be totally ordered. Nevertheless, the overhead could be a concern for practical
applications. We are currently investigating ways to reduce the overhead by considering the semantics of the applications.

**DISCUSSION**

Our fault tolerance framework described above aims to overcome the three major challenges in building dependable wireless Web services. To accommodate the limited computing resources of mobile devices (challenge 1), we chose to use a lightweight SOAP implementation (ksoap2) and a lightweight fault tolerance algorithm. We do not consider some existing optimizations such as SOAP-over-TCP, SOAP-over-UDP, or XML-aware encoding because of interoperability concerns. To address the unreliable network connections and unpredictable online/offline behaviors of mobile devices (challenge 2), we adopted the caching mechanism both at the client and at the server. To meet the high dependability requirements for service-oriented computing, we resort to state machine replication with a lightweight fault tolerance algorithm so that the services are still available to their users even if some replicas fail.

**CONCLUSION**

In this paper, we introduced the Web services technology and its applications to mobile business transactions. We argued that the Web services technology could be a powerful tool to build next-generation e-Commerce applications for wireless mobile devices following the service-oriented architecture. Such approach would bring numerous benefits to organizations involving with e-Commerce, such as rapid application development and faster systems integration, and thus, it is widely believed that the Web services technology will bring excellent return on investment (Erl, 2005). We further argued the importance of ensuring high dependability of Web services – service downtime often would lead to significant financial losses (Patternson, 2002).

In the main body of this article, we first enumerated the challenges for building highly dependable wireless Web services and presented a literature review of the state-of-the-art solutions. We then presented our research work on the design, implementation and performance evaluation of a light-weight fault tolerance framework for Web services. We showed, by using both a micro-benchmark test and a more realistic travel reservation application, that our framework incurs moderate runtime overhead. Nevertheless, our current approach has a number of limitations. Despite the optimizations we introduced, the runtime overhead might still appear to be high for practitioners, especially for update-intensive applications. We are currently investigating ways to reduce the overhead by considering the semantics of the applications. Furthermore, our current framework assumes a crash-fault model, which would not work if an adversary could penetrate the replicated systems and introduce arbitrary attacks. We are investigating a secure-version of our framework to withstand such attacks.
Existing research in the field of highly dependable wireless Web services have largely focused on the optimization of SOAP-based communication and caching. Although there is no doubt that further investigation is needed for better optimized SOAP-based communication protocols, and robust and efficient caching protocols for resource constrain devices, what seems to be lacking is an integrated approach towards practical highly dependable wireless Web services that considers all aspects of the challenges. Our research is the first step towards this direction.

ACKNOWLEDGMENT

We sincerely thank the anonymous reviewers and the associate editor for their invaluable comments and suggestions on an earlier version of this article. This work is supported by a Faculty Research Development award from Cleveland State University.

REFERENCES


Wenbing Zhao received the Ph.D. degree in Electrical and Computer Engineering from the University of California, Santa Barbara, in 2002. Currently, he is an assistant professor in the Department of Electrical and Computer Engineering at Cleveland State University. His research interests include Web services, distributed systems, fault tolerance and security. In the past, he has conducted research in the fields of superconducting materials and quantum optics. Dr. Zhao has more than 60 academic publications.