Apostolos Papageorgiou, Marius Schatke, Stefan Schulte, and Ralf Steinmetz:
Lightweight Wireless Web Service Communication Through Enhanced Caching Mechanisms. In:
International Journal of Web Services Research, vol. 2, no. 9, p. 42-68, April 2012.

Lightweight Wireless Web Service Communication Through Enhanced Caching Mechanisms

Apostolos Papageorgiou, NEC Europe Laboratories, Heidelberg, Germany

Marius Schatke, Ecodaxi, Blue Moon GmbH - Berlin, Germany

Stefan Schulte, Vienna University of Technology, Austria

Ralf Steinmetz, Technische Universität Darmstadt, Germany

ABSTRACT

Reducing the size of the wirelessly transmitted data during the invocation of third-party Web services is a worthwhile goal of many mobile application developers. Among many adaptation mechanisms that can be used for the mediation of such Web service invocations, the automated enhancement of caching mechanisms is a promising approach that can spare the re-transmission of entire content fields of the exchanged messages. However, it is usually impeded by technological constraints and by various other factors, such as the inherent risk of using responses that are not fresh, i.e., are not up-to-date. This paper presents the roadmap, the most important technical and algorithmic details, and a thorough evaluation of the first solution for generically and automatically enriching the communication with any third-party Web service in a way that cached responses can be exploited while a freshness of 100% is maintained.

Keywords: *Adaptation, Caching, Mobility, Proxy, Web Service, Wireless*

INTRODUCTION

It seems that we are in the middle of an era where the spectrum of computing and bandwidth capacities is being stretched in both directions. At one end, highly capable systems are enabled through the Cloud and the continuous enhancements of the hardware, as well as of the communication media. At the opposite end, more limited, handheld, embedded, and mobile devices are taking over the market in a scale that inspires some people to argue that we are finally entering the "post-PC era," which has already been proclaimed as early as 1999 (Press, 1999). Like any technology that inter-operates between very different systems, Web service technologies must be an efficient solution for systems of both mentioned ends as well as the systems in between these two extremes. For this, they have to be adaptable to the entire computing and bandwidth spectrum in which they are used. Thus, Web service technologies

DOI: 10.4018/jwsr.2012040103

The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, not withstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.

have to focus on the needs of all their main application domains.

Undoubtedly, enterprise systems have been the application domain where Web services gained importance, being the most common technology for implementing Service-oriented Architectures (SOA) (Papazoglou & Heuvel, 2007). However, "everyday apps" might already be an equally important class of Web service consumers. Independently of which is the main application domain for Web services, the involvement of wireless devices as Web service consumers is increasing in both of them. A recent survey of TechTarget (Frye, 2009) positioned Web service-based mobile apps at the second place in the category "service-based implementations planned for the future" (planned by 60% of the questioned developers/companies), even higher than the "composite application assembly" (planned by 58%), which has been often named as the main potential of SOA (Papazoglou & Heuvel, 2007). The popularity of Web service technologies is due to the interoperability and platform-independence that are achieved through the self-description of the interfaces and the messages, but it is exactly this self-description that causes some communication overhead, for which Web services have been criticized since they appeared (Davis & Parashar, 2002). Enterprise systems may be affected by that only rarely but the same is not true for wireless devices.

Although some argue that the constraints of mobile devices (limited bandwidth, CPU, memory, or energy resources) are disappearing due to technological progress and such devices are "riding the wave of Moore's Law" (Christin, Reinhardt, Kanhere, & Hollick, 2011), the gap between communication requirements and such device's capabilities will not cease to exist. This is indicated by the latest analyses of future wireless communications. In the book of Sesia, Toufik, and Baker (2009) about LTE (Long Term Evolution of 3G mobile networks), five categories of user equipment are defined, with smartphones being placed only in the second or third category. According to this categorization, devices of higher categories will be able to use wireless internet connection rates up to six times greater than those of lower categories. Furthermore, the wired connections of the future will be even faster than that, not to mention the fact that devices less capable than smartphones, such as sensor nodes, will be able to consume Web services. So, the big differences in device capabilities and connection qualities will maintain the need for adaptation of communication methods, as the size of the data that is processed and wirelessly transmitted is growing parallel to all other technological developments (Canali, Colajanni, & Lancellotti, 2009).

Therefore, most of the approaches that have appeared for reducing the overhead of Web service communication focus on wireless systems or are even specially designed for them. Client-side caching of Web service responses is such a broadly used lightweight technique and many different algorithms and strategies exist for it. However, all client-side caching algorithms contain some risk of using information that is not fresh, i.e., up-to-date. If one wants to be sure that the freshness of information is guaranteed, a new service request has to be sent. Due to technical restrictions that will be explained in the upcoming sections, (XML-based) Web services always transmit a complete response when they receive a request. Thus, the following research question arose: "How can Web services exploit the caching concept, i.e., the reuse of information from former responses, but with certainty that they are up-to-date?"

In Papageorgiou, Schatke, Schulte, and Steinmetz (2011) we presented a mediator-based solution for enabling freshness-safe client-side caching of SOAP responses. This has been achieved thanks to the innovative idea of enabling the automated and generic (i.e., service-independent) generation of a particular type of caching proxies. The article-at-hand takes the mentioned work further by analyzing the vision that led to the described solution, by offering a description of how the proxy generation logic can be realized, and by providing an extended evaluation that demonstrates the possible benefits of the approach in different Web service usage scenarios.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

RELATED WORK

Caching is only one of many approaches towards lightweight Web service consumption. When caching approaches are developed in this context and with the goal of reducing the amount of data that is being processed and transmitted during mobile Web service usage, they have to be considered in the general fields "Web service performance enhancement" or "Lightweight Web services." Indeed, this is where this work is positioned: Our approach aims at achieving lightweight Web service communication based on caching. However, research for caching can be applied with small changes in many fields including web content (WWW caching), database information, and more. Thus, related work for caching in other domains cannot be ignored. On the contrary, it is one of the primary fields where related approaches have to be searched. Specialized approaches for caching Web service responses have also appeared, indicating some special challenges and particularities of Web service caching. In the following, our approach is first positioned in the research field where it originates from. Then, two classes of caching-related research works (general-purposed and SOA-related) are discussed, leading us to the identification of the gap that has not yet been successfully addressed.

Positioning Our Approach

The listing of the most important approaches that have been designed to reduce the amount of data processed and/or transmitted during Web service calls is here out-of-scope and has been partly covered by our previous survey (Papageorgiou, Blendin, Miede, Eckert, & Steinmetz, 2010). Instead, it is meaningful to briefly summarize the results of the mentioned survey, as well as the results of a more recent survey of SOAP processing enhancements (Tekli, Damiani, Chbeir, & Gianini, 2011), in order to explain why we decided to pursue the use of caching as a Web service adaptation mechanism for lightweight Web service consumption.

The most relevant of the adaptation mechanisms examined in Papageorgiou et al. (2010) fall into one of the following categories: (i) reduction of redundancies, as in Oh and Fox (2006), (ii) on-the-fly protocol transformation (SOAP-to-X), as in Aitenbichler, Kangasharju, and Mühlhäuser (2007), and (iii) compression (or optimization of the representation), as in Tian, Voigt, Naumowicz, Ritter, and Schiller (2004). Obviously, the use of caching in the form of an adaptation mechanism would be a category by itself. A proof of this is that all the approaches listed in Papageorgiou et al. (2010) have calculable upper limits for the "message size reduction ratio" that they can achieve. Such limits do not exist for an adaptation mechanism that is based on caching, as will become clear in our evaluation.

Tekli et al. (2011) is not limited to the examination of possible adaptation mechanisms, but it rather considers all SOAP performance enhancement approaches. The difference is that some of the approaches described there may not be applicable without changes on the side of the original Web service, i.e., may not make much sense when external, third-party services are considered. However, this survey helps us position our work more precisely, because it defines six levels (i.e., six points/phases of the "lifecycle" of a Web service call), where the focus of the enhancement may lie. These are: serialization, parsing, de-serialization, security policy evaluation, compression, and multicasting. Although approaches that are limitless in terms of the "message size reduction ratio" may appear there (a corresponding analysis is not provided), it is self-evident that enhancement approaches with different focus make sense under different circumstances, depending on the goals of the system. There, a client-side caching approach is listed as an enhancement of the serialization phase, because it does not always reconstruct and, even better, does not always retransmit identical requests. However, if we do not consider caching as a client-side optimization but as an adaptation mechanism of the service itself, i.e., if we do not use the idea of caching in order to avoid connection

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

establishments, but in order to reduce the size of messages, then we have an approach that falls under a different category, creates new potentials, and achieves different goals, as we will see in the following.

Caching in General

Much of the research effort in the area of caching has been devoted to the development of cache replacement strategies (Podlipnig & Böszörmenyi, 2003). Such strategies concern the maintenance of the cache and their goal is to retain in the cache the entries that are most likely to be needed again soon, i.e., to maximize the *cache hit ratio*. Although most of the caching approaches are based to some extent on the classical strategies "Least Recently Used" (LRU) (cf. Karedla, Love, & Wherry, 1994) and "Least Frequently Used" (LFU) (cf. Karedla, Love, & Wherry, 1994), the research interest in the field is still alive. New approaches are still being developed, exploiting the characteristics of particular technologies in order to be more efficient. For example, Jelenkovic and Radovanovic (2008) recently developed a replacement policy that has better performance than LRU and less complexity than LFU in the case of Zipfian request probabilities and big cache sizes. Cao (2002) has enhanced cache management for mobile computing systems based on a concept for adaptively prefetching the content.

However, the hit ratio is irrelevant to the *freshness* of the cache information. Thus, if the usage of up-to-date information is of high importance, additional techniques have to be used in order to update the cache regularly. While the aforementioned solutions focus on the cache hit ratio, the state-of-the-art solution to optimize the cache freshness is the use of server Invalidation Reports (IR) (Cao, Zhang, Cao, & Xie, 2007). In that case, the servers indicate the changed data items to the clients at intelligently-determined intervals. Otherwise, the freshness normally just relies on "Client Validation," which means either "polling every time," or defining an adequate Time-To-Live (TTL) for the cached objects, as is explained in more detail in the survey of Cao and Özsu (2002).

Either because of their scope (i.e., they often do not consider freshness at all) or because it is impossible to apply them "as they are" in an arbitrary domain, these general-purposed works may be very important and fundamental, but they are not always sufficient, when it comes to the caching of Web service responses. This is explicitly argued in related surveys (see next paragraph) and implicitly proven by the fact that specialized approaches for the caching of Web service responses are explored.

Caching Web Service Responses

By studying the particularities of Web service response caching (i.e., questions like: what are the technical differences compared to caching in other domains? What phases of Web service calls are the most resource-consuming? What are the alternatives for the representation of cache responses?), some researches came up with specialized solutions for the field. Before proceeding to the description of proposed solutions, we refer to one of the most detailed analyses of the peculiarities of caching standard, XML-based Web services for mobility (Terry & Ramasubramanian, 2003). A demand for new technical solutions and standards, rather than algorithmic extensions, has been identified, because most of the challenges were related to technical enablements, and not to enhancements of algorithmic efficiency. Thus, implicit directives such as those provided by Terry and Ramasubramanian (2003) led researchers to suggest three different types of solutions: *provider-based*, *mediator-based*, and *client-based*. Provider-based solutions rely upon logic or modules that are added to the host of the original Web service and thus assume access to the provider system. Mediator-based solutions need modules that are hosted separately from both the provider and the consumer. Client-based solutions enrich only the client-side (usually with some client-side caching middleware) and do not need any other entity elsewhere. The categorization is not perfectly unambiguous,

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

but it depends on where the main components or the important caching logic of each approach are located and it helps to identify aspects that each class of solutions cannot address.

Provider-based solutions have the major drawback that they need modifications of the original service (or of its hosting system). Because of our target scenario, which will be described in the following section, something like this comes out of question and our solution will be able to work with existing (probably third-party) services without "touching" them. However, some techniques and ideas may be common with ours. Liu and Deters (2007) describe an approach based on metadata that indicate if Web services are cacheable or not. The important issue of cache performance vs. cache freshness (or consistency, as it is called there) is mentioned, but without any attempt to maximize the two properties at the same time. Further, the authors focus on the case of "connectivity loss," which will not be our focus, while they also rely upon provider-side metadata and modules. Li, Zhao, Qi, Fang, and Ding (2008) present a concept that is closest to ours more than any other work. When responses have been previously delivered to the same client, the provider responds with a hashcode instead of retransmitting an identical complete response. Unfortunately, it is not clear if the provider-side middleware (called *SigsitAccelerator Proxy*) could work outside the provider system. Furthermore, the proxy might become a bottleneck, as it seems as if there is only one dedicated proxy for all backend services. Even if the above problems could be solved, we would still be missing a proof that the proxy could be generated for any Web service and a technical description of how this could be achieved in an automated manner. Further, the solution of Li et al. (2008) is not designed for wireless communication. As will be seen later, the latter aspect will affect the nature of our proposed solution a lot. Some interesting ideas about efficient caching based on metadata-based knowledge about how response messages are generated is presented in Tatemura, Po, Sawires, Agrawal, and Candan (2005), but this work is, again, strongly bound to a new architecture and new standards for the provider, while the freshness of Web service responses is not considered at all.

Although many of the referenced works mention the use of proxies, they are not considered here as *mediator-based* if the proxy is actually part of either the provider- or the client-system or network (which is often the case). In order to consider an approach as mediator-based, it must be possible (and it also must make sense) to physically separate the mediator (or proxy) from both the provider- and the client-system. Schreiber, Aitenbichler, Göb, and Mühlhäuser (2010) present an efficient mediator-based technical solution about how the caching of mobile Web services should be handled when processes are concerned. Processes can be learned and responses can be prefetched. Nevertheless, prefetching is obviously forbidden when the goal is 100% freshness. CRISP (Elbashir & Deters, 2005) can be also used as mediator-based solution, as its architecture enables method-call interception without further requirements at the provider-side. However, the existence of two modes ("consistency mode" and "performance mode") proves that the constraint mentioned for Liu and Deters (2007) is present here, as well. Our approach is going to be mediator-based but it differs from the referenced approaches in that the proxies at the mediator-side are generated automatically and separately for each service and, of course, in that it will guarantee 100% freshness.

Client-based solutions attempt to enhance the way a client can store, represent, handle, replace, and re-use identical requests and responses. However, when a message-exchange *does* occur, it conforms exactly to the message-exchange that occurs in the absence of any caching mechanism. Within the context of our work, Takase and Tatsubori (2004) present the most similar client-based approach. There, the idea of using "HTTP-like" validity-checks in order to enhance freshness is theoretically mentioned, but not further handled. It was not considered from a technological point of view and it was, of course, not generically enabled for existing Web services. The work "assumes

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

that it is the responsibility of the client application administrator to configure a Time-To-Live (TTL) for each operation." The existence of TTL is a proof that the risk of using outdated data is accepted. A very important point of that work is the idea to compare application objects instead of XML messages. We will embrace this idea, but we will implement it in a more generic manner, because the modules that compare the responses will not be built once statically, but they will be built automatically for every Web service, thus being able to handle every application object. We cannot tell if the cache of Takase and Tatsubori (2004) uses some other way in order to be able to handle every application object. Many of the previously referenced approaches compare XML messages. This is slower and reduces the probability of a hit (or "match") because of small, unimportant, content-irrelevant differences that may appear in an XML message.

The Gap and Our Contribution

Putting it all together, caching may be applied in different domains (databases, WWW content, Web services etc.), may have different goals (reduction of used bandwidth, reduction of user-perceived latency, reduction of server load, confrontation with connectivity loss etc.), and may be subject to different constraints (minimum hit ratio, minimum cache freshness levels etc.). The conclusion drawn by the analysis of related work is that in the domain of Web services, *no* existing solution can achieve the goal of *reduction of bandwidth and user-perceived latency* while simultaneously satisfying the constraint of *100% freshness*, i.e., always using up-to-date responses. The use of cached or prefetched responses without establishing a connection to the server each time leads, by definition, to a risk of using out-of-date information. Following these approaches, whenever 100% freshness is desired, caching must simply be deactivated. The authors of Schreiber et al. (2010) explicitly state that caching or prefetching of critical responses must be avoided. This statement is true, unless the responses are verified before use. This verification concept will be the basis of our solution and will be explained in the next section, following the description of our scenario. To the best of our knowledge, the approach that we describe is the very *first* approach that allows for using cached Web service responses with guaranteed 100% freshness generically, i.e., for any Web service, even if it the latter is provided by a third-party.

BACKGROUND AND SCENARIO

In the following, the context that motivated our work is briefly described. Then, a current constraint for the caching of Web service responses is described, along with an explanation of why the generic withdrawal of this constraint is just now starting to be motivated in modern service-oriented landscapes. Our vision is to be able to automatically withdraw this constraint from any Web service without having access to its hosting system. Thus, the third subsection explains how the communication with a Web service can look like if our vision is achieved.

Mobility Mediation in the Internet of Services

New service description specifications such as the Unified Service Description Language (USDL) (Cardoso, Barros, May, & Kylau, 2010), which include the business and operational aspects of services in addition to their technical details, turn Web services into perfectly tradable goods and lead to the so-called Internet of Services (IoS) (Oberle, Bhatti, Brockmans, Niemann, & Janiesch, 2009). In the IoS, a great number of Web services offered by different providers through global service marketplaces co-exist with a big set of client devices with different features. As the trading and consumption of the services will have a loose relationship to their development, limited mobile clients will often need to use Web services that are not specially designed for them. For this reason, we are concerned in our work with the development of a Mobility Mediation Layer (MML) for the IoS (Papageorgiou, Leferink,

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Eckert, Repp, & Steinmetz, 2009). Complementary to other tasks, such as automated inter-application context-enrichment, the MML supports proxied consumption of services that are available on the IoS-marketplace in order to perform overhead reduction for the wireless transmission. To achieve this, approaches like those analyzed in Papageorgiou et al. (2010), e.g., protocol transformation or compression, are employed, along with further techniques, such as service-specific, personalized cropping of irrelevant data.

Thus, the MML is also concerned with supporting wireless clients with the caching of service responses. How would traditional caching work? Naturally, every client can store responses for future usage. However, there are three main reasons why the client may avoid doing it:

- Frequent changes: The content changes so frequently that client-side caching does not make any sense.
- Criticality: The service call is so critical for the user that she/he needs to be 100% sure about the validity of the content, even if the old response has good chances of being up-to-date.
- Legal issues: Caching may be implicitly or explicitly stated to be illegal for a particular service.

In a global marketplace, it is difficult or unusual to know about these features, unless relevant information is included in a description such as USDL. But even if it is known that one of the above statements is valid, the best that the user can do is *always* request a new response.

An Important Constraint of Web Service Caching

As already identified shortly after the appearance of Web service technologies, XML-based Web services present many technical challenges and cannot be involved in a caching process similar to the one used for simple Web content (e.g., pure HTML) (Terry & Ramasubramanian, 2003). Based on directives provided by Web servers and supported by Web browsers, simple Web content is not reloaded each time it is requested, if the old (cached) content is still valid. The main technical reason why this mechanism cannot be transferred to the Web service technology is that Web services do not just rely on the HTTP-GET method for their communication, but they implement a rather more complex message exchange in order to export a diverse set of operations. As a result, Web services *always* transmit the complete response when they receive a request. The only alternative in order to support a validity check is to manually implement it inside the logic of each service. A survey among 20 services provided at www.webservicex.net (the 10 most popular and the 10 most recent) has revealed that not a single one of them has inherently implemented such a logic. The reason is obvious: the individual implementation effort outweighs the benefits, while the latter may often concern a limited number of clients. But what about a generic solution that could add this feature to any Web service in the IoS? Why has research not focused on it until now and what exactly would it enable?

In fact, the MML is one of the first systems confronted with a scenario where such a generic “browser-like” solution would be worth its effort, because:

- The MML has access to a huge amount of services over the global marketplace. Some of them would benefit from being enhanced with support for safe client-side caching. However, the MML cannot extend them directly with such logic, because it has no access to their code or their hosting systems.
- The MML target group consists exclusively of wireless clients, while the service providers are often more concerned about larger PC-based clients.
- The MML can estimate better which services (or their clients) would profit more from client-side caching, because of extra information about “cacheability” that it can retrieve from USDL descriptions.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

However, a general-purpose mechanism should be able to function even without this extra information.

- The adaptation of Web services for mobility is the business model of the MML. Thus, the access either to the proxies generated by the MML or directly to the MML software in order to generate own proxies would be tradable goods in line with the MML business model.

Envisioning a "Browser-like" Caching Approach for Web Service Responses

For these reasons, the MML needs a mechanism which, when applied to any external Web service, allows the mobile clients to perform service calls that have the chance to be satisfied with very small, lightweight responses, whenever there is no reason to retransmit lots of data. Upon receiving responses with particular codes in their content (for example, 304, as in simple HTTP calls), clients would know that their cached data was still up-to-date. This would not eliminate the need for establishing a connection, but could significantly reduce the wirelessly transmitted data. There is no way to be 100% sure of the validity of the data without establishing a connection each time that the data is needed, and this "100%" is a fixed goal of ours. The principal difference between typical Web service communication and the communication scheme that our solution will enable is explained in the following with the help of an example.

Figure 1a shows a SOAP request / response pair of an example Web service that informs its invoker about recorded noise incidents at a particular place / address. Such a response may contain binary data, resulting in very big content sizes. Of course, this is only an example. Responses that do not contain binary data may become very big, as well. Currently, with the typical implementation of Web services, the transmission of a request such as this of Figure 1a leads *inevitably* and *each time* to the transmission of a complete response.

Our vision is to enable a communication scheme where it can be automatically recognized if the response contains any data that would be "new" for the invoker. In that case, instead of receiving a response such as this of Figure 1a at every invocation, the invoker would sometimes receive a response that signals the fact that the invoker still has up-to-date information from a previous invocation and can thus reuse it. An example response is shown in Figure 1b, making it obvious that there is a potential for much more lightweight communication.

Enabling request / response pairs such as these shown in Figure 2 normally requires reengineering and reprogramming of the Web service. However, the MML needs a generic solution which somehow enables the described vision for *any* third-party Web service *without* access to its code and can be used by the clients without having to bother about any of the obstacles listed previously (frequent changes, criticality, legal issues). Note that although the MML-scenario is used to explain our motivation and to support the technical understanding, the presented solution is general purpose and will be surely very useful in other scenarios.

THE WEB SERVICE PROXY GENERATION

The described vision is enabled by a solution called Web Service Proxy Generator (WSPG). In addition to proxies for caching, the WSPG can also generate other proxies for overhead reduction (e.g., protocol-transformation, compression), but the latter are out-of-scope of this paper because they are based more on existing technologies, rather than on new concepts, techniques and evaluations, as is the case with caching. To provide a good understanding of the (caching) WSPG, an overview of its functionality is given along with a description of the resulting technical landscape. After the presentation of the most important details about its logic and its algorithms, a cost-benefit analysis is presented, which can also be used for

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 1. Visualization of the vision of "browser-like" caching for Web services

Example SOAP Request

```
<soap:Envelope>
.........
<soap:Body>
        <getStreetNoiseStatus xmlns=".........">
                <param1> Rundeturmstrasse </param1>
                <param2> 64283 </param1>
        </getStreetNoiseStatus>
</soap:Bod   y>
.........
</soap:Envelope>
```

Example SOAP Response

```
<soap:Envelope>
.........
<soap:Body>
        <getStreetNoiseStatusResponse xmlns=".........">
                <incident>
                        <date> … </date>
                        <time> … </time>
                        <log> … </log>
                <noiseRecording      InBytes>................................
                ..........................................................
                ..........................................................
                </noiseRecordingInBytes>
                </incident>

                ..........................................................

                ..........................................................
                <incident>
                        ................................
                        ................................

                </incident>
        </getStreetNoiseStatusResponse>
</soap:Body>
.........
</soap:Envelope>
```

a) SOAP Request/Response pair with the typical Web service communication scheme

Example SOAP Request

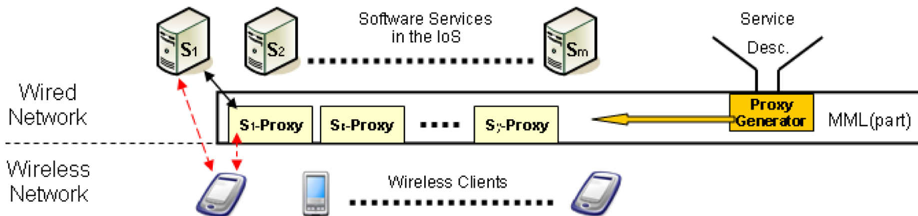
```
<soap:Envelope>
.........
<soap:Body>
        <getStreetNoiseStatus xmlns=".........">
                <param1> Rundeturmstrasse </param1>
                <param 2> 64283 </param1>
        </getStreetNoiseStatus>
</soap:Bod   y>
.........
</soap:Envelope>
```

Example (short) SOAP Response

```
<soap:Envelope>
.........
<soap:Body>
        <getStreetNoiseStatusResponse           xmlns=".........">
                <cachingCode>     304   </cachingCode>
        </getStreetNoiseStatusResponse>
</soap:Body>
.........
</soap:Envelope>
```

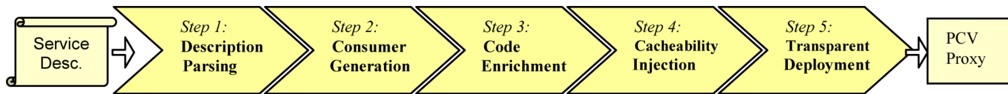
b) SOAP Request/Response pair with the envisioned Web service communication scheme

Figure 2. Example technical landscape after some proxy generations



Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 3. Workflow of the proxy generation logic



deciding for which services the proxy generation is worth its costs.

The Technical Landscape

The WSPG is a module of the MML and should not be confused with the actual proxies that it generates, which are then deployed in Web containers of the MML. Figure 2 shows a possible technical landscape after the generation of some proxies. As shown in Figure 2, after the generation of a proxy (here: a proxy for service S_1 of Figure 2), the direct communication of the device with the service (long dashed arrow) can be replaced with a proxied consumption, where the backend call is performed between IT systems over fast connections (short direct arrow) and the wireless call (short dashed arrow) has the option of getting a very short answer, as envisioned in the previous section and depicted in Figure 1.

Figure 2 is limited to presenting the broader technical setting and abstracts from the most important part, which is the internal logic of the WSPG. As depicted, the proxies of Figure 2 are generated automatically and based only on the service descriptions thanks to the logic provided by the WSPG, which is elaborated in the following.

The Logic of the Generic Proxy Generation

Figure 3 shows the steps of the workflow that is used in order to achieve the automated proxy generation. Each step will be described separately. Most descriptions are provided from an engineering point of view, though some helpful references to details of the Java-based implementation are made. The highlights are to be identified in the conception of the workflow itself and in the innovative code adjustment techniques (esp. of Steps 3 and 4), but also in the difficulty of implementing such a logic generically.

Step 1. Description Parsing: All the necessary information, such as operations, data types, interfaces, ports etc., is read from the service description. Because specialized information is needed, but also because the parser has to be extensible in order to fully support USDL (in addition to WSDL), the WSPG does not use an off-the-shelf service description parser but a new one. A part of its logic, which is important for our scenario, is explained later, based on an algorithmic extract.

Step 2. Consumer Generation: This step generates the part of the proxy that is responsible for calling the original Web service. As no sophisticated or scenario-related actions have to be performed during the consumer generation, the standard tool *wsimport* is transparently and automatically used by the WSPG for this step.

Step 3. Code Enrichment: The code generated by *wsimport* is different for each service. Each time, the WSPG has to dig deep into that code in order to find the points that have to be enriched. Then, it automatically modifies / adds code as needed. For example, (i) the addition of extra parameters in the request / response wrappers and (ii) the adjustment of the interface classes, which determine what the new WSDL (i.e., the WSDL of the proxy) will look like, are two important code enrichment actions.

With regard to (i), a request (or response) wrapper is a class that determines what the SOAP request (or response) contains. After detecting these classes by analyzing the annotations of their code, the WSPG

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

inserts the field *ifModifiedSince* to the request wrapper and the fields *statusCode* and *identTag* to the response wrapper, always preceded by the annotations that are necessary for their inclusion in an XML message and accompanied by the corresponding *getters* and *setters*. This allows for a communication pattern such as the one presented in Figure 1 and for a validity check implemented based on the comparison of *ifModifiedSince* parameters with previously saved *identTag* parameters. As for (ii), the WSPG has to detect the interface classes and replace pairs of RequestWrapper / ResponseWrapper annotations that are included there with the annotation: @javax.jws.soap.SOAPBinding (parameterSytle=ParameterStyle.BARE). This allows for the use of extra parameters (see previous paragraph) without having to change the Web service signature, i.e., without additional wrapper classes for the new service.

Step 4. Cacheability Injection: Similar code adjustments are also needed for the cacheability injection. However, this step concerns the actual handling of the cache, i.e., of the classes that access it or support it. Its logic is separate from the previous step, and most of the results of the code enrichment actions of Step 3 are preconditions for the cacheability injection.

The most interesting actions of the WSPG during this step are the insertions of appropriate annotations to all the entities that must be persistable, i.e., are involved in the storage of request / response pairs, as well as the automatic generation of two further modules for each operation: A *builder* module and a *controller* module. The first is capable of storing request / response pairs so that validity checks can be performed, while the latter can search among the mentioned pairs and compare new responses with cached ones, in order to decide whether the new response or just a status code should be sent back to the wireless client.

Step 5. Transparent Deployment: In this step, not only the code that results from the execution of Step 4 has to be packed and deployed but also the file structure and the configuration files of the Web container that will host the proxy have to be transparently adjusted, so that the generated proxy becomes automatically available as a new Web service.

Important Details of the Proxy Generation Process

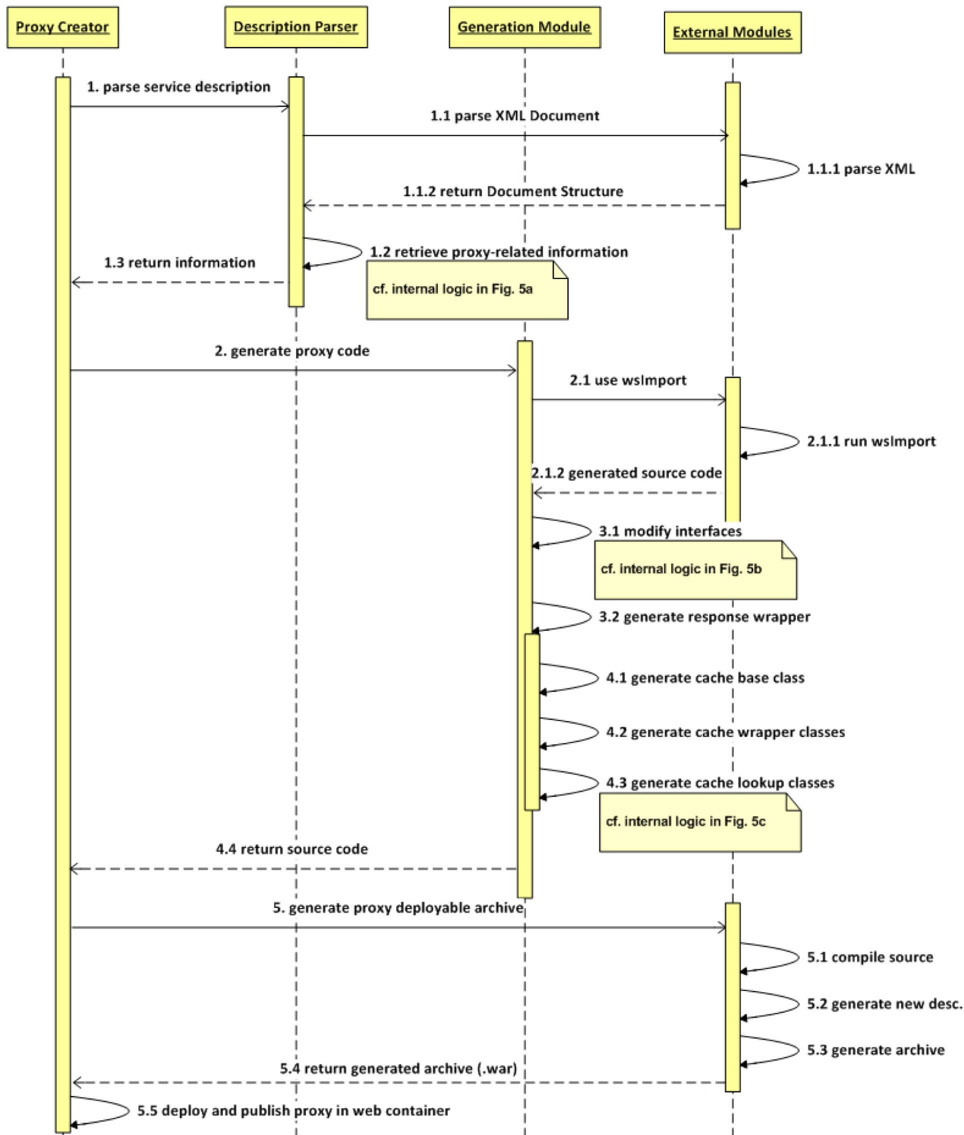
In order to make the proxy generation process more comprehensible and traceable, a sequence diagram is presented, along with extracts of algorithms that highlight and further explain some important procedures.

The sequence diagram of Figure 4 reveals how the proxy generation program runs through concrete actions of the five workflow steps. An understanding of the workflow described in the previous subsection is necessary in order to study the sequence diagram, while algorithms that refer to the actions 1.2, 3.1, and 4.3 are provided separately in Figure 5. The depicted entities should be understood as follows:

- The *Proxy Creator* is the module that coordinates the process and stores globally needed information. It also arranges the generated software parts, ensuring thus the easy integration of the generated proxies into the system(s) where they will have to operate.
- The *Description Parser* analyzes the service description. As shown in the sequence diagram, it is used by the Proxy Creator and, apart from integrating an XML parser, it has no further interactions with other modules.
- The *Generation Module* is the module that actually implements all the complex tasks of the workflow steps 3 and 4. After initiating the consumer generation, it generates various classes, the most important of which are shown and explained in the diagram.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 4. High-level sequence diagram with the main entities of the WSPG



- The *External Modules* include all third-party libraries as well as any non-source-code-related components that are used throughout the process (compilers, tools etc.).

The purpose of the algorithms of Figure 5 is to explain the internal logic of important actions, because this logic does not become obvious from the sequence diagram alone. Furthermore, some of these actions are closely related to aspects that distinguish our work from the state-of-the-art. The algorithms are high-level and coarse-grained, as well, and the provided parts may correspond with hundreds of lines of code in the actual implementation:

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 5. Internal logic of selected proxy generation actions (high-level pseudo-code)

```
1 // descService: the variable in which the XML parsing result is stored
2 // service: the variable in which the necessary info will be stored
3 // '$' means "value of…" and ':' means "element of…"
4 parse_XML_description
5 if valid_description_document
6     service.name = $(descService.name)
7     service.package = $(targetNamespace).toPkgName
8     for_each binding:descService
9         endpoint.name = $(binding.name)
10        for_each operation:binding
11            params<name, type> = operation.findAllParams
12            requestWrapper = find_wrapper_pkg/type_in_descService
13            responseWrapper = find_wrapper_pkg/type_in_descService
14            endpoint.addOperation(name, $(requestWrapper.class),
$(responseWrapper.class), params)
15        end for
16        service.addEndpoint(endpoint)
17    end for
18 end if
19 return service
```

a) Retrieve proxy-related information

```
1 // src: the variable in which auto-generated code is stored
2 // service, endpoint, operation: see pseudcode a)
3 // '$' means "value of…" and ':' means "element of…"
4 src = " $std_package + $(service.package) "
5 src += " import ......" // import all Web service models/annotations
6 src += " web_service_annotation " // (e.g. @WebService)
7 src += " class $(service.name) "
8 for_each operation
9    src += " request_wrapper_annotation "
10   src += " $(requestWrapper.class) " // ...of the current operation
11   src += " response_wrapper_annotation "
12   src += " $(responseWrapper.class) "// ...of the current operation
13   src += " set_ParameterStyle_to_BARE " // to allow modified wrapper
objects to transport additional information like "ifModifiedSince" etc.
14   src += " public $(responseWrapper.class) $(operation.name)() { "
15   src += " modified_method_impl "
16   src += "}"
17 end for
18 wirte_$src_into_new_file
```

b) Modify interfaces

```
1 // src: the variable in which auto-generated code is stored
2 // service, endpoint, operation: see pseudcode a)
3 // '$' means "value of…" and ':' means "element of…"
4 for_each operation:endpoint
5   src = " $std_package + $(service.package) "
6   src += " import ......"
7   src += " class $(operation.name).toCacheName() extends
CacheBaseClass"
8   src += " List<(operation.name).toRequestWrapperName,
(operation.name).toResponeWrapperName> cached; "
9   src += " constructor() {cached <- popupate_Cache()} "
10  // requests and responses in the following contain
11  // ifModifiedSince and IdentTag attributes etc.
12  // NOTE: Talking about requests & responses is a further
13  // abstraction in order to explain the logic of the
14  // auto-generated method lookupCache.
15  // Don't forget that for each operation, requests and responses are
16  // objects of different data types
17  src += " lookupCache(request) { "
18  src += "   if compare_iteratively(request, x:cached) = true "
19  src += "     if mediator_offline_caching_allowed "
20  src += "         return Code_303 " // mediator had recent response
21  src += "     else "
22  src += "         response = call_Service(request) "
23  src += "         if (response = response_in_cached) "
24  src += "             return Code_304 " // client response was fresh
25  src += "         else "
26  src += "             add_to_cache(response) "
27  src += "             return response "
28  src += "         end if "
29  src += "     end if "
30  src += "   else "
31  src += "     response = call_Service(request) "
32  src += "     add_to_cache(response) "
33  src += "     return response "
34  src += "   end if "
35  src += " } "
36  // The lookup class for each operation is stored separately
37  Create_directory_based_on_ $std_package+$(service.package)
38  wirte_$src_into_new_file
39 end for
```

c) Generate cache lookup classes

- Figure 5a shows how our description parser preprocesses and stores the necessary information in a way that will optimally support the next steps of the workflow. For example, line 7 prepares the package structure that will be needed based on the target namespaces found in the service description. If the packaging was not based on the target namespaces the way it is done, it could lead to the automatic generation of wrong or inconsistent code, which would not compile. The next lines gather and store the operations-related information (name, wrappers, parameters) exactly as they will

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

be needed later for the code enrichment and the cacheability injection. Such actions are not included in state-of-the-art (WSDL) description parsers.

- The way in which the WSPG digs into the generated code in order to add the annotations and the code that will determine how the new service interface and, accordingly, the new service description (i.e., that of the proxy) will look like, is shown in Figure 5b. Lines 9-12 indicate, for example, how wrapper annotations are added directly before the declarations of class variables. As can be seen, the types of these variables are those that have been identified by the parser (cf. Figure 5a).
- A closer look at the automatic generation of the class that performs the actual comparison of Web service responses (Figure 5c) is also interesting. Contrary to the generation of the "caching base class," which is similar for all services / operations, the "cache lookup classes" have many operation-specific parts. This makes it very difficult for the *Generation Module* to perform its task generically. The pseudo-code in Figure 5c summarizes (among others) how (i) the generated code must use the particular request / response wrapper objects that correspond with the target operation (e.g., line 8), (ii) a constructor must be built that is able to populate the cache by using operation-specific SQL queries (line 9), (iii) the actual comparison will later be done by code that does not compare XML documents but response objects (lines 18-34). As explained in the "Related Work" section, this is an important aspect, which is very often implemented differently by related approaches. The comparison of objects makes the cache lookup more efficient and bypasses the undesired possibility that small changes in non-content-related details of irrelevant XML parts (e.g., headers) lead to retransmission of actually identical responses. The code 303 (line 20) is returned when the mediator thinks that the response the client already possesses is still fresh. This feature is very useful when the calls to the original service need to be reduced, but it is out-of-scope of this paper. It is only code 304 (line 24) that can guarantee freshness of the response that the client has.

Cost-Benefit Analysis

The proxy can be generated and deployed in up to a few seconds. Because the proxy is generated once a priori (e.g., during service registration) and not during a service call, this cost is trivial. However, there are other reasons why the generation of proxies for all known services may be impossible or undesirable. For example, the proxy-hosting platform may have limited capacity (it should be considered that a huge number of services shall exist on the IoS), the administrator may want to avoid having too many open ports or public interfaces because of security or management issues etc. Therefore, an estimation of the expected benefit should be supported by the WSPG.

The "total bandwidth saved by a proxy generation" (*tbs*) is exemplarily used here as the benefit metric because of its simplicity. Different metrics can be used, resulting in different models. If *hits* are the service calls for which the cached data can be used (i.e., 304 is returned) and *misses* are the service calls for which the complete response must be retransmitted, then

tbs = avg_bandwidth_saving_of_hit
* number_of_hits
– *avg_bandwidth_loss_of_miss*
* *number_of_misses* (1)

which, if we define s_{avg} as the average response size, r as the cache hit ratio ($0 \leq r \leq 1$), s_{304} as the size of the minimal response that corresponds with a hit s_{304} > "size of the extra information needed in a proxy response", and N as the total number of calls, gives after some simple calculations:

$$tbs \geq (s_{avg} * r * N) - (s_{304} * N) \quad (2)$$

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

In order to assign values to these variables for particular services, the WSPG has two options. First, the expected values, e.g., for *r* or s_{avg}, could be read from an extended description such as USDL. Our related suggestions are being examined from the developers of USDL. Second, estimations can be performed based on information from standard descriptions. For example, the amount of parameters of an operation, as well as their types and name lengths, let the WSPG calculate the XML overhead needed to wrap the actual data in the corresponding responses. To provide a simplified example, *n* parameters (of primitive types) appearing sequentially with an average name length of *l* would produce a response overhead of $n * (2 * l + 5)$ characters, according to the schema <paramName>...</paramName>. Similar but more complex functions are used for calculating the overhead for the complete response, helping to estimate s_{avg}.

PERFORMANCE EVALUATION

In the following, the caching approach enabled by the WSPG will be thoroughly evaluated regarding performance. The presented approach can be combined with other overhead reduction techniques and its usefulness is not restricted to the cases in which the devices act as SOAP clients. Actually, in the MML it is currently often combined with REST-translation plus proxy-side caching, thus eliminating the external call completely when it is safe. The extra benefits of the presented approach are notable in these combined cases, as well. However, all other techniques are out-of-scope for our evaluation, whose purpose is to measure what our solution can contribute when used *independently*. Thus, all measurements refer to a scenario where devices, MML, and external services all communicate using SOAP, without any other optimizations.

Metrics and Setup

Compared approaches: The presented approach is novel in that it achieves 100% freshness. Thus, it should be compared with the only existent approach for achieving this, namely the direct call of the external Web service without the use of any cached results. However, the performance of our solution must also be compared with that of traditional client-side caching, in order to evaluate what has to be sacrificed for achieving this absolute freshness. Thus, three approaches will be referred to, namely DCN (Direct Call, No cache), DCC (Direct Call with Caching), and PCV (Proxied Call with Validity check, which is our approach). The expectations are as follows: DCC performs best (but risks using old data), while PCV may complicate the communication compared to DCN (PCV uses extra data for the validity check and replaces one service call with two) but it reduces the wirelessly transmitted data in the case of a hit.

Dependent variables: We focus on two parameters, which –depending on the scenario– may be most important to the client: The reduction of the amount of wirelessly transmitted data or *saved bandwidth* (*sb*) and the *user-perceived latency* (*upl*).

Independent variables: The main variables that determine the performance are the *response size* (*s*) of the Web service and the *hit ratio* (*r*). For PCV, the hit ratio is the probability of the cached response being still up-to-date, while for DCC it is, equivalently, the probability of using a cached response (of course, without knowing if it is up-to-date). *s* and *r* will be varied, while the rest will be controlled.

Controlled variables: Other parameters that affect the results are the client device, the network connection, the structure of Web service responses, and the workload (service call pattern). The next subsection explains how these variables have been chosen/controlled/varied and why.

Environment and simulated aspects: *sb* is exact and environment-independent. *upl* has been measured as response time plus parsing time, because it refers to the time

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

between the user action that sends the request and the moment where the received results are ready to be presented to the user (Schreiber et al., 2010). Parsing times have been measured on a real device, while response times have been measured with a simulator in order to ease tests with different networks (GPRS, UMTS, LTE). However, these response times have been validated with sample tests from the device.

Setting the Controlled Variables

The measurements were done with an iPhone 3G (iOS 4.0.2) *device*. Less capable devices would obviously favor our approach even more. Furthermore, even if more capable devices had been used, similar conclusions could be reached by slightly increasing the examined Web service response sizes, which would be completely realistic. Thus, the mentioned device was exemplarily selected because it is popular and it obviously does not favor our approach.

With regard to the *network connection*, different results will be shown, namely for GPRS (currently most used), UMTS (successor of GPRS in 3G networks), and LTE (future technology). EDGE and HSDPA perform similarly to GPRS and UMTS, respectively. WLAN can perform similarly to LTE, while the cases where WLAN achieves a performance almost equal to wired networks are obviously out-of-scope for any adaptation technique.

The experimental *Web services* have been chosen so that their response size can be easily varied. However, other service features may affect the results, as well. Responses with different structure may need different parsing times even for the same response sizes. Thus, two different real Web services were used: YellowMap's Point-Of-Interest service (http://www.yellowmap.de/Partners/XML/PoiXmlServiceV21.asmx?wsdl) (S_1) has high parsing complexity, while a SOAP version of Apple's "iTunes new releases" service (http://ax.phobos.apple.com.edgesuite.net/WebObjects/MZStore.woa/wpa/MRSS/newreleases/rss.xml) (S_2) is easy to parse.

Figure 6 shows the parsing times measured for different response sizes and the *lines of best fit* that represent the parsing time as a function of response size *s*. The latter functions were calculated with linear regression to be approx. $8.6228 * s + 13.849$ and $4.632 * s + 22.264$ for S_1 and S_2, respectively, while other Web services would probably give a line somewhere between them. Each point depicted in Figure 6 is the average value after ten repetitions of the experiment. The deviation has been small and therefore omitted from the graph.

Different *workloads* have been tested. In the first part of the evaluation (evaluation of basic benefit), most of the results are presented for a scenario where a number of clients just send the same request twice. The number of clients is irrelevant because the saved bandwidth and the user-perceived latency reduction will be presented relatively. The existence of many clients is only assumed (and simulated) in order to assign all possible values to the cache hit ratio. More precisely, the whole response is always fetched at the first request, while the second request leads either to re-transmission of the response (cache miss) or to the transmission of our "small" 304 response. Thus, the savings cannot exceed 50%. This generic scenario was chosen because its triviality increases the comprehensibility of the results, its results consist again a worst case (minimal repetition of identical requests) for our approach, and no other scenarios are commonly accepted as realistic (Schreiber et al., 2010). This workload will be called 50-50, indicating that 50% of the calls have a chance to –but will not necessarily– be satisfied by cached results.

In the second part of the evaluation (evaluation of the tradeoffs), the workloads refer to a single client that performs the same Web service call repeatedly but with different traces each time, i.e., with different assumptions about how often the service is called, how often its responses change etc. These workloads will be justified with references to related surveys, as well as with examples of mobile applications.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 6. Examining Web services with regard to their parsing complexity

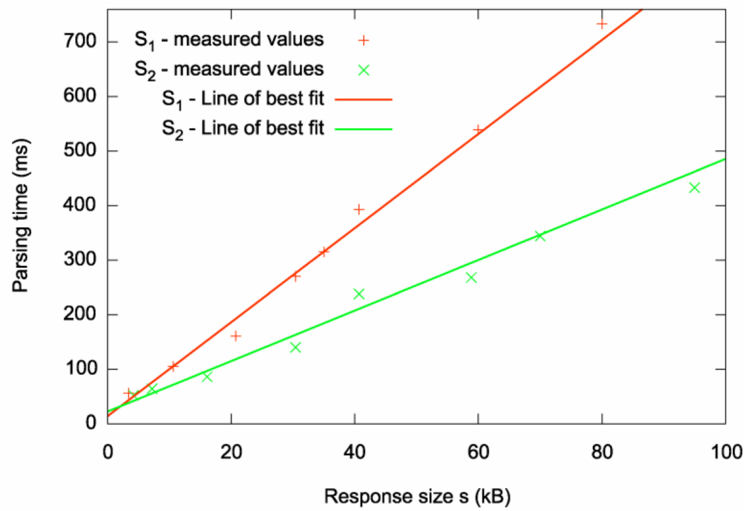
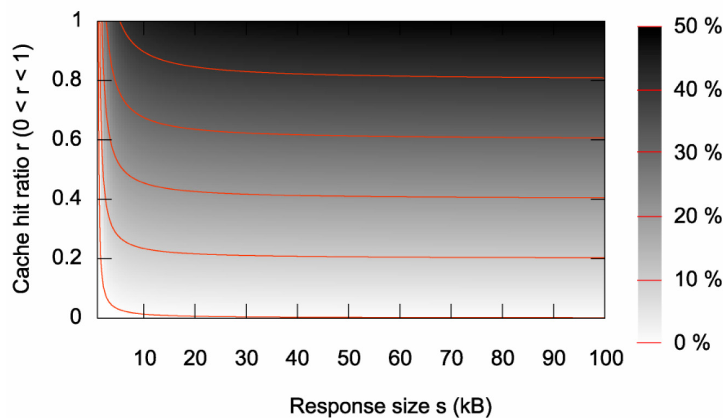


Figure 7. Saved Bandwidth (sb) as a function of response size and cache hit ratio

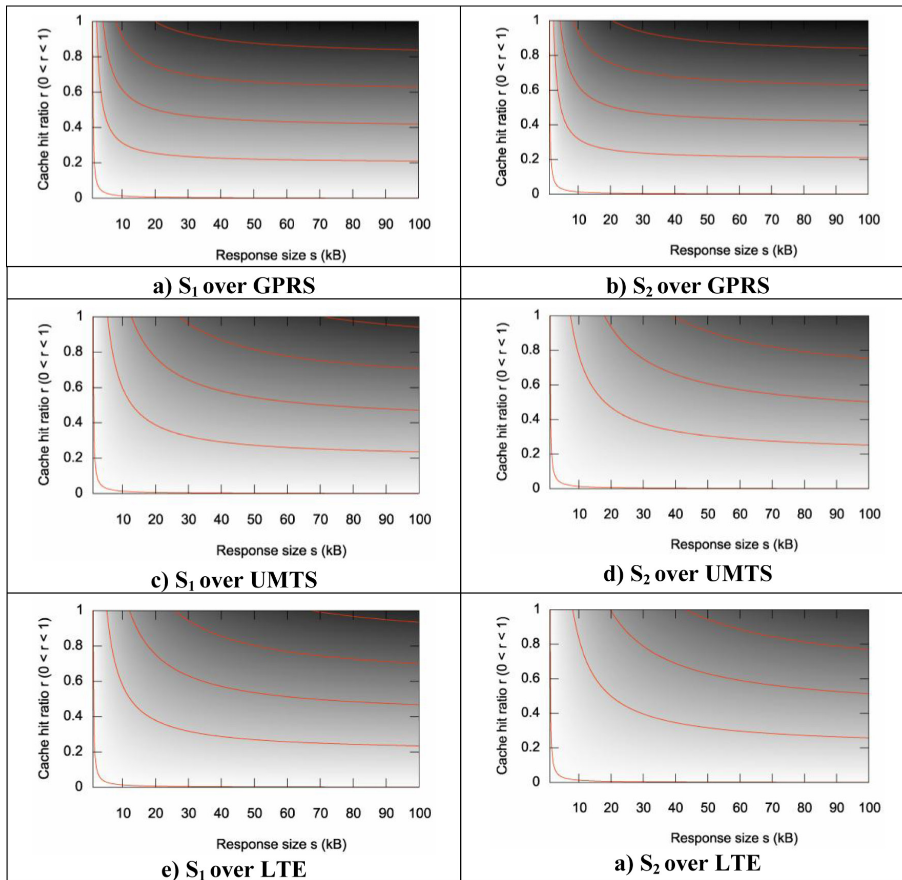


Selected Results: The Basic Benefit of the Approach

The explicit goal of PCV is to increase the saved bandwidth (*sb*). Both response time and parsing time reduction are actually caused by *sb*, so that *upl* depends up to an extent on *sb*. The question with regard to *sb* is for what values of *s* and *r* it becomes significant. Obviously, PCV cannot save much bandwidth when the exchanged messages are small and it is not worth it either when the expected cache hit ratio is low. For the 50-50 workload, Figure 7 represents *sb* in percentage of the originally used bandwidth, i.e., the bandwidth used in the case of DCN. The results prove that *sb* gets significant values early, e.g., ca. 25% of the bandwidth can be saved for mediocre values of *r* even when *s* is smaller than 10kb. Note that *sb* is service-independent. The extra data used by PCV are normally trivial and may only cause minimal performance degradation for very small values

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 8. User-perceived latency (upl) reduction with the 50-50 workload (maximum reduction < 50%) as a function of response size and cache hit ratio

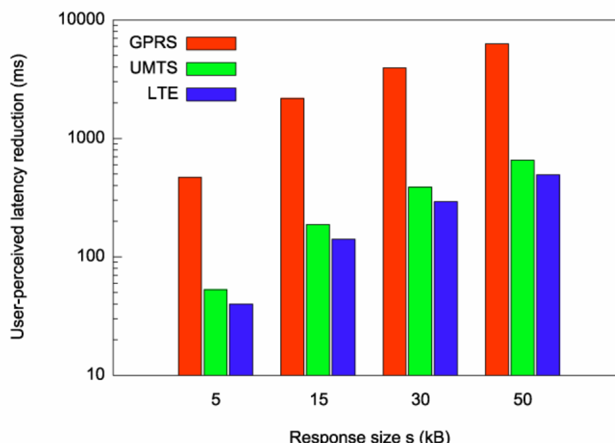


of *s* and *r* (under the lowest delimitating line in Figure 7). In all other cases, the bandwidth savings are positive.

Figure 8 shows the reduction of *upl* with a similar representation, i.e., as a percentage of the values of *upl* without caching (DCN). As expected, the reduction of *upl* is bigger for S_1. Interesting is the fact that, depending on the relationship between parsing times and response times, the percentage of *upl* reduction may be similar for LTE and UMTS (compare S_1-UMTS with S_1-LTE in Figure 8). This is due to the fact that the parsing time sometimes dominates over response time in the overall *upl*, so that the connection becomes less important. The absolute *upl* reduction is, of course, bigger for UMTS, but the same does not necessarily hold when examining the relative reduction in percentage. Most important is the fact that *upl* presents significant reductions even though the call is proxied and the backend (wired) call is, naturally, also included in the measurements. Figure 9 shows exemplarily and scenario-independently some absolute values for the *upl* reduction when S_1 is called with PCV (compared to DCN). A logarithmic scale is used because the reduction is much bigger for a GPRS network. Nevertheless, the *upl* reduction is significant for UMTS and LTE, as well, because enhancements of hundreds of milliseconds cannot be considered to be trivial. Again, each

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 9. Absolute reduction of user-perceived latency for service S_1



measurement has been repeated ten times and only average values are presented.

The response sizes used throughout the evaluation were realistic and not very big. Both test services can give response sizes > 500kB, while other Web services cause even "heavier" communication. Frequent usage of such services causes significant amounts of data to be exchanged.

Selected Results: The Tradeoffs Under Particular Realistic Call Traces

The previous results were limited to presenting the *sb*- and *upl*-reduction that PCV can achieve compared to DCN. DCC has not been involved yet because the fact that no service call actually happens at a cache hit makes it senseless to talk about *upl* and lets the reader easily assume how the results would look like. However, the following evaluation results will illustrate the spectrum between the performances of DCC and DCN where PCV is expected to lie, i.e., the tradeoff "performance vs. freshness" between the approaches. In order to demonstrate the mentioned tradeoff, an appropriate test dataset ("workload") is needed. This dataset should normally consist of different Web service call traces, i.e., logs of the monitored activity of mobile applications that perform repeated service invocations. As also explained by Schreiber et al. (2010), there are no such real public datasets because monitored activity of popular applications is usually private. Furthermore, finding a couple of such datasets would not be enough. They should also be proven to be representative. Taking this into consideration, the three approaches have been evaluated here with artificial traces, which are designed based on a use case analysis supported by related surveys. An infinite number of different traces could be designed and could appear in a real system. The following traces are single incidents that are here considered to be realistic and representative. The results must not be seen as a complete and exhaustive comparison.

The trace characteristics that are of interest –because they determine the performance of the approaches– are the *dynamicity* and the *response sizes*:

- *Dynamicity (d)* is defined here as "*the probability of each response to be identical with a previously sent response*". The dynamicity depends on two factors: The expected time intervals between subsequent service calls and the nature of the contents of the response (are they static or do they change often?). The dynamicity is measured in %.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

- *Response sizes (s)* will have a different spectrum for each trace, depending on how big the responses of the trace are expected to be. Response sizes are measured in kB.

Use Cases (UC) of mobile, Web service-based applications that may perform repeated (identical) calls will be identified and discussed with respect to their expected dynamicity and response sizes: A survey of Gartner, Inc. (Petty & Stevens, 2009) identified the following top 10 consumer mobile application categories for the near future: "*i) money transfer, ii) location-based services, iii) mobile search, iv) mobile browsing, v) mobile health monitoring, vi) mobile payment, vii) near field communication services, viii) mobile advertising, ix) mobile instant messaging, and x) mobile music*". Among them, ii, iii, v, vi, viii, and maybe x, seem to be the less sensitive and/or most promising concerning the use of cached data. Along with these results, it should be considered that the findings of two scientific surveys about public Web services (Fan & Kambhampati, 2005; Li, Liu, Zhang, Li, Xie, & Sun, 2007) can be summarized as follows: ca. 50% or more of the public Web services offer *data lookup services*, i.e., connections to –usually not very dynamic– databases, while the other important categories (10%-15%) are *sensing services* (Li et al., 2007) probably include them in data lookup services), and *data processing / conversion*. After considering the categories of the Gartner survey, the mentioned types of public Web services, and some of the "hottest" application domains, the following UCs have been identified as interesting and relevant to the purposes of a caching evaluation:

- **UC1:** Data lookup, location-based service in the domain "mobility and transport."
  - *Example*: A mobile car-sharing app developed in the context of our project Green Mobility (http://www.green-mobility-project.de) includes a Web service request for Points-of-Interest (POI). This request may be sent every time that the user wants to show possible meeting points on his map in order to choose one. Another good example from this domain would be a mobile monitoring app of a car rental company, which is informed periodically about the status (incl. location) of its cars through Web services.
  - *Characteristics*: The mentioned POI service sends responses that are usually some tens of kB but could also be much bigger. Similar could be true for a service that reports the status of cars, as location-based services in this domain may include similar format and information fields. Concerning dynamicity, responses may change because of newly added POIs. This does not happen very often, but it should be considered that identical calls do not occur extremely often but between longer time intervals, a fact that obviously increases the dynamicity. In the car rental app, the time intervals may be much smaller (maybe some seconds) but the status of a car is more likely to change often, so a given dynamicity is there present, as well. All in all, this UC is expected to have a medium dynamicity.
- **UC2:** Data lookup advertising service in the domain "live ads in communication apps."
  - *Example*: One can think of a Web service request included in a messaging app. The request would periodically look for "current ads" (push mechanisms may be used but are, especially for mobile apps, hindered by addressing problems or implementation difficulties, so that one could argue confidently that periodical pull is used very often). The concept can be understood by considering Skype, for example, where ads are continuously downloaded and shown at the bottom while the app is being used.
  - *Characteristics*: Ads may contain images, moving images, or even sound. The message sizes can vary a lot

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

here, but big-sized messages must be considered here as a possibility, especially for the future. The dynamicity is not necessarily very high, because the request may be sent often, but the "current ad" will not change every few seconds. This would be bad for the advertised party and annoying for the user.

- **UC3:** Sensing service for mobile health-monitoring in the domain "assisted living."
  - *Example*: We refer directly to the survey of Kameas and Calemis (2010), where many different example systems are mentioned. According to the survey, the most widely monitored medical variables are the electrocardiograph, the heart rate monitor, the blood pressure monitor, and the oxygen saturation monitor.
  - *Characteristics*: Although sensor data are usually compact, a look at example electrocardiograph reports reveals that a Web service response containing such a report could reach very big sizes (maybe hundreds of kB). However, if this degree of detail is needed, no chance for caching exists. Thus, systems will be considered where summarized, smaller reports are sent, which do not indicate every detail, but the status. These have smaller sizes but a very low dynamicity, because the time intervals are very small (critical apps), while the summarized status has great chances of being identical for many consequent invocations.
- **UC4:** Data processing/conversion service for mobile payment systems.
  - *Example*: A classic currency conversion service (http://www.webservicex.net/CurrencyConvertor.asmx?WSDL) can be considered here, maybe combined with the processing of some extra user data.
  - *Characteristics*: Responses of such a service may normally not exceed a maximum of 2-3 kB. The dynamicity is also pretty high, because usual apps are not expected to perform many payments on a single day, while the response of the currency converter is normally modified at latest one day later.

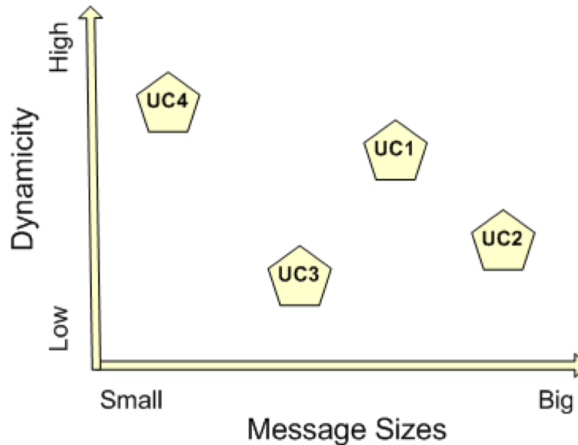
In accordance to the discussion of the UCs, Figure 10 coarsely depicts the characteristics that the Web service call traces of the UCs are expected to have. Based on these expectations and remaining close to the values used in the previous subsection, the following four traces have been designed, mirroring the discussed UCs:

- Trace 1: Test run of Web service invocations with d = 50% and s = random (20kB..80kB).
- Trace 2: Test run of Web service invocations with d = 20% and s = random(50kB..200kB).
- Trace 3: Test run of Web service invocations with d = 10% and s = random(5kB..20kB).
- Trace 4: Test run of Web service invocations with d = 70% and s = random(1kB..3kB).

Figure 11 shows for each trace the accumulated bandwidth used by DCN, DCC and PCV for 20 subsequent identical requests. DCC uses for its cached objects a Time-To-Live (*TTL*) such that the objects expire after a time that corresponds with 5 calls, so that the 6th call has to fetch a new response. TTL is normally used as part of DCC (Cao & Özsu, 2002), while the particular value has been chosen so as to be appropriate enough for the examined cases. Other values, which make DCC look less efficient, have been tested but are not included, because they do not offer many interesting new insights. Even so, it is easy to imagine how altering the TTL would alter the results. Each trace has been run many times, thus providing different instances. Averaging the results would be wrong and meaningless, as they already contain many invocations with random values. Instead, we select two typical instances of each trace and discuss their meaning.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 10. Expected dynamicity and message sizes of the use cases of interest



After a first look upon all the results, it becomes obvious that for *TTL* < ∞, PCV can even achieve better results, i.e., use less bandwidth, than DCC. However, this happens only in cases in which the responses change rarely and, in such cases, DCC would not use many outdated information. Even in the normal case, where PCV uses more bandwidth than DCC, PCV often achieves a bandwidth usage close to that of DCC. Furthermore, the crosses denote invocations for which DCC would have used outdated data. It is reminded that the only other approach that can prevent these red crosses from occurring is DCN, whose results are depicted, as well.

More concretely, Figure 11a and Figure 11b show that, for Trace 1, the bandwidth consumption of PCV remains somewhere between those of DCN and DCC. Indeed, this was true for all instances of Trace 1. However, the appearances of red crosses in the case of DCC are so often that PCV should definitely be considered, at least for critical applications.

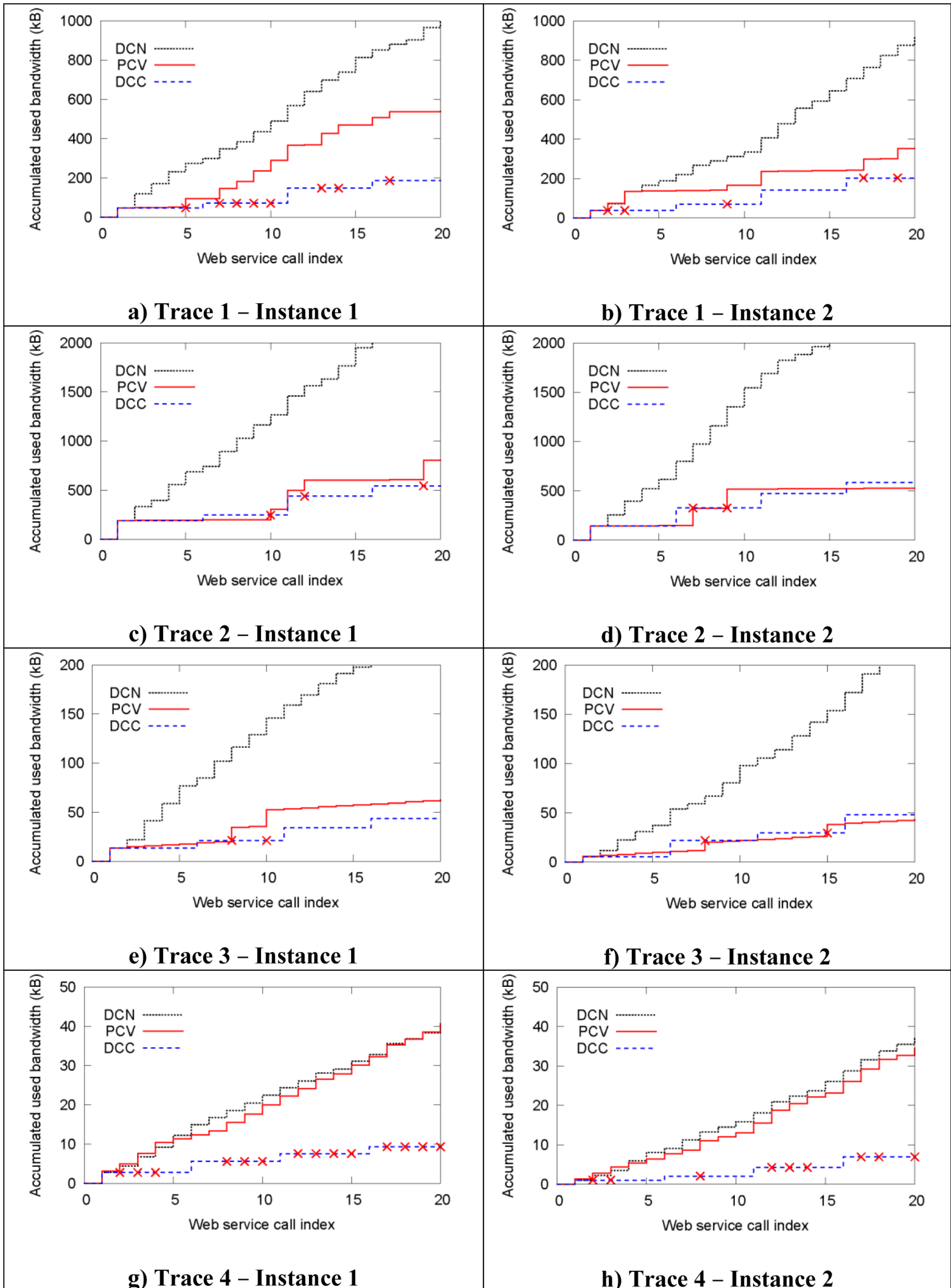
Traces 2 and 3 are cases where PCV has very high chances of being the preferred approach. Contrary to Trace 1, the difference of the used bandwidth between PCV and DCC was usually insignificant and there have even been cases where PCV consumed less bandwidth in total, although DCC has used a couple of outdated responses (cf. Figure 11d and Figure 11f). However, it must be reminded that the used bandwidth is not the only metric that may be of interest. If, for example, the number of established connections (may affect response times and/or energy consumption) is more important than the freshness of the responses, then DCC may remain preferable.

As shown by Figure 11g and Figure 11h, Trace 4 refers to cases where PCV is not efficient at all, at least in terms of bandwidth usage. In the corresponding instances, PCV has used almost as much bandwidth as DCN. Sometimes, PCV uses even more bandwidth than DCN (cf. Figure 11g), because in the case of small messages (combined with low hit ratio), the overhead of the extra fields used by the proxy is not insignificant compared to the size of the content field of the response. Analogously, the data whose re-transmission is spared by PCV, is sometimes no more than half the size of the response, which is transmitted anyway. Even if absolute freshness is demanded, PCV may still be a suboptimal solution here. DCN may be preferred because of its simplicity.

Concerning the impact that these differences of used bandwidth may have on the application in terms of user-perceived latency and costs, we refer to the first part of our evaluation.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 11. Accumulated bandwidth used by the 3 caching approaches (PCV, DCC, DCN) for the different traces



Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

SUMMARY AND OUTLOOK

Different adaptation mechanisms have been proposed in order to reduce the amount of data that is wirelessly transmitted when Web services are invoked from wireless devices. Employing caching-support as an adaptation mechanism opens new potentials, because in optimal scenarios it can spare the transmission of entire content fields, thus achieving a much bigger data reduction than any compression- or encoding-based approach. After discussing technological constraints and scientific challenges towards achieving an automatic enablement fresh and efficient response caching of third-party Web services, we presented the technical details of the first solution that can generically enhance the communication with any Web service in order to combine the use of cached responses with certainty about their freshness. The basic benefit of the approach during simple Web service invocations on wireless clients, as well as its efficiency under realistic Web service call traces of mobile applications, have been shown through an extensive evaluation.

The main focus of our future work is the investigation of the circumstances under which our solution can reduce energy consumption. As shown by Balasubramanian, Balasubramanian, and Venkataramani (2009), the reduction of the used bandwidth is not always the most important factor regarding energy consumption. Other characteristics, such as the number of connection establishments or the temporal distribution of actions, play an important role. The solution presented in this paper has been designed taking energy into consideration. However, the approach has to be extended in order to achieve the best possible results in this direction.

ACKNOWLEDGMENTS

This work was funded in part by means of the German Federal Ministry of Economy and Technology under the promotional reference 01MQ09016 (Green Mobility Project). The authors take the responsibility for the contents. Many thanks go to Philippe-Henri Marcy and Mouhannad Akhkobek for participating in the WSPG implementation, as well as to the authors of Schreiber et al. (2010) for the knowledge exchange.

REFERENCES

Aitenbichler, E., Kangasharju, J., & Mühlhäuser, M. (2007). Mundocore: A lightweight infrastructure for pervasive computing. *Pervasive and Mobile Computing*, *3*(4), 332–361. doi:10.1016/j.pmcj.2007.04.002

Balasubramanian, N., Balasubramanian, A., & Venkataramani, A. (2009). Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference* (pp. 280-293).

Canali, C., Colajanni, M., & Lancellotti, R. (2009). Performance evolution of mobile web-based services. *IEEE Internet Computing*, *13*(2), 60–68. doi:10.1109/MIC.2009.43

Cao, G. (2002). Proactive power-aware cache management for mobile computing systems. *IEEE Transactions on Computers*, *51*(6), 608–621. doi:10.1109/TC.2002.1009147

Cao, J., Zhang, Y., Cao, G., & Xie, L. (2007). Data consistency for cooperative caching in mobile environments. *IEEE Computer*, *40*(4), 60–66. doi:10.1109/MC.2007.123

Cao, L., & Öszu, M. (2002). Evaluation of strong consistency web caching techniques. *World Wide Web Journal*, *5*(2), 95–123. doi:10.1023/A:1019697023170

Cardoso, J., Barros, A., May, N., & Kylau, U. (2010, July). Towards a unified service description language for the Internet of services: Requirements and first developments. In *Proceedings of the International Conference on Services Computing* (pp. 602-609).

Christin, D., Reinhardt, A., Kanhere, S. S., & Hollick, M. (2011). A survey on privacy in mobile participatory sensing applications. *Journal of Systems and Software*, *84*(11), 1928–1946. doi:10.1016/j.jss.2011.06.073

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Davis, D., & Parashar, M. (2002). Latency performance of SOAP implementations. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid* (pp. 407-412).

Elbashir, K., & Deters, R. (2005, July). Transparent caching for Nomadic WS clients. In *Proceedings of the IEEE International Conference on Web Services* (pp. 177-184).

Fan, J., & Kambhampati, S. (2005). A snapshot of public Web services. *SIGMOD Record*, *34*(1), 24–32. doi:10.1145/1058150.1058156

Frye, C. (2009). *SOA growth and change*. Retrieved August 23, 2011, from http://searchsoa.techtarget.com/news/1351532/SOA-growth-and-change-TechTarget-survey-shows-SaaS-BPM-emerging

Jelencovic, P., & Radovanovic, A. (2008). The persistent-access-caching algorithm. *Random Structures and Algorithms*, *33*(2), 219–251.

Kameas, A., & Calemis, I. (2010). Pervasive systems in health care. In Nakashima, H., Aghajan, H., & Augustus, J. C. (Eds.), *Handbook of ambient intelligence and smart environments* (pp. 315–346). New York, NY: Springer. doi:10.1007/978-0-387-93808-0_12

Karedla, R., Love, S., & Wherry, B. (1994). Caching strategies to improve disk system performance. *IEEE Computer*, *27*(3), 38–46. doi:10.1109/2.268884

Li, W., Zhao, Z., Qi, K., Fang, J., & Ding, W. (2008, July). A consistency-preserving mechanism for web services response caching. In *Proceedings of the IEEE International Conference on Web Services* (pp. 683-690).

Li, Y., Liu, Y., Zhang, L., Li, G., Xie, B., & Sun, J. (2007, July). An explanatory study of Web services on the Internet. In *Proceedings of the IEEE International Conference on Web Services* (pp. 380-387)

Liu, X., & Deters, R. (2007). An efficient dual caching strategy for Web service-enabled PDAs. In *Proceedings of the ACM Symposium on Applied Computing* (pp. 788-794).

Oberle, D., Bhatti, N., Brockmans, S., Niemann, M., & Janiesch, C. (2009). Countering service information challenges in the Internet of services. *Business & Information Systems Engineering*, *1*(5), 370–390. doi:10.1007/s12599-009-0069-9

Oh, S., & Fox, G. C. (2006). Optimizing Web service messaging performance in mobile computing. *Future Generation Computer Systems*, *23*(4), 623–632. doi:10.1016/j.future.2006.10.004

Papageorgiou, A., Blendin, J., Miede, A., Eckert, J., & Steinmetz, R. (2010, July). Study and comparison of adaptation mechanisms for performance enhancements of mobile web service consumption. In *Proceedings of the IEEE World Congress on Services* (pp. 667-670).

Papageorgiou, A., Leferink, B., Eckert, J., Repp, N., & Steinmetz, R. (2009, December). Bridging the gaps towards structured mobile SOA. In *Proceedings of the International Conference on Advances in Mobile Computing and Multimedia* (pp. 288-294).

Papageorgiou, A., Schatke, M., Schulte, S., & Steinmetz, R. (2011, July). Enhancing the caching of Web service responses on wireless clients. In *Proceedings of the IEEE International Conference on Web Services* (pp. 9-16).

Papazoglou, M. P., & Heuvel, W. J. (2007). Service oriented architectures: Approaches, technologies and research issues. *The Very Large Data Base Journal*, *16*(3), 389–415. doi:10.1007/s00778-007-0044-3

Patty, C., & Stevens, H. (2009). *Gartner identifies the top 10 consumer mobile applications for 2012*. Retrieved September 14, 2011, from http://www.gartner.com/it/page.jsp?id=1230413

Podlipnig, S., & Böszörmenyi, L. (2003). A survey of Web cache replacement strategies. *ACM Computing Surveys*, *35*(4), 331–373. doi:10.1145/954339.954341

Press, L. (1999). The post-PC era. *Communications of the ACM*, *42*(10), 21–24. doi:10.1145/317665.317670

Schreiber, D., Aitenbichler, E., Göb, A., & Mühlhäuser, M. (2010, July). Reducing user perceived latency in mobile processes. In *Proceedings of the IEEE International Conference on Web Services* (pp. 235-242).

Sesia, S., Toufik, I., & Baker, M. (2009). *LTE: The UMTS long term evolution: From theory to practice*. Chichester, UK: John Wiley & Sons. doi:10.1002/9780470742891

Takase, T., & Tatsubori, M. (2004). Efficient web services response caching by selecting optimal data representation. In *Proceedings of the International Conference on Distributed Systems Computing* (pp. 188-197).

Tatemura, J., Po, O., Sawires, A., Agrawal, D., & Candan, S. (2005). WreX: A scalable middleware architecture to enable XML caching for Web services. In *Proceedings of the ACM/IFIP/USENIX International Conference on Middleware* (pp. 124-143).

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Tekli, J., Damiani, E., Chbeir, R., & Gianini, G. (2011). SOAP processing performance and enhancement. *IEEE Transactions on Services Computing*, *99*, 1–18.

Terry, D., & Ramasubramanian, V. (2003). Caching XML Web services for mobility. *ACM Queue; Tomorrow's Computing Today*, *1*(3), 70–78. doi:10.1145/846057.864024

Tian, M., Voigt, T., Naumowicz, T., Ritter, H., & Schiller, J. (2004). Performance considerations for mobile Web services. *Computer Communications*, *27*(11), 1097–1105. doi:10.1016/j.comcom.2004.01.015

Apostolos Papageorgiou is currently working for NEC Europe Laboratories in Heidelberg Germany) and holds a PhD degree from the Multimedia Communications lab at the Technische Universität Darmstadt (Germany). He received his diploma from the Computer Engineering and Informatics Department, University of Patras (Greece) in 2007, while he has also stayed in 2005 as a visiting student at the University of Valladolid (Spain) and in 2011 as a visiting researcher at the Yale University (USA). While involved in lectures about net-centric systems, his research and his publications have focused on service-oriented computing, especially mobile Web service adaptation and optimization. From 2009 to 2011 has also worked for the Hessian Telemedia Technology Competence Center (httc e.V.) as a leader of the research project Green Mobility.

Marius Schatke studied at the Technische Universität Darmstadt, where he finished his BSc degree in electrical engineering in 2011. During his studies, he specialized in distributed data processing and gathered software engineering experience in smartphone development. His Bachelor thesis was about caching mechanisms for Web service-based mobile applications. Marius continued to work in this technology area, being co-founder and technical leader of a start-up company in Berlin (Germany), which develops smartphone-related services and achieved to win a founder's grant from EXIST, a business start-ups' support program of the German Federal Ministry of Economics and Technology (BMWi).

Stefan Schulte is a Project Assistant and Postdoctoral Researcher at the Distributed Systems Group of the Vienna University of Technology. Stefan holds a PhD degree from the Technische Universität Darmstadt (Germany), where he headed the research group "Service-oriented Computing" of the Multimedia Communications Lab until July 2011. He received a diploma degree in economics and a Bachelor in computer science from the University of Oldenburg, Germany, and a Master of Information Technology (with Merit) from the University of Newcastle, New South Wales, in 2005 and 2006, respectively. His current research focus is on Quality of Service (QoS) and Quality of Experience (QoE) aspects of Service-oriented Computing, SOA Benchmarking, Semantic Web Services.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Ralf Steinmetz worked for over nine years in industrial research and development of distributed multimedia systems and applications. Since 1996 he is head of the Multimedia Communications lab at Technische Universität Darmstadt, Germany. From 1997 to 2001 he directed the Fraunhofer (former GMD) Integrated Publishing Systems Institute IPSI in Darmstadt. In 1999 he founded the Hessian Telemedia Technology Competence Center (httc e.V.). His thematic focus in research and teaching is on multimedia communications with his vision of real "seamless multimedia communications." With over 200 refereed publications he has become ICCC Governor. He was awarded as Fellow of the IEEE and in 2002 as Fellow of the ACM, becoming the first German researcher awarded with this title from both IEEE and ACM. In 2008, he was awarded the first ACM SIGMM "Award for Outstanding Technical Contributions to Multimedia Computing, Communications and Applications."

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.