Server to Mobile Device Communication: A Case Study

Ricardo Anacleto[1], Lino Figueiredo[1], Ana Almeida[1], and Paulo Novais[2]

[1] GECAD, Knowledge Engineering and Decision Support Research Center,
School of Engineering of the Polytechnic Institute of Porto, Porto, Portugal
{rmsao,lbf,amn}@isep.ipp.pt
[2] CCTC - Computer Science and Technology Center,
University of Minho, Braga, Portugal
pjon@di.uminho.pt

Abstract. Develop a client-server application for a mobile environment can bring many challenges because of the mobile devices limitations. So, in this paper is discussed what can be the more reliable way to exchange information between a server and an Android mobile application, since it is important for users to have an application that really works in a responsive way and preferably without any errors. In this discussion two data transfer protocols (Socket and HTTP) and three serialization data formats (XML, JSON and Protocol Buffers) were tested using some metrics to evaluate which is the most practical and fast to use.

Keywords: Client-Server Communication, Mobile Applications, Protocol Buffers, Performance.

1 Introduction

Nowadays mobile devices still have several limitations (network traffic and battery consumption) compared to traditional computers that must be considered when developing a mobile application. It was based on these limitations that led us to the question: Which is the best way to exchange information between a server and a mobile client in order to minimize these limitations?

This question started to appear when developing a mobile application PSiS (Personalized Sightseeing Planning System) Mobile [1] to support a tourist when he is on vacations - more information about PSiS Mobile can be seen in section 2.

To answer this question a case study was performed, where the data transfer protocols performance were tested. It was done by transferring the points of interest data between the two sides (PSiS server and mobile application). Each point of interest is represented by 13 data fields where each one is formatted as string. The field which contains more data is the description, which in some cases can have more than 1000 characters. Each point of interest has about 600 Bytes of data.

Since the mobile application was developed to be used by an Android mobile device, a Google Nexus S with Android 4.1 was used. A normal notebook PC was used as server. Both were connected to the same IEEE 802.11g network. To decide which is the best technique to perform the data exchange, five metrics were used:

A. van Berlo et al. (Eds.): *Ambient Intelligence – Software & Applications*, AISC 219, pp. 79–86.
DOI: 10.1007/978-3-319-00566-9_11

- Process Duration, includes server request, data transfer, deserialization and data record on local database. This is important to realize which is the fastest technique;
- Average CPU load, important to see which system resources are being used;
- Average used Memory, the same as the previous one;
- Total bytes sent, this is very important because of the expensive data costs that carriers charge, less data consumption means less money spent;
- Total bytes received, has the same importance as the previous one.

In section 3 the performed case study is presented. This case study involves the transfer of points of interest from the server's database into the mobile device database using different technologies and based in some metrics to evaluate the results and understand which is the more appropriate to use. Section 4 presents an analysis and discussion about the obtained results. Finally, in section 5 some conclusions about the case study results are presented.

2 Case Study Context

The necessity to discover which is the best transfer protocol and data serialization format to transfer information between a server and a mobile application came when the authors were developing PSiS Mobile. This mobile application appears on the context of PSiS, which is a web application that aims to define and adapt a visit plan combining, in a tour, the most adequate tourism products (interesting places to visit, attractions, restaurants and accommodations) according to the tourists specific profile (which includes interests, personal values, wishes, constraints and disabilities) and available transportation system between different locations [1].

PSiS Mobile is composed by three pieces (see figure 1), the server-side, the middleware and the mobile client. In the server exists have a complete database with all the information about points of interest in a certain city/region and a complete users portfolio. The middleware was implemented to enable the communication between the server side and the mobile application.

The mobile client is a very important part of this system, because it is the bridge between the central services and the user visits. With a mobile device, the user can see the generated planning and the information about the nearby sights to visit, which are recommended according to his profile and current context. Also, the trip planning can be re-arranged according to the current context.

Since PSiS Mobile is an occasionally connected application, a temporary database is used on the mobile device to enable the access to part of the data without being constantly consuming network traffic, allowing the application to work without an internet connection (with some limitations, like no access to new points of interest).

After requesting a recommendation for a trip, all the necessary data is transferred from the server and stored on the mobile device. This was found to be necessary, because of the mobile Internet low speed rates and the possible unavailability. This necessary data represents the information about all the points of interest present on the planning schedule and other points of interest nearby the first ones.

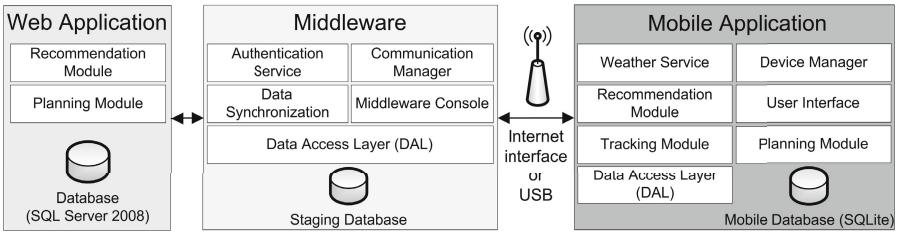

Fig. 1. PSiS Architecture Overview

3 Inter-process Communication Flow

There are several ways to exchange information between a server and a client, but in this case only two of the most used protocols were chosen to test, the Java Socket API [5] and the HTTP (Hypertext Transfer Protocol) REST (REpresentational State Transfer Web Services) [3]. The SOAP Web Services were left behind because of the bigger headers compared to the REST architecture, which increases the amount of network traffic and process power [7].

After data transfer protocols selection, the structure to serialize the information was defined. This is important in order to the two parties (server and client) "understand" each other, in this case it was chosen the XML, JSON and Protocol Buffers data structure formats.

Raw socket was the first tested approach since normally they are used to quickly exchange information [6]. First of all, a raw socket client and server modules were implemented. For each established connection, the server creates two threads: one to send data and another to receive data. Since there are two different threads the exchange can be performed asynchronously, avoiding waiting states on the client application. To test this protocol the data was serialized by the SAX Parser using a XML structure. With this protocol, message sizes were more compact since there aren't any headers (*e.g.*, HTTP or SOAP headers).

However, this system poses several problems in sockets management. Besides the need to specify a hard-coded and very inflexible communication protocol, raw sockets also need further implementation for error detection and transaction control.

The other tested protocol was HTTP, which is one of today's most popular client-server communication protocols. HTTP is a mature approach and a widely used protocol that already handle errors, simplifying its use and implementation. The only downside, comparing to the raw socket communication protocol, is the size of the sent/received data frames. This mainly happens because of the HTTP header, which is added to the sent/received data.

The header size along with the sent and received ACK (Acknowledgement) packages, to validate the transaction, varies between 6% and 10% of the size of the transferred data. For example, for a XML file with a size of 1.875 Mb, the client receives a total of 2.048 Mb (9% more than the original file size).

After the protocols chosen, three data structure formats were selected to test. The first one is the XML, since it is one of the most popular data structure formats used to store

information. To have a better understanding about the XML performance three different XML parsers were used: DOM (Document Object Model), SAX (Simple API for XML) and Pull. DOM was chosen since it is the World Wide Web Consortium (W3C) standard and the other two because they claim to be the fastest XML files parsers.

Second one is JSON (JavaScript Object Notation) [2], which has a structure identical to the XML, but tries to be a low-overhead format. Finally there is Protocol Buffers [4], which is a serialization format developed by Google Inc. with the purpose to be simpler and faster than XML.

4 Empirical Analysis

In this section the results for each of the previously described exchange data techniques will be presented. To ensure more accurate results, four different tests with different file sizes were performed. Each of these tests was executed five times, and the presented results are the average of the five attempts. The file sizes, for each test and data serialization format are described on table 1.

Table 1. File sizes (in kB) for each test and data serialization format

Serialization format	First	Second	Third	Fourth
XML	1	253	375	1875
JSON	0.779	227	313	1564
ProtocolBuffers	0.665	195	256	1276

In the first test only the information of one point of interest was used. This was valuable to get a first look of the mobile devices behavior when few data bytes are exchanged over network compared to big files.

Analyzing table 2, it appears that the fastest architecture is HTTP using Protocol Buffers, followed by HTTP using JSON. The raw socket protocol was slower mainly because of the connection initialization, which is a time consuming process, especially when we try to detect and control communication errors.

However, as expected, it was the raw socket with XML architecture that had fewer bytes transferred between server and client followed by the HTTP protocol with Protocol Buffers. Finally, HTTP with XML is the heaviest of them all.

To this test weren't provided any data for the CPU load and memory metrics because the process is completed so quickly that significant values can't be obtained (the readings are made per second).

In the second test the information about 250 points of interest was transferred. One of the most relevant findings is that the XML parsing algorithms have significant performance differences. The DOM, one more time, was the slowest and SAX proved to be the fastest, surpassing Protocol Buffers that only in this test wasn't the best.

Table 2. First test results

Protocol	Duration (ms)	CPU (%)	Memory (MB)	Data Received (kB)	Data Sent (kB)
HTTP XML SAX	595	-	-	1.5	0.5
HTTP XML DOM	773	-	-	1.5	0.5
HTTP XML Pulll	555	-	-	1.5	0.5
HTTP JSON	511	-	-	1.2	0.5
HTTP PROBUF	**506**	-	-	1.1	0.5
SOCKET	1893	-	-	**1.0**	**0.5**

Table 3. Second test results

Protocol	Duration (ms)	CPU (%)	Memory (MB)	Data Received (kB)	Data Sent (kB)
HTTP XML SAX	**2023**	46.5	4.59	270.0	5.8
HTTP XML DOM	14947	90.1	6.02	270.2	5.5
HTTP XML Pulll	4940	76.4	5.43	270.0	7.1
HTTP JSON	3784	78.9	5.26	241.2	6.4
HTTP PROBUF	2036	55.8	5.03	**206.9**	5.3
SOCKET	7485	**22.7**	**4.27**	262.9	**4.7**

Looking at table 3, can be seen that socket method consumes less system resources (CPU and memory) than the others because it doesn't have so many parsing routines. However, the whole process still takes a long time to execute. Protocol Buffers was the one that had transferred less bytes, since it includes some data compression.

In the third test, it was transferred the information about 461 points of interest. The results follow the same pattern of the previous tests, where Protocol Buffers was the fastest, though only for a little margin (table 4).

JSON behaved as expected, serialization turns the file lighter than XML, but it has a weak decoder (the Android platform native JSON parser was used) and becomes slower when compared with the, also Android native, SAX Parser.

Analyzing the CPU utilization data, can be observed that the worst is HTTP with DOM parser, since it uses an average of 93% during 26 seconds, which can represent a lot of battery spent. Another important analysis is that the socket method only has used 48% of CPU but it has an overall duration of almost 8 seconds. Comparing it with HTTP using Protocol Buffers, can be seen that Sockets aren't so good, because HTTP with Protocol Buffers uses 51% but only for 2 seconds. Considering the memory usage, socket method uses less memory than the others protocols.

Finally, the fourth test, where it was decided to perform a more thorough test to denote additional differences on the obtained results. In this test the information about 1884 points of interest (four times all the points of interest stored on the database) was used.

Table 4. Third test results

Protocol	Duration (ms)	CPU (%)	Memory (MB)	Data Received (kB)	Data Sent (kB)
HTTP XML SAX	2797	70.4	5.65	398.9	8.6
HTTP XML DOM	26985	93.3	5.95	399.4	8.5
HTTP XML Pulll	5331	81.6	5.44	398.9	8.4
HTTP JSON	4876	79.7	5.16	332.6	8.6
HTTP PROBUF	**2316**	51.0	5.14	**271.5**	7.4
SOCKET	7949	**48.0**	**4.99**	384.7	**7.0**

Table 5. Fourth test results

Protocol	Duration (ms)	CPU (%)	Memory (MB)	Data Received (kB)	Data Sent (kB)
HTTP XML SAX	12171	**70.0**	6.59	2048	37.3
HTTP PROBUF	**10060**	72.3	**5.89**	**1400**	**28.5**

Comparing the third with the fourth test, can be observed that the processing time has been 5 times more and the amount of data transferred is only 4 times the transferred data on the third test. This is mainly explained because of the limited mobile device memory. The operating system is always trying to get more and more memory and it slows down the entire process.

Notice that only results for two techniques are provided. This happened because all the others gave an "Out of Memory" error due to the mobile device lack of memory. This happens because Android heap memory is limited to 16MB per application on the most available devices, and only the high-end ones have a limit of 24MB. These two techniques were also the ones that have produced better results in the other tests (HTTP with SAX Parser and HTTP with Protocol Buffers). As can be seen on table 5 both used almost the same system resources.

In this test can be seen a bigger difference in performance between Protocol Buffers and SAX, especially in the transferred data size. Protocol Buffers transmitted about 600 kB less data (since the serialized file is that much smaller) and in lesser two seconds than the SAX parser.

5 Conclusions

The purpose of this study was to discover which technology/technique is more reliable and faster to use in a server-Android mobile application environment. Therefore, in this chapter the conclusions about the obtained results and what technique was chosen to use are presented. Also, some considerations that have been learned and validated during these tests are discussed.

In theory, socket approach seems to be the right choice. In practice, it was found some important disadvantages compared to the other approaches, since it proved to be error prone and slower. Considering the analysis of cost over benefit between this approach

and HTTP, it was concluded that the socket gains on the transferred kBs between the two sides, don't outweigh the associated disadvantages. The socket results can be explained by a poor optimization of the Android Socket API.

Sockets were left behind due to the few advantages that they actually bring, compared to the HTTP protocol. Also, raw sockets are much more complex and hard to work with. It's like reinventing the wheel when it already exists. On the other hand, HTTP is reliable and is able to perform error handling. HTTP was the chosen protocol for the PSiS Mobile implementation. With these tests the research team attested, that the time spent in the implementation of sockets is not worth the supposed superiority of performance, which in this case there wasn't any of it besides the smaller data messages.

After choose the transfer protocol the most commonly used data serialization formats to encapsulate the data to be sent over that protocol were inspected. Starting with XML, the case study revealed that after all it isn't so slow to parse, but instead it depends highly on the used parser. Regarding file size it is only slightly behind the others, because of the inclusion of multiple tags and for no data compression implementation. Another issue that has to be considered is to not rely only in the theory, but try to understand it and put it into practice in order to confirm the results for our case.

Considering the XML parsers, it is noteworthy that DOM is definitely the slowest and the most complex to work. The SAX ends up having a similar performance to Protocol Buffers, which proved to be the lightest and the fastest in almost all the tests. These two are, according to our tests, the best approaches. SAX is overtaken by the Protocol Buffers when it comes to speed and file sizes, thus can be concluded that Protocol Buffers is the fastest and lightest serialization format. Then and as expected, since it is one of its claims, JSON files are smaller. However, the Android native JSON parser proved to be slower than the best XML parser.

According to the previous statements, the HTTP protocol in conjunction with Protocol Buffers was the chosen mechanism to exchange information between PSiS server and mobile application, since it spent less system resources (therefore less battery) and less network data consumption. Thus, some of the limitations of mobile devices were minimized.

Another lesson that was learned is that there is no advantage in sending few or a lot of information at once, but something in between them. If few information is sent at once a great waste of time exists in the initialization of the communication. Comparing the second and third tests, where twice the information was sent, can be seen that it takes just a little more time to process it. However, if a lot of information is sent at once, as done in the fourth test, some memory problems can be experienced and thereby slow down the whole process. The best thing to do is to choose something in the middle, *i.e.*, medium-sized files.

Finally, the research team has learned that it is worth investing some time in these small tests, because with them the user experience can be improved. These tests don't take so long to implement and can result in a good knowledge for the team. Has can be seen, for Android platform the HTTP protocol and Protocol Buffers are well implemented and it is worth to give a try, getting a fast and reliable solution to transfer information between a server and an Android mobile device.

Acknowledgement. The authors would like to acknowledge FCT, FEDER, POCTI, POSI, POCI and POSC for their support to GECAD unit, to the project PSIS (PTDC/TRA/72152/2006) and for the PhD grant (SFRH/BD/70248/2010).

References

1. Anacleto, R., Luz, N., Figueiredo, L.: Personalized sightseeing tours support using mobile devices. In: Forbrig, P., Paternó, F., Mark Pejtersen, A. (eds.) HCIS 2010. IFIP AICT, vol. 332, pp. 301–304. Springer, Heidelberg (2010)
2. Crockford, D.: JSON: the fat-free alternative to XML. In: Proc. of XML, vol. 2006 (2006)
3. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Transactions on Internet Technology (TOIT) 2(2), 115–150 (2002)
4. Google: Protocol buffer (2012), `http://code.google.com/apis/protocolbuffers/docs/overview.html`
5. Harold, R., Loukides, M.: Java network programming. O'Reilly & Associates, Inc., Sebastopol (2000)
6. Pakin, S., Karamcheti, V., Chien, A.A.: Fast messages: Efficient, portable communication for workstation clusters and MPPs. IEEE Concurrency 5(2), 60–72 (1997)
7. Pautasso, C., Zimmermann, O., Leymann, F.: Restful web services vs. big'web services: making the right architectural decision. In: Proceeding of the 17th International Conference on World Wide Web, pp. 805–814 (2008)