Towards Peer-to-Peer Long-Lived Mobile Web Services

Fahad Aijaz, Bilal Hameed, Bernhard Walke
RWTH Aachen University, Faculty 6
Communication Networks
Kopernikusstr. 16, 52074 Aachen
{fah, bhd}@comnets.rwth-aachen.de

Abstract

Mobile phones in today's era are not just small devices that provide the means of communication, rather, they are equipped with more processing power, storage capacity and battery performance. Now, the hand held devices are not only service consumers but are also capable of hosting and providing services to their peers. These services deployed on mobile devices bring in the idea of Mobile Web Services (Mob-WS) to the research community.

This paper concentrates on a middleware for long-lived Mob-WS that are accessible asynchronously over the network. Since the synchronous Mob-WS are not feasible for long durational tasks, therefore a concept and architecture of controllable and monitor able asynchronous Mob-WS middleware is presented. The service interaction techniques are discussed followed by the middleware subsystem and high-level architecture and control flow.

The presented middleware is a potential basis for innovative mobile applications, therefore, a proof-of-concept prototype of asynchronous Mob-WS application is developed and presented with a special focus on network optimization for Wireless Sensor Networks (WSN).

1 Introduction

Mobile phones in today's era are not just small devices that provide the means of communication, rather, they are equipped with more processing power, storage capacity and battery performance. The continuous growth in mobile technology has introduced variety of research domains that strongly focus on developing innovative mobile applications and services that are human centric and closer to the user's personal needs. In such a dynamic and rapidly growing ubiquitous mobile environment, hand held devices are not only service consumers over the Internet, wireless network or within the operators' network, but are also capable of hosting and providing services to their peers. Such services, when hosted on mobile devices are often termed as Mob-WS.

In this paper, a concept and architecture of a middleware for asynchronous Mob-WS is introduced to perform long-lived operations. The work is based on the existing Mob-WS framework first presented in [4]. Since synchronous Mob-WS are not a feasible choice for long durational tasks, therefore a need for asynchronously accessible Mob-WS arises. These Mob-WS demand mechanisms for control and monitoring at runtime, since the requirements of service consumer may change dynamically. In this manuscript, we propose a mobile middleware to develop and deploy such controllable and monitor able asynchronous Mob-WS and further present a proof-of-concept prototype for network optimization.

2 Mobile Web Services

The concept of Mob-WS involves two basic roles, termed as Web Service Consumer (WS-C) and Web Service Provider (WS-P). The interaction between these roles can be classified in three distinct ways that are illustrated in figure 1. Within the context of this work, we will focus on Peer-to-Peer (P2P) interaction, where both the WS-C and WS-P are mobile nodes.

The general web services based communication is transparent to the interaction strategy and therefore does not care if the WS-C and WS-P are fixed or mobile nodes. In most of the cases, the published web services are developed to be synchronous in nature, that is, a web service client is blocked until a response is received from the service provider or the connection timeout occurs. This approach is feasible for time critical

978-1-4244-1841-1/08/$25.00 ©2008 IEEE

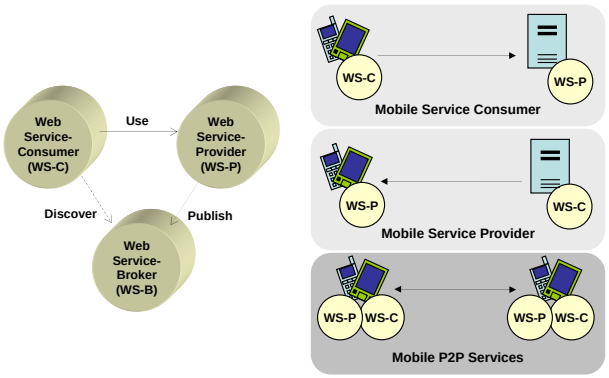

Figure 1. Classification of Mobile Web Services

and short-lived services, whereas for complex long-lived services that take some time to complete, synchronous Mob-WS are not the right choice as they block other operations that could be performed during the waiting time (see figure 2). Furthermore, there is no mechanism in synchronous way of interaction that enables the dynamic control and monitoring of Mob-WS at runtime. For such long-lived Mob-WS, asynchronous interaction strategy becomes a natural choice.

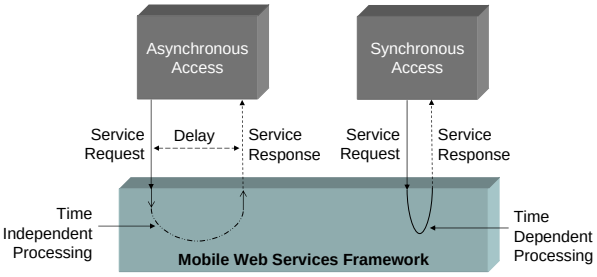

Figure 2. Service Access Mechanisms

Within the context of this work, a middleware has been proposed that enables the development and deployment of long-lived asynchronous Mob-WS on mobile devices. In addition, the middleware provide mechanisms for management, control and monitoring of these long-lived Mob-WS at runtime. In order to conform to web service standards, the middleware utilizes the Asynchronous Service Access Protocol (ASAP) [3] which is currently a proposed draft from OASIS[1]. The asynchronous Mob-WS middleware reduces the development effort and cost for applications that require time independent communication strategy by providing underlying communication architecture.

[1] http://www.oasis-open.org

3 Service Interaction

For services to communicate asynchronously, interaction techniques like "Callback" and "Polling" are usually adapted. Each of these techniques have their own benefits depending upon the scenarios they are used in.

In case of "Callback", an asynchronous interaction based on the principle of *Don't call us, we will call you back* is used. In such scenarios a client application sends a request and waits for the response without blocking its state. The response is send back to the client when it is available. Callback results in increased processing efficiency since the service solely performs its tasks and is not bogged down in processing and responding to status updates. This also results in less network traffic which in turn requires less amount of resources. However callback may result in delayed retrieval of response at the client end, specially when the service has to inform a lot of clients about the status of the service.

Where as in "Polling" the client application repeatedly asks for the status of an external service that it has invoked. In this technique, there is no way for external service to notify the client about its status. Polling results in increased traffic over the network, which decreases the processing efficiency of the service since it has to process and respond to status update requests. It also requires high resource allocation such as more powerful processors and more network bandwidth. However polling results in an immediate retrieval of response, or retrieval of response within an acceptable time frame which is determined by the time spacing between two polling requests.

The proposed asynchronous Mob-WS middleware provides support for both Polling and Callback mechanisms and leaves the choice to ignore one and rely solely on the other technique on the service and application developers. Services that need instant response notifications could implement polling whereas services that need to be deployed on resource critical devices could rely on callbacks alone.

4 P2P Asynchronous Mobile Web Services Middleware

The asynchronous Mob-WS middleware provides a foundation to develop and deploy asynchronous long-lived services, not asynchronous messaging. This is because asynchronous messaging does not necessarily imply that the deployed services are asynchronous in nature. Since asynchronous services start and continue to perform their tasks over a longer duration

of time, therefore, the need to be able to monitor and control such services becomes essential. This section presents the architectural details of the proposed asynchronous Mob-WS middleware, their control and monitoring mechanisms and service interactions techniques. The high-level architecture is presented in this manuscript since the focus is solely on the concept and architecture. The implementation and further technical details are planned to be published in a separate paper.

4.1 Middleware Subsystem

In order to build a system of asynchronous Mob-WS three different types of endpoints are defined to cater for the three distinct roles. These are the Instance, Factory, and Observer components which form a subsystem within the middleware architecture. Figure 3 illustrates the interaction between these three components.

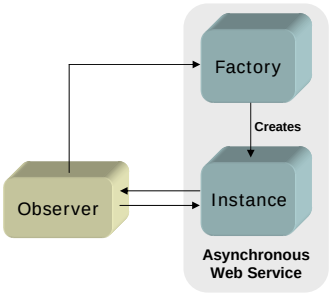

Figure 3. Middleware Subsystem

4.1.1 Instance

Instance resource is the actual performance of work [3]. Instance embodies the contextual information that distinguishes one execution of an asynchronous service from another. On each invocation of an asynchronous Mob-WS, a new instance is created having its own unique Endpoint Reference (EPR) and context data. It is this service instance that is used by observers to send messages to monitor and control the status of an asynchronous Mob-WS. A service instance could be created, paused, resumed or terminated, which in turn controls the Mob-WS according to the corresponding control message. Under normal conditions, the asynchronous Mob-WS eventually completes its task and the service instance thereby notifies all the clients.

4.1.2 Factory

Factory provides a way of doing some work and the potential for getting that work done [3]. Factory is a focal point in the entire asynchronous system and could be seen as a manager. It creates instance resources that carries out the actual tasks, maintains a list of all service instances, and could be used to search for specific service instances. Factory has a fixed and unique EPR. Although Factory embodies the knowledge of how the work is performed, it does not do the work itself. It acts as a manager and not as a worker, the service instance is the worker that does all the work.

4.1.3 Observer

Observer provides client application a way to communicate with service instance, and a way to the service instance to communicate with the client application reactively. Observer might query the current status of the service instance or receive notifications from it, therefore, observer subscribes to service instance by utilizing Web Service Addressing [2] properties.

4.2 System Architecture

Figure 4 provides a general overview of the core architecture of the asynchronous Mob-WS middleware. The HTTP and UDP interfaces waits for incoming HTTP/UDP requests from the clients. The Mob-WS client could send a request either over Hypertext Transfer Protocol (HTTP) or User Datagram Protocol (UDP) since the Mob-WS invocation is independent of the transport protocol. The client's request, when reached at the server, is delegated to the Request Handler that determines the request type, that is, asynchronous or synchronous. In case the requested message is directed toward a synchronous service, the control flow continues as described in [1]. Else, the Request Handler passes the message to the ASAP Handler which decides if the message is meant for the Factory or the Instance. After determining the proper recipient of the requested message it is passed to the specific Factory or Instance for further processing. The Factory or Instance performs the requested task and sends the result back to the ASAP Handler which further passes it to the Response Handler. The Response Handler after receiving the response data from the ASAP Handler, sends it back to the client over any transport protocol preferred by the client. The Mob-WS that are initiated at the time of start-up communicate with the Deployment Interface to serve the client request. This is however vital to note that as soon as the asynchronous

Mob-WS is invoked, the client is notified about the status of the service. This releases the service client from blocked state whereas the service continues to perform its tasks at the server.

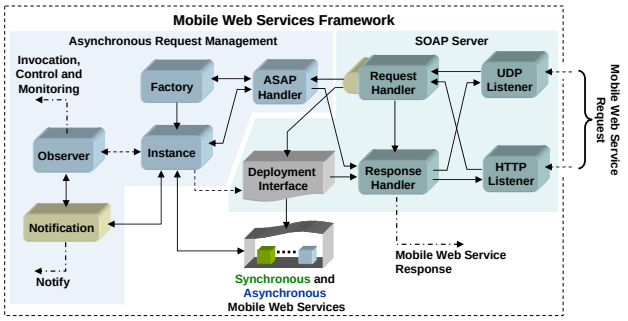

Figure 4. System Architecture

4.3 Dynamic Service Management

As highlighted earlier in section 4, that, for an asynchronous long-lived Mob-WS it is essential to be controllable and provide mechanisms for monitoring its state. This is because during the execution of such services, the service consumer might be interested in receiving status updates or the requirements of the client might change. For instance, in a sensor based environment where the real-time information is constantly changing, an asynchronous Mob-WS client might want a service to utilize the updated context information that it has just received from its environment and discard the one that was previously sent at the time of service creation. In order to meet such dynamic change in requirements, a client should be able to send control messages to the service instance and on the other hand, the service instance should be capable of updating itself at runtime.

The asynchronous Mob-WS middleware incorporates the control and monitoring mechanisms within its architecture that makes the deployed services capable of adapting to dynamic changes. In order to elaborate the control and monitoring message flow, some implementation level details have to be considered.

4.3.1 Service Monitoring

Consider an example in which a service client wishes to receive the properties of the service that is currently in execution phase. Once the status information is received by the client, it can then analyze and act accordingly. The client might wish to control the service based on its current state. Such monitoring requests can be send as Simple Object Access Protocol (SOAP) envelopes from the peer Observer for either the Service Instance or Factory (refer to section 4.1). In order to get the properties of the service, a corresponding SOAP envelope is constructed and send to the middleware. This request message once received by the ASAP Handler is analyzed and passed on to the respective resource sought by the Observer. The example SOAP envelopes for request (GetPropertiesRq) and response (GetPropertiesRs) are shown in figure 5 and 6 respectively.

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
   <SOAP-ENV:Header>
      <!--  WS-Addressing request headers -->
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <computePerf xmlns="urn:Services" id="o0"
       SOAP-ENC:root="1">
          <getPropertiesRq xsi:type=":getPropertiesRq"/>
       </computePer f>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5. Service Monitoring Request

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
   <SOAP-ENV:Header>
      <!--  WS-Addressing response headers -->
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
   <computePerfResponse xmlns="urn:Services" id="o0"
   SOAP-ENC:root="1">
     <getPropertiesRs xsi:type=":getPropertiesRs">
       <EPR xsi:type="xsd:string">
           http://www.comnets.rwth-aachen.de/instanceEPR
       </EPR>
       <Name xsi:type="xsd:string">
           Performance Computation Service
       </Name>
       <Description xsi:type="xsd:string">
           Computes the network performance
       </Description>
       <ContextData xsi:type=":ContextData">
           <!-- Extensible Element -->
       </ContextData>
       <ResultData xsi:type="xsd:string">
           Status: Computation under process.
           <!-- Extensible Element -->
       </ResultData>
     </getPropertiesRs>
    </computePerfResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6. Service Monitoring Response

4.3.2 Service Control

Now for controlling a service, consider a scenario in which a service client wishes to change the state of the running service, that is, service control. Such control

message could involve activities from setting various properties of the service to pausing or terminating the service. In such case, a control request as a SOAP envelope from the client is received by the ASAP Handler that describes to which state the service should be changed. An asynchronous Mob-WS at any given time could be in the states [3]; *open.notrunning, open.notrunning.suspended, open.running, closed.completed, closed.abnormalCompleted or closed.abnormalCompleted.terminated*.

After changing the state of the asynchronous web service the current state of the asynchronous Mob-WS is passed back to the ASAP Handler which delegates it to the Response Handler to be sent to the client that triggered this control message sequence.

Any change in state of the service triggers a Callback StateChanged message to be sent to all the clients that have subscribed to this particular asynchronous Mob-WS. A change in state could be caused by some internal event such as occurrence of an exception during processing, or could be triggered externally by a client. As soon as the service instance changes the state of the asynchronous Mob-WS, the notification is send to all the clients. The clients acknowledge the reception of this message to the service instance. Figure 7 shows a sample `changeStateRq` SOAP messages. The response SOAP message follows exactly the same structure except that the message name is changed to `changeStateRs`.

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
   <SOAP-ENV:Header>
      <!--  WS-Addressing request headers -->
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <computePerf xmlns="urn:Services" id="o0"
      SOAP-ENC:root="1">
          <changeStateRq xsi:type=":changeStateRq">
            <State xsi:type="xsd:string">
              closed.completed
            </State>
          </changeStateRq>
       </computePerf>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 7. Service Control Request

5 Prototype

The asynchronous Mob-WS middleware could be a potential foundation for network performance optimization based on in-network computations. As a special case, we will focus on the WSN. WSN are no longer limited to being data networks where sensors merely act as a source of data. Instead, these networks are increasingly being developed and deployed to fulfill application specific tasks.

A proof-of-concept prototype has been developed to perform collaborative in-network computations with WSN as its special case. It has been assumed that the asynchronous Mob-WS middleware is deployed on the collaborative computation nodes within the WSN which are better equipped in terms of processing power and battery consumption than the ordinary sensors. The middleware is used to monitor the state of computations. In case new sensor data is available, the computation node is updated by sending control messages. Once the service ends, the analyzed result could be send to some higher hierarchy as result data to further evaluate network performance. The prototype does not have a GUI as it functions as a background process.

6 Conclusion

In this paper, a concept and architecture of an asynchronous Mob-WS middleware is introduced that discusses the idea of deploying long-lived complex asynchronous Mob-WS on mobile devices. The asynchronous Mob-WS interaction techniques are also presented and briefly compared. Furthermore, the middleware architecture is elaborated and control flow within the major components is described. The presented middleware enables control and monitoring of P2P long-lived asynchronous Mob-WS and therefore forms potential basis for innovative mobile applications covering variety of domains and reduces development cost. A prototype has been developed with WSN as a special case. Performance evaluation of the middleware is planned.

References

[1] G. Gehlen, F. Aijaz, and B. Walke. An enhanced udp soap-binding for a mobile web service based middleware. In *Proceedings of IST Mobile Summit 06*, page 8, Myconos, Greece, Jun 2006. ComNets, Faculty 6, RWTH Aachen University, Germany. 4.2

[2] M. Gudgin, M. Hadley, and T. Rogers. Web Services Addressing 1.0 - Core. Published on the internet, May 2006. W3C Recommendation. 4.1.3

[3] K. S. F. S. J. R. John Fuller, Mayilraj Krishnan. Oasis asynchronous service access protocol (asap). 2, 4.1.1, 4.1.2, 4.3.2

[4] L. Pham and G. Gehlen. Realization and Performance Analysis of a SOAP Server for Mobile Devices. In *Proceedings of the 11th European Wireless Confernce 2005*, volume 2, pages 791–797, Nicosia, Cyprus, Apr 2005. VDE Verlag. 1