SOA and its usage in mobile services

José María Márquez Toro
Technische Universität Berlin
coripeno87@gmail.com

***Abstract*—Nowadays, the term or acronym SOA is widely used and has become a mainstream technology[1], but there is not a lot of precision in the way that it is used[2]. "The World Wide Web Consortium (W3C) refers to SOA as *'A set of components which can be invoked, and whose interface descriptions can be published and discovered'*[3]. We see similar definitions being used elsewhere; it is a very technical perspective in which architecture is considered a technical implementation. This is odd, because the term architecture is more generally used to describe a style or set of practices"[2]. In this paper, I introduce the concept and the keys of SOA and its usage in mobile services. Also, I explain briefly what is a web service. Finally, I use RESTEasy and Jersey technologies to implement a few web services that show the principles of SOA.**

I. Introduction to SOA

"Service-oriented architecture (SOA) is an evolution of distributed computing based on the request/reply design paradigm for synchronous and asynchronous applications. An application's business logic or individual functions are modularized and presented as services for consumer/client applications. The key of these services is their loosely coupled nature; i.e., the service interface is independent of the implementation. Application developers or system integrators can build applications by composing one or more services without knowing the services' underlying implementations. For example, a service can be implemented either in .Net or J2EE, and the application consuming the service can be on a different platform or language"[4].

A service has been defined "as an application's business logic or individual functions that are modularized"[4]. The question is if there is a more specific definition for a service. The answer is yes, a good definition for a service is provided by W3C. W3C refers to service as "an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities"[3]. Therefore, a service can be saw like subapplications or components, which are used in other applications.

The Figure 1 shows four basic services. These services can be used by clients or developers using Internet. The student service could provide related services like loading all students, saving a student, etc., while the course services could provide related services like loading all courses. The account services could load/save accounts, calculate assessments, etc., and finally, the security service could provide authentication and authorization using enterprise library's authentification provider[5].

The key characteristics of SOA services are the following:

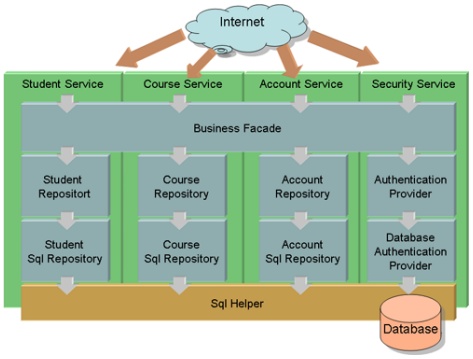

Figure 1. Example of four services[5].

- SOA services have self-describing interfaces that are implementation-independent[4][6].
- Services are self-contained and loosely coupled[6].
- Services can be dynamically discovered[6].
- Composite services can be built from aggregates of other services.[6]
- SOA services communicate with messages. Communication among consumers and providers or services typically happens in heterogeneous environments[4].
- SOA services are maintained in the enterprise by a registry that acts as a directory listing. Applications can look up the services in the registry and invoke the service[4].
- Each SOA service has a quality of service (QoS) associated with it.[4]

A. Why SOA?

People may ask why they should use SOA, or about the benefits of SOA. "The reality in IT enterprises is that infrastructure is heterogeneous across operating systems, applications, system software, and application infrastructure. Some existing applications are used to run current business processes, so starting from scratch to build new infrastructure isn't an option. Enterprises should quickly respond to business changes with agility; leverage existing investments in applications and application infrastructure to address newer business requirements; support new channels of interactions with customers, partners, and suppliers; and feature an architecture that supports organic business"[4]. Is supposed that SOA with its loosely coupled nature allows enterprises to plug in new services or upgrade existing services in a granular fashion to address the new business requirements, provides the option to make the services consumable across different channels, and exposes the

existing enterprise and legacy applications as services, thereby safeguarding existing IT infrastructure investments[4].

To use efficiently a SOA, the following requirements must be met, in other case, the implementation of the SOA could be failure.

- "Interoperability between different systems and programming languages provides the basis for integration between applications on different platforms through a communication protocol. One example of such communication is based on the concept of messages"[7]. Using messages across defined message channels should decrease the complexity of the end application, thereby allowing the developer of the application to focus on true application functionality instead of the intricate needs of a communication protocol[7].
- "Desire to create a federation of resources. Establish and maintain data flow to a federated data warehouse. This allows new functionality developed to reference a common business format for each data element."[7].

B. Web services approach

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network[3], ie, web services make software functionality available over the Internet so that programs like PHP, ASP, JSP, JavaBeans, the COM object, and all our other favorite widgets can make a request to a program running on another server (a web service) and use that program's response in a website, WAP service, or other application[8]. So, if I need to implement a task of a project and I want to save time, or I dont know how implement this task, I can use a Web service that provide me the functionality I need. The Web service is called over internet, and by making requests with data to the web service, I can expect a response with the result calculated by the webservice. For example, I can use any of Amazon's Web services in my project to implement a task[9].

II. SOA IN MOBILE APPLICATIONS

"In a SOA, a mobile service is equivalent to an application realized by combining several services. Mobile services are composed of several standardized components, which can be interconnected at runtime to accommodate user movements. However, mobile services pose additional requirements to the service architecture, which are not necessarily supported by service architectures for other services like enterprise and business services. For example, mobile services should tackle several types of movements:

- Movement of users between devices.
- Movement of devices across networks.
- Movement of services across domains"[10].

Mobile computing can improve the service offered to customers. For example, diners can pay their meal using the wireless connection of their phone. So, being mobile is great, but is challenging. "First, the connectivity is not guaranteed. Connectivity disruptions can happen at anytime. So the application has to handle such disruptions. Mobile devices have smaller resources and screens than desktop. Moreover, user interactions are quite different.Another problem is the fragmentation of the mobile market. There is a lot of different devices. Providing global solution requires a good knowledge of the difference between such models. SOA simplifies the way mobile solutions are designed and developed. The loose coupling promoted by SOA is one of the key points of modern applications. This should be also true for mobile applications. Is supposed that loose coupling simplifies the reuse of existing services, allows abstracting implementations details, and eases the composition of mobile solutions"[11].

"SOA can be used in several locations in mobile solutions: On the device part, where the applications can rely on features provided by the operating system or by others applications. As an example, an application does not have to contain the "take a picture" functionality, but can just use the "take a picture" service provided by the operating system (OS), which deals with the built-in camera. On the server-side, SOA has already demonstrated its utility. By relying on decoupled (local or remote) components, enterprise applications can win in term of modularity, maintainability and clarity. The middleware used between the mobile devices and the servers may also use services"[11].

A. SOA on devices

"The new generation of mobile devices such as the iPhone and Android phones promote the reuse of existing functionalities by providing SOA-like interactions. These operating systems provide services to access built-in features of the phone. Android promotes component-based development. The components are loosely coupled. An intent mechanism allows components to interact together. When a component wants to use any other features, it describes an intent and then invokes it (the invocation depends on the type of targeted component). Moreover, Android proposes the concept of service for background tasks[11].

B. SOA on the server side

Using SOA in enterprise application became very common. The majority of new enterprise applications follow this trend: components of the application should use remote or local services. This allows separate evolution of the different part of the application, as well as a clear separation between the functionality and the implementation.Is supposed that the popularization of web services had a great impact on the development of such applications[11].

C. SOA in the communication middleware

The middleware is responsible for the communication between the two sides of the system. Is supposed that such middleware has to be very flexible as it should be a critical part of the system. A bad choice here could be definitely fatal for the whole system. This middleware does not manage only the data transport, it can also ensure quality of service, support security, pre-process messages, and help to be scalable by dispatching event in a smart way, etc[11].

III. Implementing SOA

Developers can implement SOA using different technologies, like: SOAP (Simple Object Access Protocol), RPC (Remote procedure call), REST (Representational state transfer), DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture), DDS (Data distribution service) or WCF (Windows Communication Foundation). Following, I introduce the two most popular technologies in the last years: SOAP and REST[14][12].

A. SOAP (Simple Object Access Protocol)

"SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols"[13], like HyperText Transfer Protocol (HTTP) or SMTP (Simple Mail Protocol). "The framework has been designed to be independent of any particular programming model and other implementation specific semantics".[13].

"Two major design goals for SOAP are simplicity and exttomensibility. SOAP attempts to meet these goals by omitting, from the messaging framework, features that are often found in distributed systems. Such features include but are not limited to "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs)"[13].

"The SOAP specification defines the SOAP messaging framework consisting of:

1) The SOAP processing model: The SOAP processing model specifies how a SOAP receiver processes a SOAP message. It applies to a single message only, in isolation from any other SOAP message. The SOAP processing model itself does not maintain any state or perform any correlation or coordination between messages, even, for example, when used in combination with a SOAP feature which involves sending multiple SOAP messages in sequence, each subsequent message depending on the response to the previous message. It is the responsibility of each such feature to define any combined processing[13].

2) The SOAP extensibility model: The SOAP extensibility model provides two mechanisms through which features can be expressed: the SOAP Processing Model and the SOAP Protocol Binding Framework. The former describes the behavior of a single SOAP node with respect to the processing of an individual message. The latter mediates the act of sending and receiving SOAP messages by a SOAP node via an underlying protocol[13].

3) The SOAP binding framework: SOAP enables exchange of SOAP messages using a variety of underlying protocols. The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is called a binding. The SOAP Protocol Binding Framework provides general rules for the specification of protocol bindings; the framework also describes the relationship between bindings and SOAP nodes that implement those bindings. The HTTP binding in SOAP illustrates the specification of a binding[13].

4) The SOAP message construct: A SOAP message is specified as an XML infoset whose comment, element, attribute, namespace and character information items are able to be serialized as XML 1.0. Note, requiring that the specified information items in SOAP message infosets be serializable as XML 1.0 does not require that they be serialized using XML 1.0. A SOAP message Infoset consists of a document information item with exactly one member in its property, which must be the SOAP Envelope element information item"[13].

B. REST (REpresentational State Transfer)

"REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. If measured by the number of Web services that use it, REST has emerged in the last few years alone as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP[12] and WSDL-based interface design because it is a considerably simpler style to use"[15][14].

"A concrete implementation of a REST Web Service follows four basic design principles:

- Use HTTP methods explicitly.
- Be stateless.
- Expose directory structure-like URIs.
- Transfer XML, JavaScript Object Notation (JSON)[16], or both"[14].

"One of the key characteristics of a RESTful Web Service is the explicit use of HTTP methods. HTTP GET, for instance, is defined as a data-producing method that is intended to be used by a client application to retrieve a resource, to fetch data from a Web server, or to execute a query with the expectation that the Web server will look for and respond with a set of matching resources. REST asks developers to use HTTP methods explicitly and in a way that is consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods. According to this mapping:[14].

- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE"[14].

IV. Tools used in the code demostration

I am going to use RESTEasy and Jersey to implement the SOA code demostration. I choosed them because the are very simply to use. My goal is implement one server and its client in RESTEasy, and another servier and its respective client in Jersey. Following, I make a short introduction of this two technologies:

A. RESTeasy

RESTEasy is a JBoss[17] project that help you build RESTful Web Services and RESTful Java applications. RESTEasy is an portable implementation of JAX-RS[18] specification which provides a Java API for RESTful Web Services over the HTTP protocol[19][20].

"RESTEasy can run in any Servlet container, but tighter integration with the JBoss Application Server is also available to make the user experience nicer in that environment. While JAX-RS is only a server-side specification, RESTEasy has innovated to bring JAX-RS to the client through the RESTEasy JAX-RS Client Framework. This client-side framework allows you to map outgoing HTTP requests to remote servers using JAX-RS annotations and interface proxies"[19].

B. Jersey

"Jersey is Sun's production quality reference implementation for JSR 311[23], JAX-RS, The Java API for RESTful Web Services. Jersey implements support for the annotations defined in JSR-311, making it easy for developers to build RESTful web services with Java and the Java JVM. Jersey also adds additional features not specified by the JSR.[21]"

V. Implementation of SOA code example

I used Jersey and RESTEasy to implement two servers and two clients to show the principles of SOA, using RESTful Web Services. The two technologies are very similar, as both are implementations of JAX-RS specification. As the implementation must be shown to the partners, I tried to keep it the most simple possible. Following, I resume the implementation of the two examples that I propose:

A. Implementation with Jersey

Jersey can be used with the embedded Grizzly server. I don't know anything about Grizzly server, so I decide use Jersey with an Apache Tomcat server. I need a IDE to write the code, of course. I have experience in Eclipse, so I start to look how I could integrate Jersey and Tomcat with Eclipse. I find a useful tutorial [22] and I try to make work the "Hello Jersey" example. I spent the most of time configuring Eclipse with the Tomcat server, and installing JAVA JDK. There are many ways to integrate Eclipse with Jersey, I decide to create a Dynamic Web Project, then I import the JARs used by Jersey and finally, I modify web.xml file with the Jersey configuration.

After a couple a days, finally I make work the example and I start to understand how the server works. After that, I start to develop a simple example to show to my partners. In RESTful Web Services there are to ways to pass data, XML and JSON. I choose XML because I think that XML will be more familiar than JSON to my partners. The example chosen is a register of customers. It basis is a list of type Customer. The class Customer has the typical attributes, like id, name and address. My idea is very simple, I want to develop a server that allows the four basics operations: get customers (GET), add customers (POST), add or update customers (UPDATE) and delete customers (DELETE). Once I have developed the server, I prove it with a plugin of Firefox browser called Poster, as with a browser I only can use GET requests. Later than a few tries, and quite code modifications, I can get all the customers or a specific customer, add a customer, or update/delete it. The server can be started easily from Eclipse. Following, I start to develop a simple client in Jersey. I want to use it instead of Poster, although in the code demonstration I will use the cliente in Jersey and Poster. I developed the client to use the four basic operations, and the console shows what is happening every moment.

B. Implementation with RESTEasy

RESTEasy is a JBOSS project, so I start to search documentation about JBOSS. After a couple of hours, I still have not idea about how use RESTREasy in JBOSS and I decide to make it work with Eclipse and Tomcat. The configuration is very similar, but there are some differencies. Of course, I need the JARS of RESTEasy instead of Jersey. The biggest difference, is in the file web.xml, wich is quite different. Once I have integrated RESTEasy, I developed easily a simple server. In this case, the example is very similar, a list of Products. I notice that I can develope a client also in RESTEasy, and I do it. Here I find some problems, because the way to write a client in RESTEasy is different that in Jersey. Of course, Poster can be used also with this example.

VI. Conclusions

I have tried to make clear the concept of SOA and the easiest technologies to implement SOA. In the first section of the paper, I explain what is SOA, the keys of SOA, why people should use it and I make a short introduction to web services. In the second section, I talk about the usage of SOA in mobile services. Then, the two most poupular technologies to implement SOA,SOAP and REST, are introduced. The tools used in the code demostration are explained in section four. The last section shows how I have implemented the code demostration. This section is like a diary of the implementation of the code, the problems that I had, and how I resolved them. The reason of the simplicity of the examples, is that they can be easily understandable for my partners, and a second reason is the short time that we have for the code demostration.

References

[1] Alon Mizrahi, Evolution of SAP SOA: http://wiki.sdn.sap.com/wiki/display/bpxproj/Evolution+of+SAP+SOA, November 2011, Last access: June 2012.

[2] David Sprott, Lawrence Wilkes, Understanding Service-Oriented Architecture: http://msdn.microsoft.com/en-us/library/aa480021.aspx, January 2004, Last access: June 2012.

[3] Hugo Haas, Allen Brown, Web Services Glossary: http://www.w3.org/TR/ws-gloss/, February 2004, Last access: June 2012.

[4] Raghu R. Kodali, What is service-oriented architecture?: http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html, June 2005, Last access: June 2012.

[5] OmarAl Zabir, Developing Next Generation Smart Clients using .NET 2.0 working with Existing .NET 1.1 SOA-based XML Web Services: http://www.codeproject.com/Articles/11163/Developing-Next-Generation-Smart-Clients-using-NET, August 2005, Last access: June 2012.

[6] Qusay H. Mahmoud, Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI), http://www.oracle.com/technetwork/articles/javase/soa-142870.html, April 2005, Last access: June 2012.
[7] SOA Requirements: http://www.j2eebrain.com/java-J2ee-soa-requirements.html, Last access: June 2012.
[8] Patrick Cooney, Understanding Web services: http://www.alistapart.com/articles/webservices/, January 2002, Last access: June 2012.
[9] Amazon Web Services: http://aws.amazon.com/, Last access: June 2012.
[10] Do van Thann, Ivar Jorstad, A Service-Oriented Architecture Framework for Mobile Services, 2005.
[11] Towards SOA-based Mobile Solutions: http://blog.akquinet.de/2009/08/28/towards-soa-based-mobile-solutions/, August 2009, Last access: June 2012.
[12] Tarandeep Singh, REST vs. SOAP – The Right WebService: http://geeknizer.com/rest-vs-soap-using-http-choosing-the-right-webservice-protocol/, August 2009, Last Access: 2012.
[13] SOAP Tutorial: http://www.w3schools.com/soap/default.asp, Last access: June 2012.
[14] Alex Rodríguez, RESTful Web Services: The basics, http://www.ibm.com/developerworks/webservices/library/ws-restful/, 2008, Last access: June 2012.
[15] When to use REST based web services?: http://oracled.wordpress.com/2009/03/11/when-to-use-rest-based-web-services/, March 2011.
[16] Introducing JSON, http://www.json.org/, Last access: June 2012.
[17] The JBoss Way, http://www.jboss.org/, Last access: June 2012.
[18] Building RESTful Web Services with JAX-RS, http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html, Last access: June 2012.
[19] RESTEasy - JBoss Community: http://www.jboss.org/resteasy, Last access: June 2012.
[20] RESTEasy Tutorial: http://www.mastertheboss.com/web-interfaces/273-resteasy-tutorial-.html, Last access: June 2012.
[21] RESTful Web Services Developer's Guide: http://docs.oracle.com/cd/E19776-01/820-4867/ggnyk/index.html, Last access: June 2012.
[22] Lars Vogel, REST with Java (JAX-RS) using Jersey - Tutorial: http://www.vogella.com/articles/REST/article.html, June 2012, Last access: June 2012.
[23] JSR 311: JAX-RS: The JavaTM API for RESTful Web Services: http://jcp.org/en/jsr/detail?id=311, Last Access: June 2012.