Achieving Web Service Continuity in Ubiquitous Mobile Networks: the SRR-WS Framework

Christoph Dorn and Schahram Dustdar

VitaLab, Distributed Systems Group,
Institute of Information Systems,
Technical University of Vienna,
Vienna, Austria
dorn|dustdar@infosys.tuwien.ac.at

Abstract. In this paper, we will address two underlying problems of Web service continuity in mobile ubiquitous networks. On the one hand, our work presents a lightweight solution for connection interruption. On the other hand, we introduce a technique to switch between devices during service invocation. The Suspend-Relocate-Resume for Web Services (SRR-WS) Framework meets these challenges by allowing a device to disconnect in-between Web service invocation and response in order to retrieve the reply upon reconnection. Moreover, our proposed framework also enables on-demand transfer of service control from one device to another. Our first reference implementation demonstrates how Web service continuity can be achieved across devices as well as unreliable connections.

1 Introduction

Applying Web service technology in mobile ubiquitous environments enables interaction of heterogeneous resources through loose coupling as well as platform and context independence. Jørstad et al. present in [1] the advantage of applying the Service-Oriented Architecture in mobile and ubiquitous networks in more detail. As mobile devices are getting more powerful and widespread, accessing Web services with wireless devices is the next logical step. However, the dynamic nature of ubiquitous environments demands for adapted Web service technologies. Bandwidth and connectivity are the main factors besides computing power, battery capacity, and input interfaces that restrict the deployment of Web service technology.

In this paper, we focus on the issue of service continuity in a relaxed form. Instead of demanding anytime, anywhere available services, we argue that using the same service session on different devices from different places with possible intermediary breaks is often sufficient. In many cases, a user needs not be persistently connected or does not desire it due to cost reasons. In Section 2, we present a motivating scenario and discuss problems and related factors that arise due to the dynamic nature of mobile networks. Section 3 discusses current approaches and solutions to these issues. Thereafter, Section 4 discusses our proposal in detail. Subsequently, we present our results in Section 5. Finally, Section 6 describes future work, and Section 7 summarizes our findings.

2 Problem Statement

First, we describe which components constitute a mobile ubiquitous environment concerning Web services. As presented in Fig.1, such an environment generally consists of wireless-enabled devices that communicate with each other and/or are connected to an access point. We also include a wired network in our setting having the role of an infrastructure of any kind or offering services. We justify this decision by making the assumption that mobile users venture between mobile and fixed network with and without their devices. However, possible scenarios in our environment need not rely on such an infrastructure (e.g., mobile ad-hoc collaboration). We have also identified four interaction patterns concerning service provider and service requestor location.

1. The wireless network hosts both requestor and provider. This usually occurs in (mobile) ad-hoc networks. However, infrastructure might exist that aids interaction between peers.
2. The requestor is mobile whereas the provider is situated in the wired part. This is probably the most common setting, opening up wired services also to nomadic users. Furthermore, current tool support for mobile devices is restricted to Web service client development. Therefore, our work focuses primarily on this interaction pattern.
3. The requestor is located in the wired part and invokes a service on a mobile device. This can be used for push services or tracking purposes.
4. Conventional Web service invocation happens as both requestor and provider are located on the infrastructure network.

So far, Web service clients access services from static machines and are bound to these for the duration of service invocation. However, in a mobile environment we cannot expect a device or associated user to remain within transmission range for the full length of a possibly long lasting service interaction. Several strategies are available to handle this problem.

- First, the user can choose to switch to another device that is able to remain connected.
- The user selects an equivalent service that is within reach.
- After the initial service is available again, the user reconnects and starts from the beginning of the service session.
- The user delegates the service request to an intermediary such as agent or proxy in advance. Once reconnected, he/she collects the service response from that intermediary.

Clearly, the first three approaches are unsuitable, as in each case the user has to reinvoke the service. Besides, to repeat all previous steps of service invocation requires time and retransmission of data. This not only increases the load on the

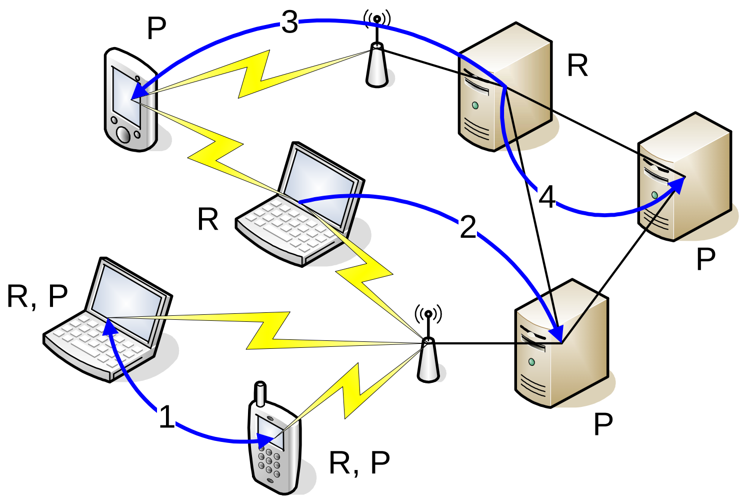

Fig. 1. Ubiquitous Mobile Environment: consisting of wireless and wired connections. R signifies a Web service requestor, while P denotes a Web service provider. Note that provider and requestor can reside on the same node. (1) to (4) indicates the outlined four interaction patterns.

wireless network but also results in the user being charged more as his/her data transfer volume increases[1]. Moreover, in the second case an additional overhead occurs as we need time and resources to find a substitute service. Furthermore, some services might not be exchangeable at all such as personal calendar, document management or infrastructure centric services in general. Our work falls into the last category together with a number of related papers. Section 3 lists existing work and points out the differences to our approach.

The following scenario presents the above-introduced issues and sets our proposed framework in a real world environment.

Mike is a senior manager at ACME. As he is rather busy, he starts working already at home after getting up in the morning. He connects to his company's network from his home computer. All services and resources of ACME are accessible by means of Web services. He requests to get an overview of all currently running projects. As this usually takes a while, Mike decides not to wait for the results at home. A conventional Web service would require him to collect the results from his home PC, but luckily ACME provides the SRR-WS proxy and SRR-WS enabled clients. Usually, he would access the outcome from work, but as Mike is on a business trip to London, he intends to use some spare time on the airport to check up on the results. Just before leaving the house he requests information on the current traffic situation in London. Then, he turns off his PC and heads for the car. While driving his car, Mike usually refrains from using

[1] Here, we assume that the mobile network provider charges by the amount of data transferred neglecting the possibility of a flat rate

his PDA as he considers this too dangerous. After having checked in at the airport, he connects his PDA to the wireless LAN in the business lounge. There he continues the web services he had interrupted by leaving his home. This is made possible as the SRR-WS framework allows to pickup a service session where it was suspended. The project report is still outstanding while the traffic report has arrived. This service, while being external to ACME, can still be access through the SRR-WS proxy. Once at his partner's office in London, Mike connects his Laptop and retrieves the project results.

The scenario highlights the main contributions of our framework.

– Transparency: As there is no need to adapt the service provider, it remains completely unaware of the SRR-WS framework.
– Service independence: Any service client that adheres to the standards can be suspended and resumed as well as relocated.
– Light-weight approach: We require no middleware or agent platform and no proprietary standards are involved. The required infrastructure is limited to the use of the slim SRR-WS framework.
– Continuity across time, space, and devices: Services can be invoked in one place and continued after some time in another place. Furthermore, the utilized devices need not be the same for suspending and resuming.
– Fixed and mobile applicability: The presented framework is not restricted to mobile networks but can also be utilized in wireline networks.
– Strategies for session keep alive: Instead of simply caching a response, different strategies can be employed to enable session resumption at a later stage.

Figure 2 visualizes the underlying ubiquitous environment depicting used devices and network topology.

3 Related Work

Work in this section is focused on the aspect of service continuity and the above-introduced sub problems of service disconnection and service control handover between devices.

Jørstad et al. [2] analyse service continuity in mobile services. They propose a service continuity layer consisting of a monitor, handover manager, service composition module, interoperability evaluator, and I/O redirector. Yet, their work remains at that level and lacks specific details concerning architecture and implementation. Furthermore, their approach does not have the notion of service suspension. Another major difference lies in the task of the handover manager. This component is concerned with switching between different means of transport rather than devices. Finally, their service continuity layer focuses at providing services, whereas our technique requires no changes to the invoked services.

Several agent frameworks and middleware have been designed to facilitate the use of Web services on mobile devices. Below, we introduce approaches which

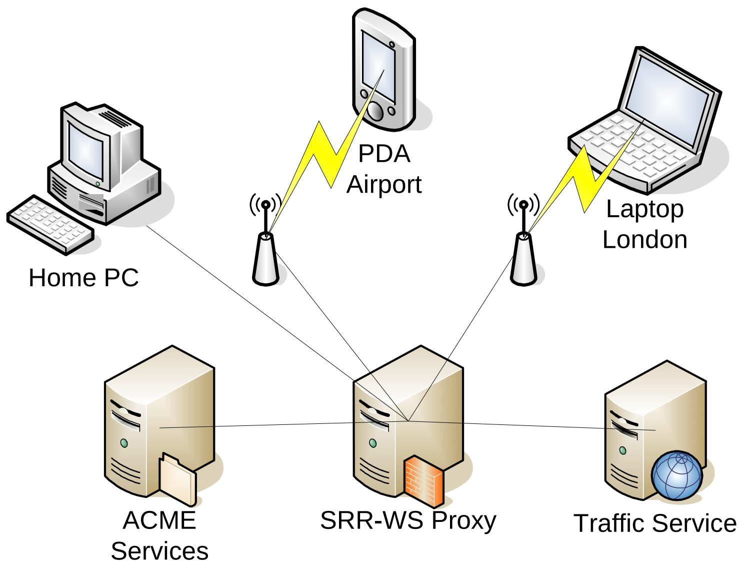

Fig. 2. Scenario Environment: presents a simplyfied version of a real world ubiquitous environment including the various devices employed, the SRR-WS framework and the example Web services.

focus on agents and/or proxies invoking Web services on behalf of a mobile client. Thus, they enable the client to go offline during the service execution.

Yu and Zhang [3] introduce a mobility system based on mobile agents to provide service continuity across networks. Yet, the need for an agent platform narrows widespread applicability. Furthermore, handover between devices is not explicitly given.

In [4], a *client-proxy-multiple server* model is designed where broker agents accept requests from mobile clients. These can resort to a knowledge base on the capabilities of service agents to perform the required task. Service agents belong to a specific problem domain and act as a wrapper to services described through a DAML+OIL ontology. The actual computation is done on the agent platform, on execution platforms or if required and possible on the client itself. Agent-deputies handle disconnections. Zahreddine and Mahmoud [5] and Maamar et al. [6] present similar agent-based frameworks.

Okuda et al. [7] explicitly address continuous service access by means of agents. They adopt the same definition regarding service continuity as we do, namely suspending services, and resuming at a later point in time. Moreover, they also support continuing from a different device. However, their work is focused on multimedia web content that is adapted to different device capabilities and not on invocation of Web services. Hence, their technique varies considerably compared to our approach.

At this point, we can highlight several shortcomings that the presented work hitherto has in common. Agent networks in general lack widespread acceptance due to the need of a respective platform. Furthermore, Web service continuity across devices is only supported implicitly and service interruption not at all. These two aspects are the central focus of Satyanarayanan et al. [8]. However, their approach goes beyond the scope of our work as the presented mechanisms transfer the whole state of the device using a distributed file system. We claim that this technique is too heavy weight and does not take into account unfinished service sessions.

Work on handover is primarily concerned with switching between different kinds of networks (eg. WLAN to GRPS or UMTS). As handover takes place at the transport layer to provide continuous connections for synchronous communication, interruptions constitute an undesirable state. However, this is less of a problem in the case of Web services where asynchronous communication is prevailing. Hence, our approach currently excludes such transport mechanisms but they pose a promising field for future extensions. For example, Calvagna and Modica [9] propose user-centric policies for vertical handover in order to reduce costs for the individual user. Bellavista et al. [10] present a middleware architecture for context aware service deployment and handoff management.

Pilioura et al. [11] analyse Web service scenarios for mobile commerce. In the case of providing stable Web services to mobile devices, they propose a proxy architecture to invoke services on behalf of a client. Although their approach is similar to ours, we point out that the proxy needs to create a stub for every unknown request as their service requestor is assumed to be Web service unaware. This results in much more overhead than simply forwarding a service invocation message. Furthermore, no explicit mechanism to switch between devices is included.

To the best of our knowledge, so far no research effort has been put into designing a generic *pause-resume* mechanism for Web service invocation. Services themselves can be designed to allow for breaks by implementing long timeouts or resending messages, but this is not a transparent approach. Another possibility would be to utilize SOAP over SMTP, which intrinsically includes delays. However, this would require SMTP servers running on the mobile clients, or a SMTP push infrastructure. Moreover, this technique is specific to the underlying transport protocol and outside the scope of the Web service interaction layer. Besides, an SMTP server does not feature any keep-alive strategies and session relocation facilities as the SRR-WS proxy provides.

This pause-resume mechanism mentioned above holds much of the asynchronous notion that is usually associated with Message-Oriented Middleware (MOM). Such systems, however, are usually heavy-weight and restrict clients to connect to services available only within each such MOM. Though, our generic technique is open to any Web service client.

Following papers do not directly address service continuity issues as such, but provide techniques that reduce the need for service suspension and relocation. Yet, our framework is likely to incorporate these approaches at a later stage.

Sen et al. [12] propose to tackle connectivity issues by invoking the Web service that is the most likely to remain available for the duration of the interaction process. A reasoner accessing a knowledge base—that contains motion information on participating service provider nodes—predicts the node's locality at the required place at the required time.

Similarly, Doulkeridis et al. [13] introduce the notion of a context-aware service directory that includes temporal information. They argue, that context related data would enable prediction of the period a certain service remains available.

Friedman [14] who proposes to cache the actual Web services within an ad-hoc network brings up another option. Upon partitions or combinations of network sections, caching proxies are either spawned or combined. In this environment only those services can be rendered ubiquitous that can be transferred or copied as a whole between nodes.

4 The Suspend-Relocate-Resume for Web Services (SRR-WS) Framework

Our approach focuses on service continuity at the client side by introducing the Suspend-Relocate-Resume for Web Services (SRR-WS) framework. It enables to suspend a service, optionally relocate the client's session data, and to continue where the client has paused its interaction. As shown in the section on related work, no generic mechanism for resuming has been proposed so far. Thus, we cannot assume that Web service providers feature suspend or resume capabilities. Hence, the first version of our framework has the requirement that the Web service provider side should be left unaffected and thus the client side has to take provisions for service interruption. The SRR-WS design consists of a proxy—that we kept as generic as possible—and client side session restorement facilities. Figure 3 gives a general overview.

The SRR-WS Proxy is composed of three components: the actual proxy (P-module) that forwards SOAP messages over HTTP to the ultimate receiver, the suspend and resume module (SR-module) as well as the session relocation module (R-module). The internal architecture of the whole proxy is given in Figure 4. The latter two modules are designed as Web services. The SR-module allows the client to explicitly request a connection interruption for going offline or shutting down, while enabling it to resume the session once it reports back. We define a *session* as a combination of client-state information and pending invocation requests. The R-Module permits a client to transfer the current session onto another device for later continuation. This feature is central to achieving service continuity across devices. The SR and R modules operate independently of each other. Thus, suspending and resuming can be executed on the same device, while session relocation can also occur in case no invocation reply is pending. Yet, the combined functionality enables virtually "anytime" relocation.

The SR functionality is most useful for stateful services that require some identifying data to be kept throughout the session. Nevertheless, stateless ser-

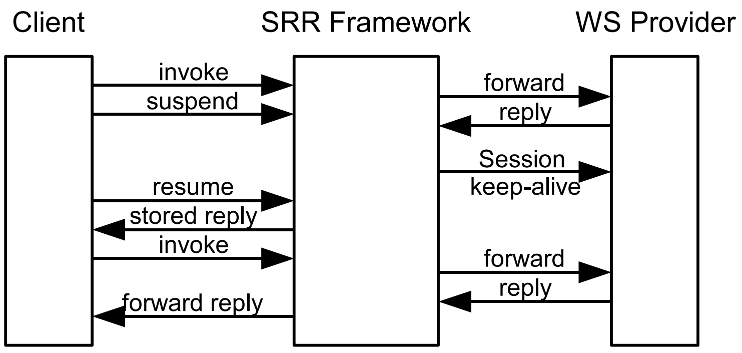

Fig. 3. The SRR-WS Framework consists of a generic Web service proxy with additional capabilities regarding caching and session relocation as well as facilities at the client to enable session suspension and continuation.

vices can also profit from the framework, as asynchronous responses are stored during a client's offline period. This also holds true for services, which exhibit high invocation costs or long execution time.

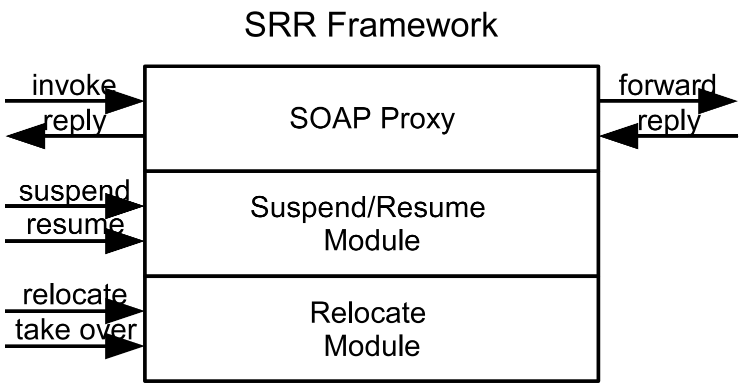

Fig. 4. The SRR-WS-Proxy consists of a SOAP proxy and Web service interfaces for suspending and resuming a process as well as transferring the client's session data onto another device.

What remains to be defined is how to keep a session alive. As we argue that the proxy should remain ignorant of the semantics of any requests, we propose that the client should know how to keep a session running. In invoking the SR interface, the client defines how this should be done. Below, we list possible strategies that we identified as suitable. Note that different strategies can be used at different points in the interaction process.

None the client knows that at this point, the session cannot be kept alive by the proxy itself. Thus, the proxy won't cache the Web service response.

Wait the client indicates that the proxy should just wait for it to report back from being off-line or down and then forward the stored Web service response. This strategy is applied if no timeout is expected to occur.

Replay the proxy should replay the last SOAP request at specific intervals. Always the latest response is kept. *Replay* works the same way as a web browser's reload functionality behaves: retrieving the most up to-date-information while keeping a possible session alive.

Custom the client provides a custom message which the proxy should forward at the given intervals. As in the previous case, the Proxy stores the latest reply from the Web service. The *Custom* strategy behaves just as the *Replay* strategy, while not sending the last message but a different one.

The latter three strategies include an interval within which the client expects to report back. Yet, the proxy can choose to reduce this value, if its "offline time span" is less. In such a case, the proxy notifies the client about the new valid deadline in the immediate reply message. Yet, to avoid side effects, the client needs to know how the Web service will behave for each of these strategies. However, we argue, the more complex a service is, the better the client needs to know it already without using the SRR-WS framework.

Session relocation is kept rather simple. The client describes relevant session data as an XML document and transfers it to the relocation module on the proxy. It also provides a deadline until itself or another client intends to retrieve the data. The relocation session data are stored on the SRR-WS proxy, which returns an identifier for later recovery. Retrieval happens exactly the opposite way. As the client provides the session identifier, the proxy returns the XML document.

On the client side, the framework consists of an additional layer between Web service proxy and transport engine to enhance the header with SRR-WS identification information. Specifically, we use the MessageId of the WS-Addressing [15] header to enable this in a standardized way. The proxy processes this information to establish later which SOAP request should be suspended. This additional SOAP header block is kept even as the request is forwarded to the actual destination as WS-Addressing might be used at the ultimate receiver's site. Figure 5 visualizes this concept on the client device. The displayed components are the client application, WS-Addressing (for providing the necessary correlation headers), the SRR Client Framework (provides functionality for handling suspend and resume as well as relocate requests), WS Client Stub and SRR Client Stub to transform methods into SOAP messages (these are automatically created by an WSDL parser), and finally the HTTP Engine (responsible for wrapping the SOAP message in a HTTP POST request and subsequent transmission to the proxy).

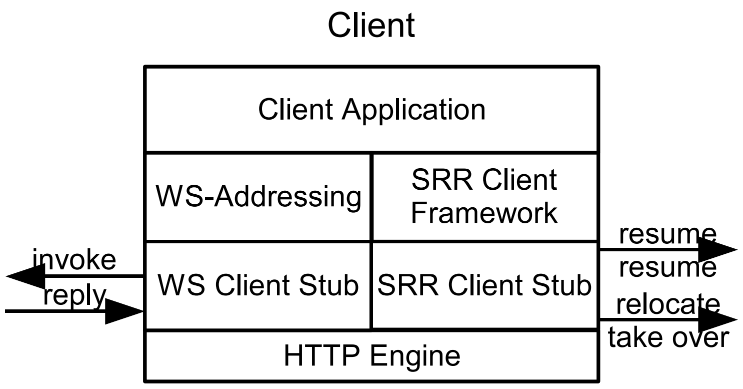

Fig. 5. At the client side the SRR-WS Framework consists of an additional layer to introduce process related session identifier and a stub for accessing the SRR-WS Web service interfaces.

5 Implementation and Results

We implemented our framework using SOAP over HTTP as it is most widely applied. Besides, using SMTP we would not be able to point out the impact our framework has on Web service interaction, as an SMTP server can, in some way, be regarded as an implicit SRR-WS-Proxy. Furthermore, WS-Addressing was our choice of message identification mechanism as it provides this functionality in a reasonably comfortable way, thus no need for another standard or proprietary SOAP header. WS-Addressing is supported by Microsofts .NET Framework 2003 Web Service Extensions 2.0[2] which is also available for PocketPC by means of the OpenNETCF [16] framework. One problem that had to be tackled in using SOAP over HTTP in conjunction with WS-Addressing results from the fact that once we have invoked a request and are waiting for a reply, we have to abort it to suspend. Thus, to receive the reply from the SRR framework upon resuming, we need to invoke the same request again but this time with a new WS-Addressing MessageId. In order for the SRR framework to know, which reinvoked requests desires what stored reply, MessageId mapping is done. This is explained in further detail below when discussing the Suspend-Resume module.

5.1 The Proxy Module

The HTTP SOAP proxy has not been written from scratch but the free Mentalis.org proxy [17] has been adapted to fit our needs. More precisely: we made the proxy SOAP aware, thus processing SOAP requests in a way to enable possible suspend and resume action. For regular HTTP requests, the proxy acts as usual. Once the proxy receives a SOAP request, it checks whether this request

[2] Microsofts Web Service Extensions 3.0 are available as well but incompatible to the OpenNETCF framework.

needs to be forwarded to the ultimate receiver (a new request) or was sent to retrieve a stored reply (a reinvoked request). In the former case, the proxy connects to the given destination and waits for the reply. Once the complete response is received it checks whether the corresponding client has suspended (subsequently storing the message) or not (forwarding the reply the the client). In case of a reinvoked request, the proxy retrieves the stored response and returns it without connecting to the actual destination.

Errors during client-proxy communication have no effect on the proxy's state. Initial requests or invocation of the proxy's suspend or relocate service are simply ignored. If an error happens during session resumption or session takeover, related data is not lost but kept for a new retrieval attempt.

5.2 The Suspend-Resume Module and KeepAlive-Strategy

The SR-module is accessible via two Web service methods for suspending and resuming. To suspend one or several request, the client needs to provide a KeepAlive-Strategy for every single one. Such a strategy corresponds to the types introduced above, namely: *None*, *Wait*, *Replay* and *Custom*. Besides identifying which request is to be suspended (by means of the MessageId), it further includes details on how long the response should be stored (at most), the maximum interval between resuming and actually reinvoking the request, and the custom message (if required), as well as the resend interval where appropriate. For resuming one or more requests, the client submits the mapping between original request MessageId and the one that is going to be used for reinvoking. Hence, the SR module, repectively the SOAP proxy, can distinguish between new and resumed SOAP requests. The client also includes the proxy identifier which is needed in case a network of distributed proxies is used.

5.3 The Relocate Module

The R-module allows session data in the form of a serialized XML document to be stored for a certain amount of time. Upon submission of a session document, a unique identifier is generated and returned to the client for future retrieval. The same client or another one that possesses the identifier, can use it to take-over the corresponding session information. The transfer of session information can also be described as pushing and pulling a session to the SRR-WS framework. We left this module intentionally simple for demonstration purpose and leave it up to the client application to transfer the session identifier to another client instance in case of relocation. The session description in XML is left to the client and the framework provides no support, as we believe session data will vary significantly between applications.

5.4 The Test Run

The testing Web service was implemented as a Google Web service [18] relay that transfers requests from document to RPC SOAP style. However, for simplicity reasons, only the spelling suggestion method is available. The service waits for ten seconds to simulate a longer lasting interaction and returns the Google result[3] marked with a timestamp. The proxy part of the WRR-WS framework runs on Windows XP Professional on the Internet Information Server (IIS). In our test bed we had a Pentium III Laptop as proxy host connected via LAN to another Windows XP machine, hosting the Google relay Web service. Our demonstrations client was implemented on an iPAQ 2210 running Windows Mobile 2003 and accessed the SRR-WS framework proxy by means of WLAN 802.11b in adhoc-mode. Our tests showed that as expected, several invocation requests could be issued, then these requests were suspended, the session pushed to the SRR-WS proxy and the client closed before the PDA was switched off. Within the suspend timeout, we restarted the client applications, retrieved the session data, then resumed the session and received all stored invocation replies. For the four keep alive strategies, we experienced behavior as expected.

- Providing the *None* strategy, the proxy aborted the connection to the test service or—if already available—deleted the reply.
- Requesting the *Wait* strategy, the timestamp in the service reply amounted for about ten seconds more than the time of initial service invocation.
- Demanding the *Replay* strategy for every 20 seconds, the timestamp in the service reply ranged between 10 and 20 seconds less than the point in time of service resumption.
- For the *Custom* strategy having the same values as the *Replay* strategy, we experienced also the same timestamp values.

5.5 Evaluation

Currently, we cannot make extensive statements about the performance of our framework as message overhead heavily depends on the applied keep-alive strategy. Nevertheless, in the following list we give an indication where to expect the most improvements and which strategies need closer attention.

- The *None* strategy is equivalent to experiencing a service interruption without the SRR-WS framework with an additional message to notify the SRR-WS proxy neither to store nor to wait for a reply.
- *Wait* involves one message to inform the proxy to store the reply and another one to resume the invocation.
- The amount of messages for *Replay* and *Custom* cannot be determined without knowledge of the exact strategy parameters. At a minimum, we have two messages (as this case includes the previous one) and at least another one, if the message is resent once only. Otherwise we have the client's offline period divided by the strategy's replay interval. Thus the worst case scenario happens if the client does not report back within the proxies internal "offline time span". This holds true for both *Replay* and *Custom.*

[3] In case a google key is not available, use "test" as the key, and only the timestamp will be returned.

While analysing the expected message overhead, this amount has to be put in contrast with the number of messages needed in case of a new session start.

Concerning scalability, the proxy certainly constitutes a bottleneck. Yet, we argue that a network of distributed SRR-WS proxies can solve this problem. A client can still use a different proxy (ProxyB) for resumption than used for suspending (ProxyA). This is achieved as the client submits the proxy's identifier (as mentioned above). All ProxyB needs to do is forward the request to the initial proxy (ProxyA), which returns the stored reply. This is in turn relayed to the client. Then ProxyB handles all subsequent traffic. Thus, no replication of state information occurs as ProxyA stores the initial reply, and ProxyB merely fetches that information upon the client's request. In such a federation, various hosts such as companies or universities (e.g., ACME in our scenario), network operators (e.g., WLAN hotspot providers), or individuals (e.g., Mike might decide to have one at home) provide proxies. Unfortunately, we lacked time to implement and test this feature.

The need for suspending a service explicitly might seem a like an unrealistic assumption in an adhoc network. However, we argue that this is less of a concern as the framework is not specially designed for an adhoc network but rather an ubiquitous one. Furthermore, users explicitly switch on or off their devices and do so when changing to another device. Moreover, context information can be used to anticipate potential disconnections. Yet, this is outside the scope of our paper.

6 Future Work

Future work will consist on the one hand of improving the demonstration implementation. At the moment a tight integration between client application and client-side framework is required. We plan to aid the developer in providing an automatic wrapper around Web services and a less tightly coupled SRR framework. For example, BPEL processes enhanced with business rules as proposed by Rosenberg and Dustdar [19] define timeouts and session constraints and thus enable automatic selection of appropriate keep-alive strategies and corresponding SOAP messages. Furthermore, direct transfer of session data between clients would be an improvement we intend to implement. On the other hand, broadening the applicability of the SRR-WS framework concerning transport protocols, keep alive strategies, and security is another issue. Combining SOAP transport protocols such as HTTP request and SMTP reply (with mail push) might prove interesting to follow-up. In addition, security capabilities that have been left out for the moment, need consideration and integration. Furthermore, our case studies will also show if the current keep alive strategies are generic but also specific enough for widespread use. The further step would consist of enabling Web services themselves to accept suspend and resume requests. This would allow a better integration of session interruption and would require no additional infrastructure such as the SRR-WS framework. However, for generic use this would need to be the subject of a new Web service standard.

7 Conclusion

In this paper, we have discussed service continuity in mobile ubiquitous environments and presented a motivating scenario. Having listed current approaches to the service disconnection and relocation issues, we introduced our solution. The Suspend-Relocate-Resume for Web Services framework (SRR-WS) acts as a generic proxy for requests and caches invocation responses while a client remains off-line. Furthermore, it enables to start an invocation request on one machine and continue on a client located on another one. Thus, we achieve service continuity across space, time, and devices. Besides discussing our framework, we have presented a reference implementation demonstrating the benefits in a wireless network. Our main contributions are a lightweight framework that acts transparently to service providers while remaining unaware of the actual service invocation content. Moreover, the SRR-WS framework can be used in wireless and wireline environments and introduces strategies to keep service sessions during client's offline period alive. We concluded that our framework remains promising, but needs to be employed in a real world environment for further improvement.

Acknowledgment

Part of this work was supported by the Austrian Science Fund (FWF) project OMNIS.

References

1. Jørstad, I., Dustdar, S., van Do, T.: Service-oriented architectures and mobile services. In: 3rd International Workshop on Ubiquitous Mobile Information and collaboration Systems (UMICS), co-located with CAiSE 2005. (2005)
2. Jørstad, I., van Do, T., Dustdar, S.: A service continuity layer for mobile services. In: IEEE Wireless Communications and Networking Conference. (2005) 2300–2305 Volume 4
3. Yu, Y., Zhang, P.: Service mobility in mobile network. In: International Conference on Communication Technology, ICCT 2003. (2003) 1698–1701
4. Chakraborty, D., Perich, F., Joshi, A., Finin, T., Yesha, Y.: Middleware for mobile information access. In: 5th International Workshop on Mobility in Databases and Distributed Systems (MDDS 2002). (2002)
5. Zahreddine, W., Mahmoud, Q.: An agent-based approach to composite mobile web services. In: 19th International Conference on Advanced Information Networking and Applications, 2005. AINA. (2005) 189–192
6. Maamar, Z., Sheng, Q.Z., Benatallah, B.: On composite web services provisioning in an environment of fixed and mobile computing resources. Information Technology and Management **5** (2004) 251–270
7. Okuda, T., Takano, M., Tajima, K., Shimojo, S., Yamaguchi, S., Miyahara, H.: Realizing continuous and transparent service using agents. In: IEEE Pacific Rim Conference on Communications, Computers and signal Processing, 2001. PACRIM. 2001. (2001) 736–739

8. Satyanarayanan, M., Kozuch, M., Helfrich, C., O'Hallaron, D.: Towards seamless mobility on pervasive hardware. Pervasive and Mobile Computing **1** (2005) 157–189
9. Calvagna, A., Modica, G.D.: A user-centric analysis of vertical handovers. In: WMASH '04: Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots, New York, NY, USA, ACM Press (2004) 137–146
10. Bellavista, P., Cinque, M., Cotroneo, D., Foschini, L.: Integrated support for handoff management and context awareness in heterogeneous wireless networks. In: MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, New York, NY, USA, ACM Press (2005) 1–8
11. Pilioura, T., Tsalgatidou, A., Hadjiefthymiades, S.: Scenarios of using web services in m-commerce. SIGecom Exch. **3**(4) (2003) 28–36
12. Sen, R., Handorean, R., Roman, G.C., Hackmann, G.: Knowledge-driven interactions with services across ad hoc networks. In: ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing, New York, NY, USA, ACM Press (2004) 222–231
13. C.Doulkeridis, Valavanis, E., Vazirgiannis, M.: Towards a context-aware service directory. In: 29th International Conference on Very Large Data Bases (VLDB 2003). (2003)
14. Friedman, R.: Caching web services in mobile ad-hoc networks: opportunities and challenges. In: POMC '02: Proceedings of the second ACM international workshop on Principles of mobile computing, New York, NY, USA, ACM Press (2002) 90–96
15. Box, D., Christensen, E., Cubera, F., Ferguson, D., Frey, J., Kaler, C., Langworthy, D., Leymann, F., Lovering, B., Lucco, S., Millet, S., Mukhi, N., Nottingham, M., Orchard, D., andE. Sindambiwe, J.S., Storey, T., Weerawarana, S., Winkler, S.: Web Service Addressing (WS-Addressing). http://www.w3.org/Submission/ws-addressing/ (2004)
16. OpenNETCF.org: OpenNETCF Website (2005)
17. Mentalis.org: Mentalis.org Proxy Website (2005)
18. Google: Google web service api (2005)
19. Rosenberg, F., Dustdar, S.: Business Rules Integration in BPEL - A Service-Oriented Approach. In: 7th IEEE International Conference on E-Commerce Technology, CEC. (2005) 476–479