

Application Partitioning for Enhancing System Performance for Services Hosted on Wireless Devices

Muhammad Asif¹, Shikharesh Majumdar¹, and Raluca Dragnea²

¹ Department of Systems and Computer Engineering, Carleton University,
Ottawa, Ontario, K1S 5B6, Canada
{masif, majumdar}@sce.carleton.ca

² Alcatel-Lucent, 600 March Road, Kanata, Ontario, K2K 2E6, Canada
raluca.dragnea@alcatel-lucent.com

Abstract. Applications based on a service-oriented architecture are getting popular in the domain of business to customer electronic commerce and in automating information exchange between business processes. In this paper, we are investigating a service-oriented architecture for hosting of the services in wireless environments. The proposed architecture uses application partitioning techniques to offload the execution of service application to a backend node. Hosting of services on wireless devices is challenging, but we have shown that by using the application partitioning techniques proposed in this paper, it is possible to deliver services from such devices that are often characterized by very limited resources and attain good system performance. The paper also discusses a set of design guidelines for partitioning of web service applications to be deployed in wireless environment.

Keywords: application partitioning, mobile web services, service oriented architecture, coarse grain partitioning, fine grain partitioning.

1. Introduction

A service-oriented architecture (SOA) is a progression of distributed computing and modular programming that provides blueprints for design, development, deployment and management of a loosely coupled business application infrastructure [14]. SOA is based on a collection of services that communicate with each other by passing data from one service to another, or by coordinating an activity between one or more services. Web services can be used to implement a service-oriented architecture. The major advantages of implementing an SOA using web services are that web services are pervasive, simple, and inter-operable. Web services are software components that can be accessed over the internet using standard protocols and well defined interfaces such as Simple Object Access Protocol (SOAP). The use of SOAP as messaging framework is one design strategy for web services based systems. The other popular design strategy is Representational State Transfer (ReST), which is a resource oriented approach [6]. SOAP based web services are used in the experiments conducted on an SOA based system in this paper.

Provisioning of services from a mobile device is a challenging research area because of the limited resources of the device and the limited wireless bandwidth. In comparison to desktop nodes, mobile devices usually come with limited memory capacity, slow processor speeds and lower battery power as well as slower and less reliable communication links. In the context of this paper, such a mobile device can be a smart phone, a personal digital assistant (PDA) or a specialized device with limited resources. Writing optimized programs in low level native codes for a large variety of such devices is possible but is an expensive solution because of the diversity of hardware architectures and operating systems. Such solutions are often not suitable for exchange of data between the business processes because of lack of inter-operability.

Recent efforts investigating the feasibility of hosting web services on mobile devices are described in [1,12,18]. An inter-operable service at least requires a SOAP/XML processor for accessing a service. SOAP/XML processing uses a considerable amount of memory and processing power. The support for extended web service specifications for security, reliability and transactions can add more overhead. In such situations, providing a service involving a complex business process is not feasible for mobile devices.

There are a number of applications where hosting web services on mobile devices is important and can improve a business process. For example, a mobile device installed on a vehicle or carried by a skilled person can expose a service to track its exact location at any time. This type of service can be used to track a skilled person such as a doctor or to track the delivery of items on a delivery truck. Mobile web services can also find their application in supply chain management systems. A person running a small business and using a small laptop or a powerful handheld device in the field can be a part of a supply chain system used by an enterprise. The services offered by such a person in the field can be available through web services hosted on his/her mobile device. For example, a technician in the field can expose his/her progress of work through a web service to a manager in a company if s/he is using a device to log the events and milestones while completing a job or a list of jobs. The reason for hosting web services on his/her mobile device is that the data to be used in a web service needs to be the most recent. Since s/he is always updating the data while working in the field, it makes sense to host the web service on his/her mobile device. A number of other applications such as a wallet service, a parcel tracking system, sharing the time or the dictionary are discussed in more detail in [12,18].

In this paper we have described some of the results of a collaboration research project on web services based Quality of Service (QoS) aware large scale applications between Carleton University and Alctael-Lucent, Ottawa, Canada. The paper demonstrates that WS partitioning is essential for getting a reasonable performance from the prototype system and application investigated in this research. The main contributions of this paper are summarized.

- A service oriented architecture that uses application partitioning techniques to facilitate hosting of services in a wireless environment is proposed.

- Two application partitioning techniques are introduced to offload a part of mobile web services to a surrogate or a backend node. The proposed techniques are coarse grain partitioning and fine grain partitioning.
- A set of design strategies to partition web services for enhancing system performance is discussed.
- The performance of a prototype system built for the investigation of the proposed techniques is analyzed by using sample web services hosted in wireless environment.

The rest of the paper is organized as follows. Section 2 summarizes the related work for hosting of web services on mobile devices and different types of application partitioning techniques discussed in the literature. Section 3 discusses two types of web service application partitioning techniques and the guidelines for partitioning of web services. In Section 4, a service oriented architecture that uses the application partitioning approach for hosting of services is discussed. Section 5 discusses the experimental setup and the performance results. Conclusions and future research areas are presented in Section 6.

2. Related Work

There is a large body of knowledge available for accessing services from a mobile device. But a very little work has been done so far in the area of hosting services on mobile devices. In our previous work [1], we proposed a light weight WS toolkit to host web services on resource constrained devices. The proposed architecture of the toolkit is implemented using Java ME and is suitable to handle approximately a dozen of concurrent clients. We have also demonstrated that by partitioning the web service toolkit and running a part of it on a fixed intermediate node, the performance of the system can be significantly improved. The current work uses the same WS toolkit proposed in [1] to host web services. The other efforts in the area of providing services from a mobile device are discussed in [12,18]. In [12], IBM researchers developed a prototype for a shopper-kiosk application running on a PDA where a shopper can use his/her wallet service to pay bills. Use of Bluetooth, however makes its application very limited to a small area. A prototype for hosting web services on a Sony Ericson smart phone is described in [18]. A tracking web service is used to test the prototype. A web service handler is developed on top of a web server to handle web service requests. The authors used PersonalJava [14] instead of J2ME in system implementation because it is the only platform available for the Sony Ericsson phone.

Little work exists in the area of hosting WS on mobile devices. The existing work in the area is good for simple services that demand very less resources. These efforts showed a reasonable performance for high end PDAs and smart phones. The devices with average resource capabilities are not expected to exhibit similar performance for hosting of light weight services. Thus the focus of this paper is on how to use application partitioning to offload a resource constrained wireless device and run a part of the application on another surrogate system. Application partitioning has been successfully used for partitioning of desktop and distributed applications. To the best of our knowledge, application partitioning has not been investigated for hosting of web services on mobile devices. A very high level overview of previous work in the area of partitioning of general applications is presented next.

Application partitioning can be achieved in different way: static and dynamic. Static application partitioning provides a separation of application components at design time. Dynamic application partitioning is usually done either at compile time or at run time. In design time partitioning of applications, which part of the application will be executed on which node is decided during application design. Typical examples of this type of partitioning are client server applications (see [16, 20]). Intelligent partitioning of applications can also reduce network traffic. Watson [22] proposed to partition the data and functionality of an application to reduce network traffic. For compile time partitioning, applications are usually partitioned into multiple parts based on an input configuration provided by the designer. Different approaches have used different configuration parameters such as the number of partitions, partitioning criteria and the association between different components of the application. A substantial work has been done on compile time partitioning and tools have been developed to partition existing applications. These include J-Orchestra [21], Addistant [19], Pangaea [17], Coign [8] and Protium [24]. In addition to these tools, there are a few approaches that suggest partitioning at the program level. For example, the idea of breakable objects (BoBs) for compile time application partitioning in Java is introduced in [9]. BoBs are the entities in a program that can be easily split. In run time application partitioning, programs are divided into partitions at execution time. This type of partitioning is more challenging than the previous two types. Research performed in this area can be found in [3, 13]. This paper is focused on partitioning of web services at design time.

3. Web Service Partitioning

In this paper, we propose two design time application partitioning techniques for web services: *coarse grain partitioning* and *fine grain partitioning*. The first suggests partitioning of a web service application at the package or the class level and wrap these partitions into multiple child web services while the second suggests partitioning of a web service at the method level. The objective of partitioning the web service application is to offload the complex components of a web service from a mobile node with limited resources and execute them on a fixed backend node. The partition of the web service that requires to be executed on a mobile device can be put in a separate web service or a remotely accessible application (depending on which technique is used) and deployed on the device. The rest of the components can execute on a fixed backend node that does not suffer from the resource limitations of a mobile device. The details of these web service partitioning techniques are discussed next.

3.1 Coarse Grain Web Service Partitioning

Coarse grain partitioning is primarily for web services that are implemented using packages and classes. With coarse grain partitioning, a web service is decomposed into child partitions such that each partition has one or more classes in it and can be executed on a separate node. Although the child partitions can be deployed as independent distributed applications, but to provide inter-operability across different

partitions we propose to wrap these child partitions into web services. Wrapping child partitions to web services is good for heterogeneous web service execution environments and platforms since the child WS can be deployed on any node irrespective of the platform it is using. But the processing of multiple (child) web services instead of a single original web services can add an extra overhead due to additional SOAP/XML messaging. For complex systems, the response of multiple child web services can be collected and coordinated by using a work flow engine. The work flow engine is responsible for the arrangement and management of different web services to achieve the desired goals. Work flow engines can be built using work flow languages that define flows of the execution of different web services or business processes. The most popular work flow languages are the Web Service Flow Language (WSFL) [11] and the Business Process Execution Language (BPEL) [2].

3.2 Fine Grain Web Service Partitioning

Fine grain partitioning is for decomposition of a WS application at a finer level: a method or procedure level for example. This partitioning technique is useful for services that are implemented using native code of devices or procedural languages such as C. The complexity of performing the partitioning is increased as the granularity level becomes finer and finer. In fine grain web service partitioning, it may be possible to keep the partitions as remotely accessible applications or wrap them into child web services. Since this scheme of partitioning is mainly for implementations that are either written in the device native code or in a language for which a WS execution environment is not available, we deploy the different partitions as remotely accessible applications communicating with each other by using distributed technologies such as Remote Method Invocation (RMI) [20], Distributed Component Object Model (DCOM) [5] and Common Object Request Broker Architecture (CORBA) [4].

3.3 Design Guidelines for Web Service Partitioning

Before discussing the design guidelines for partitioning web services, it is important to highlight the criteria for web service partitioning. The criteria for web service partitioning includes a number of factors: the number of partitions, local resource requirement, frequency of communication between different components of a web service and the resource limitations of the mobile device. Thus in the context of mobile web services, knowledge of the type of the node (an embedded device or a handheld device or a smart phone) may also be important at the time of partitioning. Depending on the complexity of a web service, it can be partitioned into more than two parts to improve the performance. In this paper, we present our experience with the partitioning of a web service into two partitions: one to be deployed on a mobile node and the other on a powerful surrogate node. Using more than two partitions forms an important direction for future research. Based on these factors, we are proposing three guidelines for partitioning web services at the design time. The first two guidelines are devised from well known guidelines in distributed computing.

1. Services hosted on mobile devices are expected to use local resources and there will be a set of local systems calls in the WS application. This guideline suggests quarantining part of the application that makes local system calls in a separate partition. The part of the application that does not need to use local resources can be put in the partition that is supposed to be deployed on a backend node.
2. For improving system performance, the communication between different modules of a WS should be minimized. The different partitions of a web service must be engineered to reduce the inter-module communication. This guideline suggests moving a part of a web service to the partition that is to be deployed on the backend node only if this part of the web service is not using local resources of the mobile device and moving it to a backend node does not demand a large amount of data transfer between the backend node and the mobile node.
3. This design guideline concerns the use of design patterns for performance optimization and focuses on devising a more optimized system by using the façade design pattern for multiple web services available on the same node. The façade design pattern [7] is a structural pattern that provides simple interface for complex subsystems. One of the most important benefits of the façade pattern is that it reduces network round trips between remote systems. It is especially important for mobile web services, since the wireless network is very slow. In the context of mobile WS applications, we are suggesting to use the façade design pattern to integrate the response of multiple web services hosted on the same node (mobile device). To explain this guideline, we use an example in which a node is offering two web services A and B. A work flow engine may require these two web services A and B to satisfy a service request. The façade pattern suggests that it would be economical to combine the two web services A and B into one service AB so that the work flow engine makes a one call to get the response of both.

4. SOA with Application Partitioning Approach

A typical architecture for invocation of a web service is based on three components as shown in Fig.1-a. A service provider first publishes the service it provides to a database usually called the service registry. A service client searches the service registry to find an appropriate service. On finding a service, the client invokes the service by interacting with a service provider. The architecture is simple and is widely used to implement a large variety of services in a fixed infrastructure.

The existing literature on hosting services on a wireless device (as discussed in Section 2) uses the architecture shown in Fig. 1-a. As already mentioned, the existing work in the field available from the literature is good for simple services using the basic requirements specified in the web service standards. To provide services with more resource requirements and to support extended web services specifications for security, reliability and transactions, this paper proposes to extend the architecture of Fig1-a. In the proposed architecture, the WS client and the registry components are the same. The service provider component is redesigned to either act as a proxy for a service hosted on a mobile device (see Fig. 1-b) or act as a coordinator to aggregate responses of multiple partitions of a web service (see Fig. 1-c) that may be hosted on multiple nodes. The multiple partitions can be achieved by using either a coarse grain or a fine grain partitioning technique. The dual role for the service provider is

required because sometimes it is not possible to partition a web service because of its light weight functionality or its design or a complete dependency on local resources. In such cases, the service provider will perform as a proxy for the actual service. In case of WS partitions, the role of the service provider is to act as a WS partitions coordinator. In both approaches, the service provider uses two main components to accomplish the role of a proxy: the service proxy and the service manager (see Fig.1-b and Fig. 1-c).

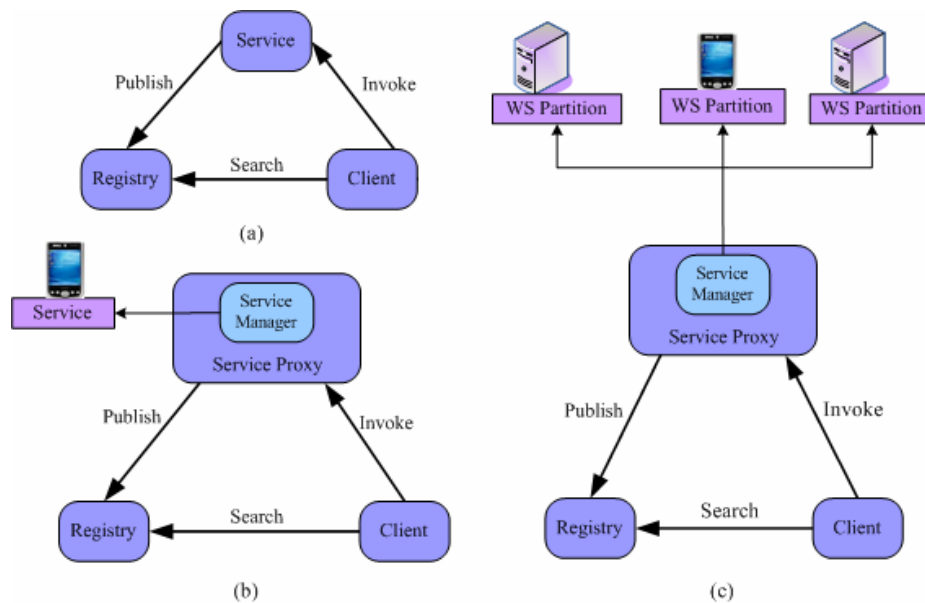


Fig. 1. a) A classic service oriented architecture b) A SOA that uses the WS provider as a proxy c) A SOA that uses the WS provider as a WS partitions coordinator

4.1 Service Provider as a Proxy

As already mentioned, it is not always possible to partition a web service for avoiding the cost associated with the redesign of an existing WS or because the service uses local resources heavily. In such a situation, this paper proposes to use a proxy service provider in the fixed infrastructure. There are a number of advantages of using this approach. The service provider proxy can facilitate the access of the existing application as a service. Use of a proxy service provider allows the service to be implemented using the device's native code while providing a WS interface to the service consumer. For example, this approach may be useful to collect data from sensor devices and provide the sensor functionality as a web service by using an appropriate wrapper without deploying any WS execution environment on the resource constrained sensor device. Even if the service deployed on a mobile device is available as a standard web service, the proxy service provider can be useful for session management and supporting other extended web service specifications for security and transactions. The service proxy is an interface of the published service to

WS clients. The service manager is responsible for delegating service requests to or collecting results from the actual service hosted on a mobile device.

4.2 Service Provider as WS Partitions Coordinator

The use of the service provider as a WS partitions coordinator is useful if the service can be partitioned using either a coarse grain partitioning or a fine grain partitioning technique. As discussed in Section 3, WS partitioning can be beneficial to augment the resource capabilities of mobile devices. The responses of different WS partitions (implemented as child services or distributed objects) however need to be aggregated by a coordinator. The main steps to invoke multiple service partitions using the service manager and the service proxy used in the prototype described in this paper are presented next.

A web service is assumed to be partitioned into multiple parts using the design guidelines for partitioning discussed in Section 3.3. In the second step, the WS partitions are deployed on different nodes as child web services or remotely accessible distributed objects. In the next step, the service manager coordinates the child partitions to achieve the overall goal of the WS. For more complex systems, the service manager can use a work flow or state diagram to provide the coordination among the different partitions. The aggregated response of all the partitions is handed over the service proxy that sends it back to the actual client as a final step.

There are a number of advantages of the proposed service-oriented architecture for hosting of services in a wireless environment. The use of the service proxy and the service manager makes the client and the service registry hide the deployment information of the service. Moreover, an optimized and wireless friendly transport mechanism can be used between the service proxy node and the mobile device while a client can use the standard transport mechanism (such as HTTP over TCP) for the exchange of SOAP messages with the service proxy. As a service proxy (as discussed in Section 4.1), the mobile application available on a device (and accessible on the network) can be published as a service without any changes to the existing code of the application. The service proxy can wrap the legacy code as a web service and expose it to the outside world.

5. Experimental Analysis

In this section, the prototype implementation of the proposed architecture and the impact of WS partitioning techniques on system performance are discussed.

5.1 Sample Services

For an experimental investigation of the techniques described earlier, we have developed two sample web services that are deployed on a personal digital assistant. These are discussed next.

Tracking Service: The goal of this service is to provide the exact location of the device on which the service is deployed. The service provides the address of the

location, its elevation, time zone and the population of the area. As a first step, the web service fetches the actual Global Positioning System (GPS) coordinates from a GPS receiver (built-in or attached). For this paper, we emulate the step of getting GPS coordinates by fetching a random set of coordinates from a local file containing more than a thousand locations. In the next step, the WS queries a database of locations to find the details of a location that is closest to the GPS coordinates. The location database is downloaded from a well known geographical organization (GeoNames). In the last step, the location information is serialized as a response message and sent back to the service requester.

Schedule Service: This is a simple service that can be used to fetch the schedule detail of a person from his/her device. It is assumed that the person is using a scheduling application for managing his/her appointments and he/she is exposing the schedule as a web service. The data of private meetings and appointments are assumed to be private and not available to the web service.

5.2 Setup

In the experiments, we used a Dell Axim x51v PDA that has an Intel processor of 624 MHz and a memory of 64 MB and is running Windows Mobile 5.0 as an operating system. Services are deployed on the device using a J2ME (J9) runtime environment [10]. The service proxy and the service manager are deployed on a desktop node that is equipped with a 3.0 GHz Intel single core processor and a memory of 1GB. The WS clients are running on a Dell Latitude D610 laptop having a memory of 1.5 GB and an Intel single core Centrino processor of 1.86 GHz speed.

The intercommunication between the service provider (the service proxy and the service manager) node and the PDA is based on a wireless local area network. The service provider node is equipped with a wireless adaptor compatible with the IEEE 802.11g standard. The PDA is also equipped with a wireless adaptor using the IEEE 802.11b standard.

The performance of the system is analyzed by measuring the end to end response time. *Response Time* is the time taken to invoke a web service. It is measured by taking the difference of the time when a web service response is received by the client and the time when the WS request is sent to the server.

5.3 Analysis of WS Partitioning Techniques

The performance of the proposed SOA with application partitioning techniques is analyzed using the Tracking service. Tracking service involves system calls to local resources and has a component for querying a large database that requires a great deal of processing and physical space for storage. For simplicity, the Tracking service was partitioned into two child partitions. The two child partitions of the sample Tracking service are deployed both as child web services (coarse grain partitioning) and as remote distributed objects (fine grain partitioning). In the first set of experiments, we used only the first two partitioning guidelines discussed in Section 3.3. The third partitioning guideline is applicable only if multiple services are available on a device.

The different partitions of the Tracking service achieved as a result of applying two partitioning techniques are shown in Fig. 2.

One of the issues that is important in the context of the two partitioning techniques is the overhead associated with communication between the partitions. Although in general, different components may be included in a partition for coarse grain and fine grain partitioning, in the prototype system described in this paper, GPS related functionalities and the database related functionalities are inserted in separate partitions for both coarse grain and fine grain partitioning. However, SOAP is used for intercommunication in the first whereas an RMI is used in the second. Thus, the comparison of the two techniques captures the impact of the different communication mechanisms used on performance.

Fig. 2-a shows the Tracking service as a whole (un-partitioned) that is deployed on the Dell Axim PDA in one of the experiment. The Tracking service is partitioned into two child services: GetGPSCoordinates (deployed on the PDA) and QueryDatabase (deployed on a desktop node). GetGPSCoordinates involves the use of local resources so it is kept in one partition and installed on the mobile device. QueryDatabase involves a lot of processing and it also requires access to a database with millions of records containing locations. Deploying this database on the device takes a lot of storage space as well. Moreover, QueryDatabase does not use any local system resources, so we moved the QueryDatabase partition with the database to the backend node in the fixed infrastructure. Fig. 2-b shows the result of a coarse grain partitioning of the Tracking service in which the two partitions are deployed as two child web services. Fig. 2-c shows the result of a fine grain partitioning of the Tracking service in which the two partitions are deployed as two remotely accessible distributed objects. The advantage of using distributed objects is that it can use an optimized transport mechanism for exchange of data. In the experiments described in this paper, we used Remote Method Invocation (RMI) for communication among the remote objects.

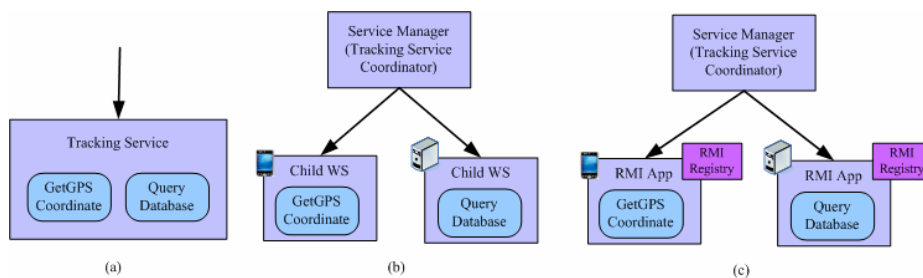


Fig. 2. a) Un-partitioned Tracking service b) Tracking service partitioned into child web services c) Tracking service partitioned into remotely accessible distributed objects

The performance of the system using these partitioning techniques on the Tracking service is observed by sending one thousand requests (one after the other) from multiple concurrent clients. The system is tested using one, five and ten concurrent clients. The results are summarized in Fig. 3. The results show that the Tracking service partitioned into two partitions performs much better in comparison to the un-partitioned Tracking service which is deployed as one entity on the mobile device.

The response time achieved with the un-partitioned Tracking service for 5 and 10 clients is inordinately large and is clearly unacceptable. WS partitioning is crucial for enhancing system performance. Tracking service deployed as two child web services (coarse grain partitioning) performs better than the service that is deployed as two distributed objects (fine grain partitioning). This result is somewhat unexpected and surprising. We were expecting the system using distributed objects to perform better than the one that uses child web services because the system with distributed objects does not give rise to additional overhead of SOAP/XML processing.

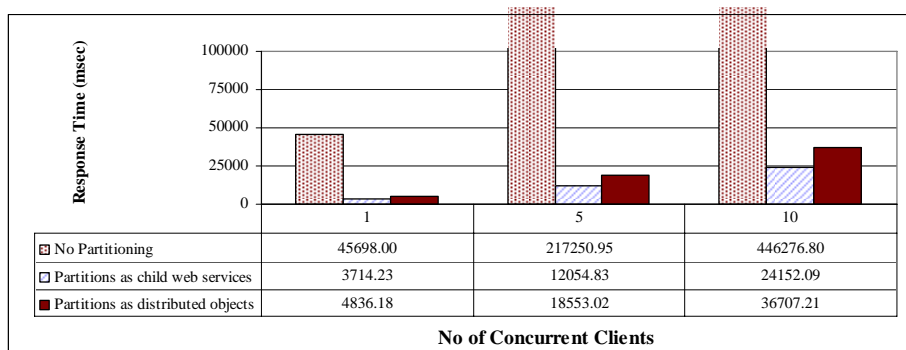


Fig. 3. Performance comparison for hosting the Tracking service with and without using partitioning techniques (The sequence of bars follows the same order as the sequence of legends)

An analysis shows that this is due to the RMI framework and its implementation available for J2ME. Accessing a service from RMI based remote object is a two step process. First, the RMI client uses a registry to get a reference of the remote object and then it uses the reference to invoke a particular method on the remote object. So the partitioning of a web service into multiple RMI based remote objects results in more overhead because of the increased number of messages between the service manager node and the nodes hosting remote objects. Another reason of the inferior performance of RMI based partitioning is because of its thread creating policy. A new thread is created for each new request. The RMI runtime keeps the newly created thread for a certain amount of time n to serve another request. If no new request arrives during n time units, the thread is destroyed. The time period duration n depends on an implementation and is not alterable by programmers. This thread creating strategy also adds overheads to service delivery. Caching the remote object reference by the WS coordinator and reuse it in subsequent calls can reduce the overhead. So applications that invoke a remote object multiple times can amortize the reference acquiring overhead over multiple object invocations and improve performance. We are conducting experiments to investigate the impact of the reference caching and the use of other communication mechanisms such a light weight RPC on system performance when fine grain partitioning is used. When partitions are deployed as child web services (Fig. 2-b), we used a light weight WS execution environment [1] as discussed in Section 2. The same WS execution environment is used for hosting the un-partitioned Tracking service (Fig. 2-a).

We have constructed specific partitions for the Tracking web service considered in this paper and demonstrated that a partitioned system is superior in performance to an un-partitioned system. For large arbitrary applications that involve sessions with multiple transactions and perform integrity and confidentiality related activities for example, a partitioning tool may be useful in determining the partitions to be deployed on the mobile device and on the backend node(s). We are currently working on a graph theoretic approach in which the different components of the application are represented by nodes and the edges between nodes model the intercommunication between components. A processing cost is associated with each node whereas a communication cost is associated with an edge. The goal of the partitioning algorithm is to group the application components into partitions in such a way that the resource constraints of the wireless device are satisfied and the overall cost is minimized. We plan to disseminate the results of this research in a subsequent paper.

5.4 The Utility of Using the Façade Design Pattern

In another experiment, we investigated the significance of the façade design pattern for aggregating light weight services if they are deployed on the same node and are required by a WS coordinator for WS Composition. In this experiment, we assume that Tracking service and the Schedule service are deployed on the same node. A WS Composition entity requires these two services to provide the location of a person and his/her current schedule. Steps for the design of interactions of different services and application of a partitioning technique are captured in Fig. 4.

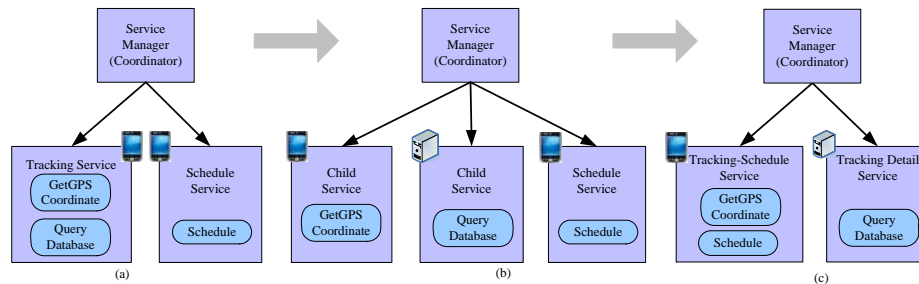


Fig. 4. Partitioning approach with aggregating two services using the façade design pattern a) a system with two web services without using any partitioning technique b) a system with the Tracking service partitioned into two parts c) a system that uses façade to combine results of two partitions deployed on the mobile node

Fig. 4-a shows that the service manager is invoking the Tracking service and the Schedule service independently to accomplish a WS request for the location of a person and his/her current schedule. Fig. 4-b shows that the Tracking service is partitioned using either the coarse grain or the fine grain partitioning and the QueryDatabase child service is deployed on the backend node. In this case, the service manager invokes three services: GetGPSCoordinates, QueryDatabase and Schedule service, to accomplish its goal. In Fig. 4-c, the two services hosted on the same device are combined using a façade pattern and now the service manager has to invoke a single service to get the GPS coordinates and the schedule.

The partitioned composite service (Fig. 4-b) without using the façade design strategy and the partitioned composite service (Fig. 4-c) using the façade design strategy are compared by observing the response time to invoke the composite service based on the Tracking service and the Schedule service. The experiments are run with one, five and ten concurrent clients and the results are summarized in Fig. 5. As the number of concurrent clients increases, the performance of the system using the façade design strategy is improved. The response time of invoking the composite service with the façade design strategy is observed to be reduced by 7% for one client and approximately 28% for 5 and 10 concurrent clients in comparison to the response time of invoking the composite service without using the façade design strategy.

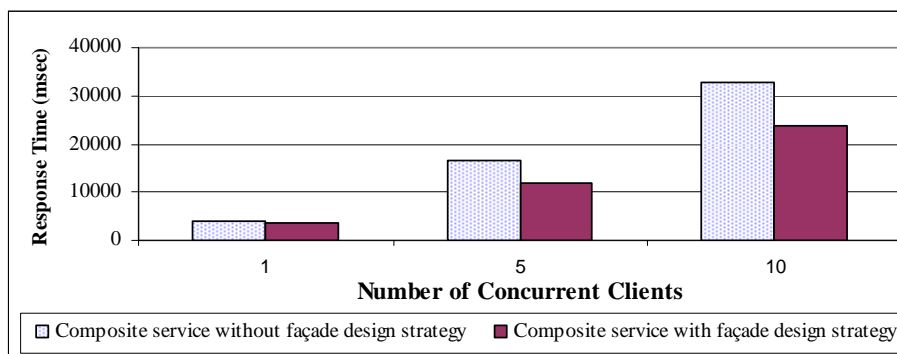


Fig. 5. Comparison of system performance with and without using the facade design strategy.

6. Conclusions and Future Work

This paper discusses a service-oriented architecture for hosting services in wireless environments. The proposed architecture facilitates the use of application partitioning techniques to offload a part of the WS application to a backend surrogate node. It has been demonstrated that the performance of a WS using the coarse grain or fine grain partitioning is much superior to the un-partitioned version of the same WS. The performance of the prototype discussed in Section 5.1 showed that an un-partitioned service give rise to an unacceptable mean response time for higher number of concurrent clients. A dramatic performance improvement is observed with application partitioning as the response time decreased by an order of magnitude when partitioning was introduced. Fine grain partitioning is suitable for partitioning of existing applications that are not based on WS technology or for applications that can only be developed in the device's native code. For the applications investigated in this paper, the system with child WS partitions that uses a light weight WS execution environment is observed to perform better in comparison to the system with child partitions that use the RMI framework for inter-communication. It is also shown that the use of the façade design pattern for services available on the same node can be helpful to improve the performance of composite services especially for a higher number of concurrent clients.

In this paper, we only partition the sample web service into two parts: one for the mobile device and for the backend node. Using more than two partitions forms an important direction for future research. We are investigating a graph theory based approach for partitioning any given application. The goal is to determine both the number of partitions as well as the application components to be grouped into each partition. This paper has used an RMI framework in the implementation of child partitions as remotely accessible applications. In future, it is also worthy to investigate other light weight communication mechanisms for improving system performance.

Acknowledgements

Financial support for this research was provided by Ontario Centres of Excellence and Alcatel-Lucent, Canada.

References

1. M. Asif, S. Majumdar and R. Dragnea, "Hosting web services on resource constrained devices", *In proceeding of 2007 IEEE International conference on web services (ICWS-2007)*, July 9-13 2007, Salt Lake City, Utah, U.S.A. pp 583-590.
2. Business Process Execution Language for Web Services version 1.1, proposed by IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, July 2002, available at <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>. [Accessed: Aug 30, 2007].
3. D. Chandra, C. Fensch, W. Hong, L. Wang, E. Yardımcı and M. Franz, "Code Generation At The Proxy: An Infrastructure-Based Approach To Ubiquitous Mobile Code", *In The 5th ECOOP Workshop on Object-Oriented and Operating Systems (ECOOP-OOSWS02)*, Malaga, Spain, June 2002.
4. Common Object Request Broker Architecture, Specifications available at <http://www.omg.org/docs/formal/04-03-12.pdf>, March 2004. [Accessed: Aug 30, 2007].
5. Distributed Component Object Model (DCOM), Microsoft Corporation, available at <http://msdn2.microsoft.com/en-us/library/ms809332.aspx>.
6. R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", *Ph.D. Thesis*, University of California, Irvine, U.S.A. 2000.
7. E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Professional Computing Series, 1st edition, January, 1995.
8. G. Hunt, and M. Scott, "The Coign Automatic Distributed Partitioning System", *In the Proceedings of the 3rd Symposium on Operating System Design and Implementation (OSDI'99)*, New Orleans, LA, U.S.A., February 1999, pp. 187-200.
9. V. Jamwal and S. Iyer, "Automated Refactoring of Objects for Application Partitioning", *In Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)*, Taipei, Taiwan, December 2005, pp 671-678.
10. Java Platform, Micro Edition (J2ME) by SUN Microsystems Inc., 2006, available at <http://java.sun.com/javame/>, [Accessed: Aug 24, 2007].
11. F. Leymann, "Web Services Flow Language (WSFL1.0)", <http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, 2001. [Accessed: Aug 30, 2007].
12. S. McFaddin, C. Narayanaswami and M. Raghunath, "Web Services on Mobile Devices – Implementation and Experience", *In Proceedings of the Fifth IEEE Workshop on Mobile*

Computing Systems & Applications (WMCSA'03), Monterey, California, U.S.A., October 2003, pp. 100-109.

13. A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. Giuli and X. Gu, "Towards a Distributed Platform for Resource-Constrained Devices", *In Proceedings of International Conference on Distributed Computing Systems (ICDCS02)*, Vienna, Austria, July 2002, pp 43-56.
14. E. Newcomer, G. Lomow, "Understanding SOA with Web Services", December 2004, Addison Wesley Professional.
15. Personal Java Software Development Platform by SUN Microsystems Inc. 1998, *available at <http://java.sun.com/products/personaljava/>*, [Accessed: Aug 30, 2007].
16. A. Silberschatz, G. Gagne, P. Galvin, "Operating Systems Concepts", Seventh Edition, December 2004, John Wiley & Sons, Inc.
17. Andre Spiegel, "Automatic Distribution of Object-Oriented Programs", Ph.D. Thesis. FU Berlin, FB Mathematik und Informatik, December 2002.
18. S. Srirama, M. Jarke and W.Prinz, "Mobile Web Service Provisioning", *In Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, Guadeloupe, French Caribbean, February 2006, pp. 120-128.
19. M. Tatsubori, T. Sasaki, S. Chiba and K. Itano, "A Bytecode Translator for Distributed Execution of 'Legacy' Java Software", *In Processing of the 15th European Conference on Object-Oriented Programming (ECOOP)*, Budapest, June 2001, pp. 236-255.
20. Tanenbaum, M. Steen, "Distributed Systems: Principles and Paradigms", 1st edition, January 2002, Prentice Hall.
21. E. Tilevich and Y. Smaragdakis, "J-Orchestra: Automatic Java Application Partitioning", *In Proceedings of the 16th European Conference on Object-Oriented Programming*, Malaga, Spain, June 2002, pp. 178-204.
22. T. Watson, "Effective Wireless Communication through Application Partitioning", *In Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*, Orcas Island, Washington, U.S.A., May 1995, pp. 24-27.
23. J. Wortmann, H. Pels, H. Jagdev, "Collaborative Systems for Production Management", 2003, Springer..
24. C. Young, Y. Lakshman, T. Szymanski, J. Reppy, D. Presotto, R. Pike, G. Narlikar, S. Mullender and F. Grosse, "Protium, an infrastructure for partitioned applications", *In Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, Elmau/Oberbayern, Germany, May 2001, pp.47-52.