Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Volume 21, No. 1, ISSN 1726-9679
ISBN 978-3-901509-73-5, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2010
Make Harmony Between Technology and Nature, and Your Mind will Fly Free as a Bird
*Young Researches and Scientist Paper / * Supervisor, Mentor*

WEB SERVICES ENGINE FOR THE ANDROID PLATFORM

RADOVICI, A[lexandru] & DRAGOI, G[eorge]*

Abstract: *Mobile systems have become more and more important. This paper describes a platform for implementing web services among mobile devices running the new Android framework. While other ports of web containers use classic approaches for creating the actual services, our system is design to use Android specific components.*
Key words: *Android, mobile, web, service, platform*

1. INTRODUCTION

Mobile devices have become more and more important. Together with the evolution of software and hardware, mobile phones are basically considered as computers that fit into one's pocket. Given the increased usage of these devices, the problem of creating software and services for them has arisen. Building applications for these mobile devices has to take into account the following facts: limited long term processing power, limited memory footprint, real time interruptions (an incoming phone call), different kind of input methods (usually there is no keyboard available, rather small screen size etc). Mobile device software and hardware is very heterogeneous thus communication between devices is rather difficult. At the time of writing, the great majority of mobile devices communicate with computers using some proprietary software and protocols. We are trying to change this fact by implementing web service communication to these devices.

The paper will present the implementation of a web service container for the Android mobile platform. We have chosen this system due to the fact that it is open source and is expected to gain a significant market share in the next few years.

2. ANDROID PLATFORM

Android is the mobile platform developed by Google. Based on an ARM optimized Linux kernel and a simplified user space library called *Bionic*, Android brings to life a new java virtual machine called *Dalvik* and together with it a new way of designing applications (Android Fundamentals, 2009). This virtual machine is optimized to run with a very small memory footprint and an improved garbage collection system. As it is based on Linux, Android supports normal C/C++ linked software. What makes Android different from any other mobile system is its large collection of libraries and components provided by the Dalvik virtual machine and its new approach to software design. Applications are a collection of components that can interact and be run independently. We can distinguish five types of components: *Activities*, *Services*, *Content Providers*, *Intents* and *Broadcast Receivers* (Murphy 2009). Moreover, applications can make use of components from other applications, just as if they where inside the same program. This feature can be used instead of dynamic class loading. The most important component in implementing a web service platform is creating a web service container. In essence this is a web server specialized for delivering XML content, this being the language for web service communication.

3. WEB SERVICE CONTAINER

The only web server implementation that currently exists on Android is iJetty (iJetty 2010). This is a standard port of the common web container Jetty, which uses dynamic class loading and web applications. It has no optimization what so ever for web services. Another downside of this server is the usage of dynamic class loading. Dalvik does not use the standard Java .class files for storing classes, instead using the .dex format. This means that java standard class loading engine is not available. Google has provided a mechanism for dynamic class loading, but does not recommend it as the SDK might change. Usage of Android application components is recommended. Moreover, as Android applications are more or less sandboxed (Brunette 2009), the usage of dynamic class loading requires the services to be shipped along with the server's package. This is not acceptable for us. To overcome these problems, we have decided to implement our own web service container, called *EasyWeb*. As it is specially designed to run on mobile device, the server implements only a subset of the large HTTP protocol. This includes only the essentials needed for request/response actions. *EasyWeb* is composed out of two components: the server itself, implemented as a *Service* and the settings interface (GUI) implemented as an *Activity*. This separation of components takes advantage of the Android component types and life cycles (Android Fundmentals 2010, Burnette 2009): the server is running in background and is started only when needed, the GUI runs as an Activity and is loaded only very rarely. Mobile devices have limited hardware resources and processing power. Thus instead of creating a thread for each request, our web server uses the *Worker Pool* strategy of managing connections. A limited number of threads are created pushed into a pool. When a request is received, it is queued until a thread becomes free for processing it. The tests we have performed have achieved the best results two to four working threads. Web Services have to implement the *EasyWebApplication* interface illustrated in figure 1.

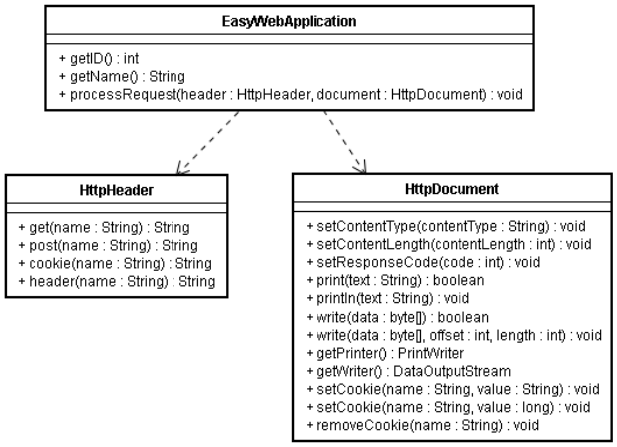

Fig. 1. Web Service proxy class its associated classes

As dynamic class loading is avoided, services are written as Android components, *Content providers* or *Services* using proxy classes on the server side as well as on service implementation side. *These* have to be first registered with the server, before being used. The server component is controlled via an AIDL interface (Murphy 2009). Besides the *start* and *stop* functions, the server exports functions for adding and deleting services.

4. IMPLEMENTING WEB SERVICES

4.1 Content Provider & Service Implementation

An Android *Content Provider* component is in fact a database that provides information upon request, similar to an SQL query. The query result is a cursor, or in other words, a result set. This can contain text and binary data as well. In order to use content providers for implementing web services, we have built a proxy class inside the server. This class transforms the HTTP request into query and sends it to the content provider where it is processed and a table with one row and one column is returned. This column contains the data that needs to be passed as a result for the HTTP request.

To make life easier for the programmer, we have built another proxy class on the content provider side that transforms the received query into a standard HTTP request and provides the typical functions for generating the HTTP response, converting it into a result set that is passed back to the server. Figure 2 illustrates the calling of such a web service. The request processor calls the server proxy class, the class searches for the Content Provider, sends the query and waits for the result. In turn, the Content Provider proxy converts the query into a HTTP request and runs the service (*processRequest*()).

Another method of implementing the actual web service is using the android *Service* component. This has the advantage of running in its own thread or process, separated from the service container. This means that the worker thread can pass the request and handle another request while the *Service* processes. The worker thread will be notified when the result is ready to be sent to the client. Communication between the service container and the actual web service is done via two proxy classes, one inside the server and the other one inside the service. The latter implements an AIDL interface used by the first one. The important advantage of this method is that the server proxy class is able to suspend its thread until the service processes the request. Besides being able to be used by computers, web services results should be able to be used by humans as well.

The best way of doing this would be to generate HTML responses instead of XML ones. Still, this would mean serious modification from the service point of view. Moreover, creating and delivering good HTML interfaces requires a considerable processing power from the phone.

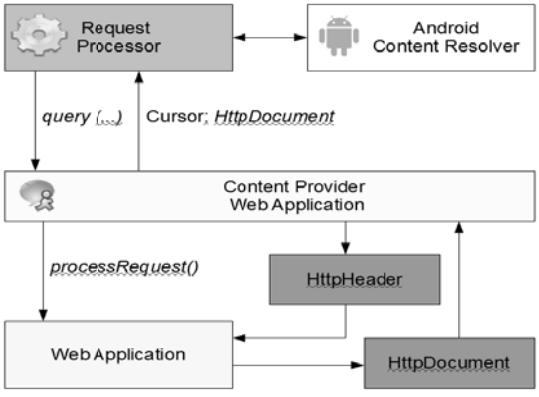

Fig. 2. Web Service implemented as a Content Provider

To overcome this problem, we have used the XSLT transform (Kay 2004). Style sheet transformations transfer the user interface processing to the client, in fact to the browser. The server provides a normal XML response and together with it a style sheet (XSL) with directions on how the data should be transformed into HTML. Based upon our tests, we can say that the processing power required on the phone diminishes by about 20 times.

4.2 Security Issues

When it comes to mobile phones, security is a very important issue. If not controlled properly, programs running on the phone might cause serious damage by releasing private data on the network or accessing paid network services. When it comes to *Service* components that are running as separate applications, permissions applied by Android (Security and Permissions 2010) on these service applications are not enough. Due to the fact that they are able to communicate with the outside world through the server, the user has to be aware of this. This is why we have implemented a supplementary, but yet not enough, security layer.

If the service is implemented as a *Content Provider*, which is a passive component, security is straightforward. The user is the one that has to register the content provider to the server. On the other hand, if the service is an Android *Service*, thus an active component, the service is the one that has to register with the server. In order to do this, the server will require a passkey that is known by the user. The service will have to ask the user for the passkey in order to register. This way assures that the user is notified of the registration.

5. CONCLUSION

With so many types of mobile devices being developed, a common communication platform between them and other computer-powered devices is necessary. The best solution known so far is the Service Oriented Architecture. This paper has presented the implementation of o web service platform for the Android framework. We have created a specialized service container, or server, which obeys and uses the most of the Android technology. Dynamic class loading is avoided by using Android specific components for implementing the actual services. This creates also some security issues that have been partially dealt with.

Last but not least, we have proposed a way in which web services can be used by computers and by humans as well, without any modification on the code or any overhead on the mobile device. To sum up, we can say that with this paper we have followed some steps in creating a serious web service mobile platform, platform that will allow mobile devices to communicate with each other regardless of their brand, operating system or software.

6. REFERENCES

Burnette, E.; (2009). *Hello, Android: Intrdoucing Google's Mobile Development Platform*, The Pragmatic Bookshelf, ISBN: 1-934356-49-2, USA

Kay, M.; (2004). *XSLT 2.0 programmer's Reference, Third Edition*, Whiley Publishing, ISBN: 0-764-56909-0, USA

Murphy, M.; (2009). *Beginning Android*, Apress, ISBN: 978-1-4302-2419-8, USA

*** (2010) *http://developer.android.com/guide/topics fundamentals.html* - Android Developers - Application fundamentals, *Accessed on: 2010-01-10*

*** (2010) *http://developer.android.com/guide/topics/security/security.html*, Security and Permissions, *Accessed on: 2010-03-28*

*** (2010) *http://code.google.com/p/i-jetty/*, Project Hosting on Google Code – iJetty, *Accessed on: 2010-01-10*