International Journal of Computer Applications (0975 8887)
Volume 93 - No. 7, May 2014

Moodle Mobile Notifier

Hassan Sbeity and Ahmad Fadlallah
Arab Open University - Lebanon

ABSTRACT

E-learning management systems (eLMS) are increasingly used as online communication platform between students and teachers in traditional, distance and open learning educational institutions. Moodle [10] is the most used open-source e-LMS platform with around 68 million users being served worldwide [9]. This server-side application can be accessed via a web browser, on any computer or Internet-enabled hand-held device (smart phone, tablet, etc.). However, this system has three limitations: first, the lack of synchronization between client and server applications, in other words between posting (server side) and viewing (client side) the information. The second limitation is that students cannot easily differentiate between newly and previously seen information. This is because most of the Moodle posts are not tagged with dates (except news and messages). The third limitation arises specifically when the Moodle server is accessed from a hand-held device using mobile telecommunication networks where bandwidth is considered as a scarce resource (low data rates and limited download/upload quotas). This limitation is caused by the ”useless” exchange of high amount of data to load all page contents even if no new information has been posted. The above listed limitations have negative impact on the tutor-student communication when eLMS platforms are used.
This article proposes a system based on a three-tier architecture named Mobile Moodle Notifier (MMN) to overcome these limitations: the first tier is a mobile application (built on Google Android operating system [3]) that communicates with Moodle using low-level programming (socket programming) in order to reduce the bandwidth and download data consumption. The second tier is a server-side application that feeds the users (mobile application) with newly posted information from the third tier, which is the - extended - Moodle database. Performance tests show a 50 times faster execution time and 300 times less download data consumption on average using MMN compared to accessing Moodle via a regular web browser. ■

1. INTRODUCTION

Moodle is an open source e-learning management system used in many types of learning environments such as education, training and development, and business settings. In education it is used as a platform providing the interaction and collaboration between students and tutors. This study focuses on its use in higher education institutions (HEIs). Moodle includes varieties of tools (assignment submission, online quiz, grading, forums, file sharing, instant messaging, Wiki, etc.) which facilitate course management and improve communication (tutor-student and student-student) in the learning process. The second aspect (communication) and in particular tutor-student communication is one of the crucial additions of the eLMS in general and Moodle in particular, which allows the exchange of various type of information between tutors and students (course material, announcements, instant messages, assignments, feedbacks, etc.). Given their variety, flexibility and ease-to-use, Moodle tools are the main means for tutor-student communication in many HEIs. An important factor for the success of this communication is the synchronization [1] between posting and viewing the information. In order to ensure this, students are asked to visit the Moodle website ”regularly” to check for any update that may have been posted by their tutors (or their peers). The term ”regularly” in this context needs to be clearly defined in order to determine a minimum frequency of accessing the eLMS, which allows the student to read any new post on time [12].
The widespread use of hand-held devices (smart phones, tablets, etc.) in addition to the large deployment of wireless and mobile technologies providing internet access, are seen as as a promising solution to increase the frequency of students access to the eLMS (thus helping to solve the synchronization issue); students can access the Moodle website from their hand-held devices ”anywhere and everywhere”. Furthermore, last years have witnessed the development of several mobile applications [7] [11] [1] [6] that provide Moodle ”mobile” users with many features of the Moodle website with a user-friendly interface. Although this ubiquity of access provided by mobile technologies increases the Moodle access frequency of students, these applications still do not completely solve the synchronization issue (will be further discussed in the related work section). Moreover, the mobile internet data consumption (bandwidth and download/upload quota) limits the use of these applications (and the access to Moodle from mobile devices) in the developing countries where mobile internet resources are considered as scare (limited and expensive). A survey[2] that has been conducted at the Arab Open University - Lebanon branch, shows that more than 90% of students have smartphones, more than 60% of them have internet access from their Mobile devices; however the access logs collected from the Moodle server[3] shows that less than 3% of accesses are from Mobile devices.
Finally, and based on students’ feedback (same survey), one reason for missing certain information posted on Moodle is the inability of students to differentiate between newly and previously posted

[1] the term synchronization is used in the context of reducing to the minimum the time interval between two events
[2] The survey was conducted in April 2013 at the Information Technology and Computing department
[3] for the time period of two months between February and March 2013

information (since their last access) given the amount and variety of such information.
The main objective of this work is to solve the synchronization problem in relation to its different aspects as mentioned above. The proposed solution has a multi-tier architecture consisting of:

(1) Mobile application: notifies students about posted information with the possibility to differentiate between newly and previously posted information, and with a minimum internet data consumption.
(2) Server application: acts as mediator between the Mobile application and the database.
(3) Database: Moodle database has been extended to support the new features provided by the application.

The discussion is organized as follows: Section 2 presents the related work. Section 3 describes the general multi-tier architecture of the proposed solution. Sections 4, 5 and 6 describe respectively the data, the server and the client tiers of the architecture. Section 7 illustrates and discusses the performance of MMN. Finally, section 8 concludes the paper.

2. RELATED WORK

As outlined in section 1, users can access Moodle from their handheld devices in two ways:

(1) Opening Moodle website from their mobile web browser.
(2) Using native applications for their Mobile devices.

In order to compare this work with similar works, the most known mobile applications listed on official Moodle website [8] have been selected.

(1) Official Moodle Mobile Application [7]
(2) Unofficial Moodle Mobile Application [6]
(3) Moodle Touch (mTouch) [11]
(4) iActive Mobile Application [1]

Table 1 summarizes the supported operating system for each of the listed applications.

Table 1. : Moodle Mobile Applications

Application Name	**Supported Mobile OS**
Official Moodle Mobile App	iOS (iPhone, iPad)
Unofficial Moodle Mobile App	iOS, BlackBerry, Android
Moodle Touch (mTouch)	iOS
iActive	iOS

Although these applications have the same ”ubiquity access” feature of the proposed application, they differ in terms of their goals and objectives and thus do not address the issues previously identified. They aim to provide users with the main Moodle web application features through a user friendly interface that better suits handheld devices (smaller screen, lower resolution, etc.).
A closer look at the features of each of these applications confirms that none of these applications is capable of solving the notification/synchronization issue. Furthermore, concerning the internet resource consumption, these apps rely - partially or completely - on an embedded web browser to access the information. None of them considers the optimization of internet resources consumption as a matter to be addressed. Further analysis and comparison of this aspect will be provided in section 7.

3. MMN ARCHITECTURE

As already mentioned, MMN is based on a three tier architecture:

—Mobile Application or presentation tier
—Middle-tier application (server or logic tier) that acts as a mediator between the presentation tier and the data tier. [4]
—Data tier which consists on the Moodle database that was extended to support the new features provided by the middle-tier application.

Figure 1 illustrates the MMN data flow:

(1) The client tier initiates a request to the middle tier asking for any updates in the extended Moodle database.
(2) The middle tier issues a query to the extended Moodle database based on the client request.
(3) The middle tier forwards the result of the query to the client tier.

It is important to mention that the extended Moodle database keeps track of any updates (posting new grades, new announcements,..) in the original Moodle database.

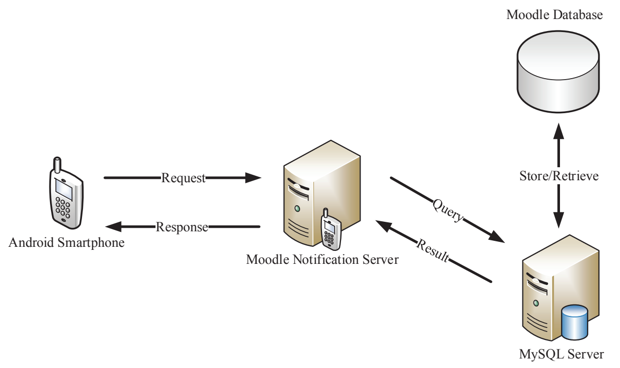


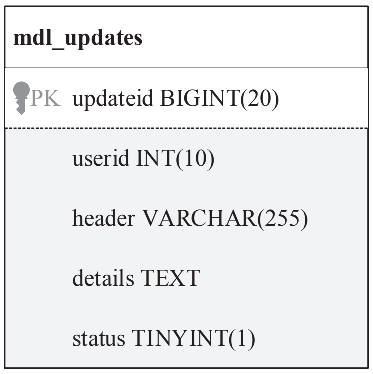
Fig. 1: MMN architecture

4. DATA TIER

The Moodle database has been extended by adding a single table ”mdl_updates” (referred to as ”mobile notification table”) (see figure 2). This table contains all the information needed to answer any request issued from the client tier through the middle tier to the data tier (through the MySQL server).
The mobile notification table is updated, regardless of the client requests using multiple triggers which are executed whenever the Moodle database is updated. Table 2 shows the inserted triggers along with the corresponding Moodle tables. The outcomes of the execution of these triggers update the mobile notification table.
Two categories of Moodle updates can be distinguished:

(1) Group Updates targeting a group of students who are enrolled in a certain class (such as posts on forum, assignment, announcements, etc.)

[4]It is important to mention that this layer includes also the MySQL Database server and the Moodle Web server applications which are part of the Moodle architecture



mdl_updates

PK updateid BIGINT(20)

userid INT(10)

header VARCHAR(255)

details TEXT

status TINYINT(1)

Fig. 2: Mobile Notification Table

Table 2. : Triggers List

Trigger	Database Table	Target
assignment	mdl_assignment	newly added assignments
chat	mdl_chat	newly added chat rooms
forum	mdl_forum	newly added forums info
forum	mdf_forum	newly started discussions
grade	mdl_grade_grades	newly added grades
message	mdl_message	newly added messages
news	mdl_course_sections	announcements and weeks summary
page	mdl_page	newly added pages
post	mdl_forum_posts	newly added posts.
resource	mdl_recourse	newly added recourses
url	mdl_url	newly added URLs

(2) Simple updates targeting single students (such as messages or grades).

Group updates cannot be done through the only use of triggers since the mobile notification table needs to be updated for every student within the group. Triggers are complemented in this case with stored procedures; once a trigger (for a group update) is executed (based on Moodle database updates), the corresponding procedure is called in order to create a record for each student in the mobile notification table.

5. SERVER TIER

The server (or middle-tier) is composed of two separate applications: a background service and a Graphical User Interface (GUI) based application[5]. The GUI main role is to (1) start/stop the service and to (2) determine the communication mode with the client tier (whether text or serialized object). It can also be used to (3) specify the port that the server is listening on.
The server application acts as a mediator between the client and the data tiers. It communicates with the client through a specific protocol, while it communicates with the database tier through SQL statements.
The communication between the server and the client is implemented using two different modes: the first mode follows a pre-defined protocol based on the exchange of text messages in a specified format, and the second mode is based on the exchange of serialized JAVA objects. Both approaches uses sockets (API) to exchange the data between the client and the server. The details of the different communication modes will be explained in section 6.

Figure 3 illustrates the activity diagram of the server application.

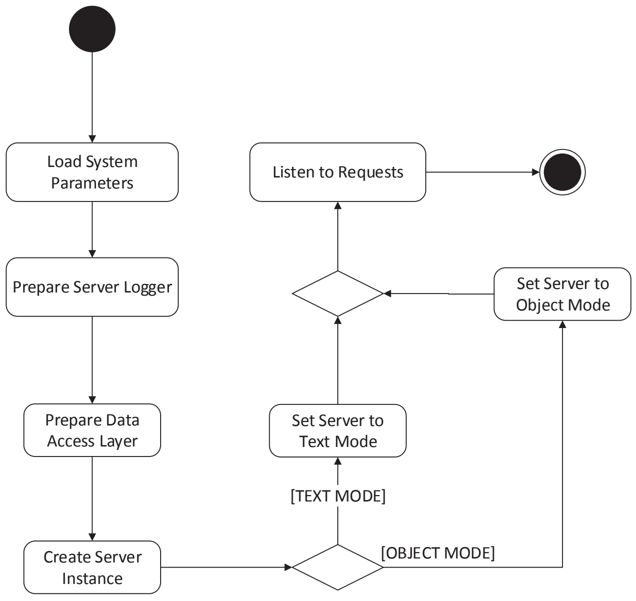


Fig. 3: Server Application - Activity Diagram

6. CLIENT TIER

The general behavior of the client application is illustrated in figure 4. This application supports - like the server application - two modes of communications: text-based and serialized objects-based modes.

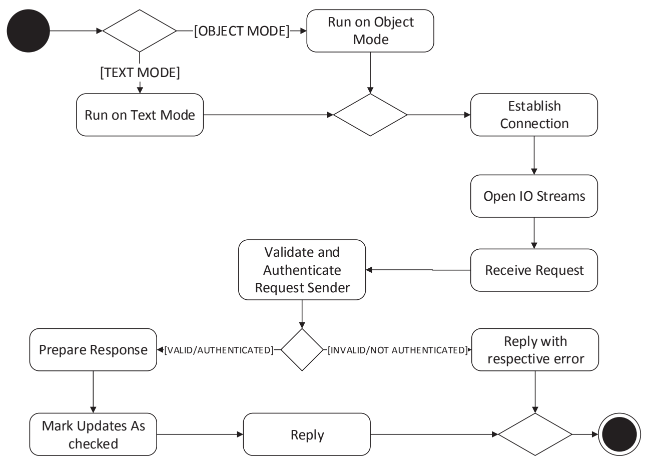


Fig. 4: Client Handler Activity Diagram

[5]A command line interface has been developed in case that the server Operating System (OS) doesn't support GUI.

6.1 Communication modes

6.1.1 Text-based communication mode. As mentioned above, the client exchanges text messages with the server using sockets. The message follows a certain format; it is composed of three words separated by space. It starts with a command word (such as WHATSUP") followed by two arguments.
Example: WHATSUP wao002 liflife, where WHATSUP is the command, wao002 is the username, and lifelife is the password. The server working in text mode will respond to this request and provide to the user the requested data after authenticating the request sender. The response data is a tab-limited formatted string.

6.1.2 Serialized object-based communication mode. Sockets in this mode are also used to exchange serialized JAVA objects with the server. The process starts when the client sends a Request Object that contains the command, the username, and the password to the server (working in object mode). If the server recognizes and authenticates the request, it replies to the client by sending a serialized Update object that contains two lists: one for update headers, and the other for update bodies.

6.2 Client-tier specifications

The client application consists of four classes that extend the Android activity classes: *Startup*, *Configure*, *UpdatesLoader*, and *UpdatesViewer*. The text mode implementation contains the same packages and classes like the serialized objects mode except the one method (getUpdates) in the UpdatesLoader class which is responsible of the communication with the server. This section presents the activity diagrams that illustrate the internal behavior of the client application.

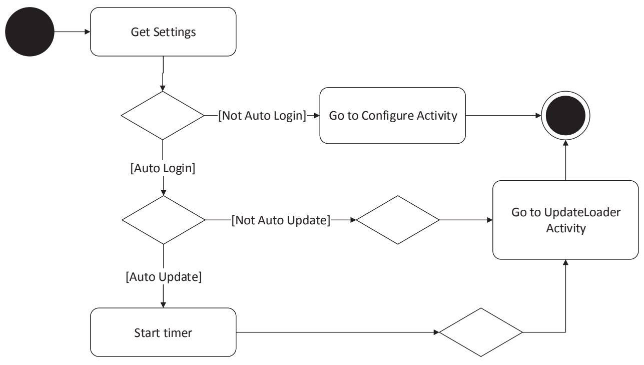


Fig. 5: Activity Diagram for Startup Activity

Figure 5 depicts the *Startup* activity, which is the starting point of any Android application and from which it gets redirected to *UpdatesLoader* activity as per the users recent configuration. If auto-update option is set, it starts the timer before redirecting, otherwise it moves to *Configure* activity.
Figure 6 illustrates the *Configure* activity used to set/modify user preferences: Moodle website URL, username, password, and auto-check option then redirects the user to *UpdatesLoader* activity.
Figure 7 presents the *UpdatesLoader* activity, which is the core of the client application. It contains the *getUpdates* method that communicates with the server, gets the response and saves the received updates if no error occurrs.

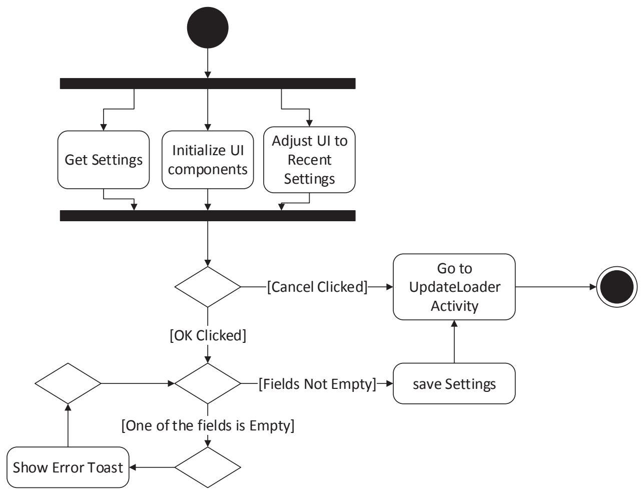


Fig. 6: Activity Diagram for Configure Activity

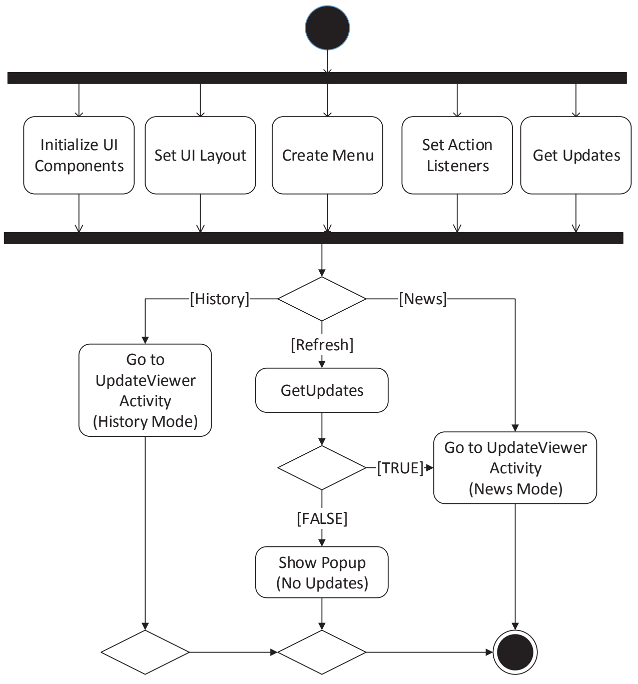


Fig. 7: Activity Diagram for UpdateLoader Activity

Figure 8 depicts the UpdatesViewer activity used to display the saved updates. Two view modes are supported: news mode and history mode.
If this method is called in news mode, it shows only unread updates (fields in table *mdl_updates* (figure 2) with status 0. When the user reads a certain update, the UpdatesViewer changes its status to 1 in the database and adjusts the ListView after removing the viewed item. In history mode, the user can view updates with status 1.

As mentioned in section 6.2, the "getUpdates" method of the UpdatesLoader is implemented differently depending on the communication mode. Figure 9 presents the activity diagram of the text-based mode implementation of *getUpdates*. The process starts with

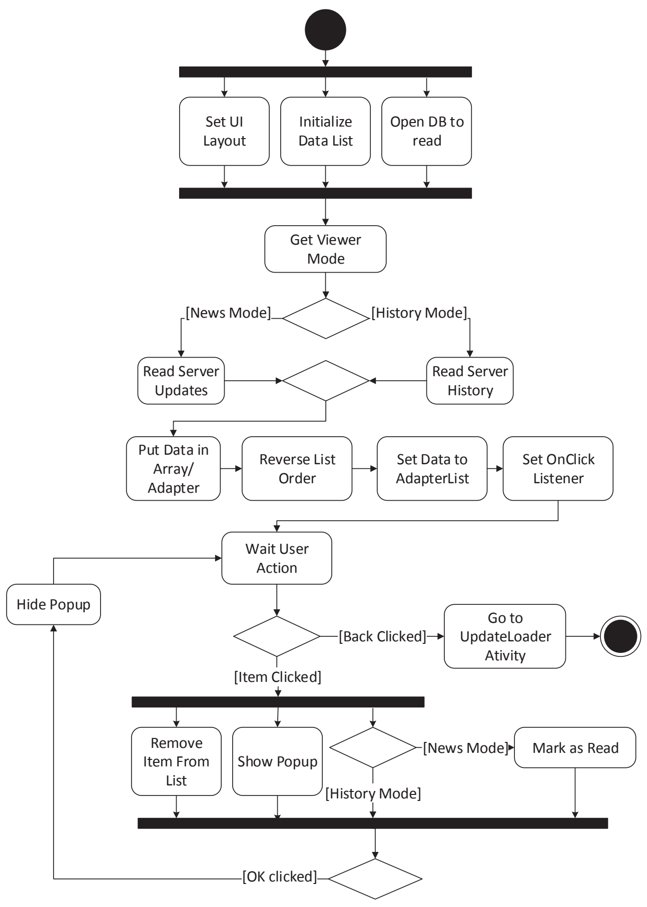


Fig. 8: Activity Diagram for UpdateViewer Activity

establishing connection with the server, it opens IO streams over sockets, sends request, receives update, parses the received data and saves them to database. If anything goes wrong, it informs the user with the appropriate error message.

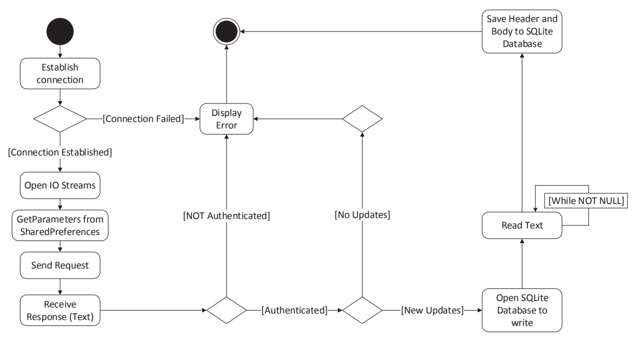


Fig. 9: getUpdates in text-based communication mode

Figure 10 illustrates the activity diagram of *getUpdates* in serialized object based communication mode. The difference is that the IO streams in this diagram are object streams (and not text-based as in the text-based communication mode). The client communicates with the server by sending and receiving JAVA serialized objects.

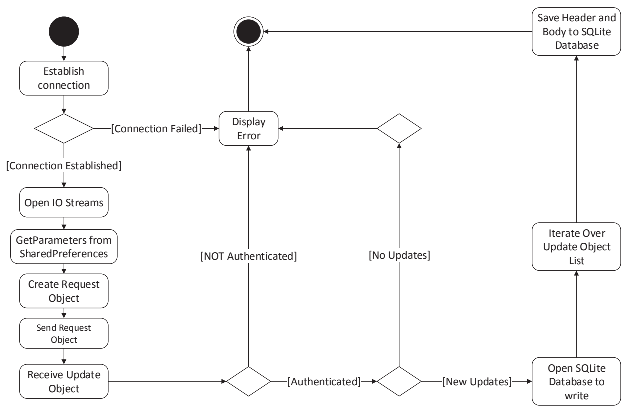
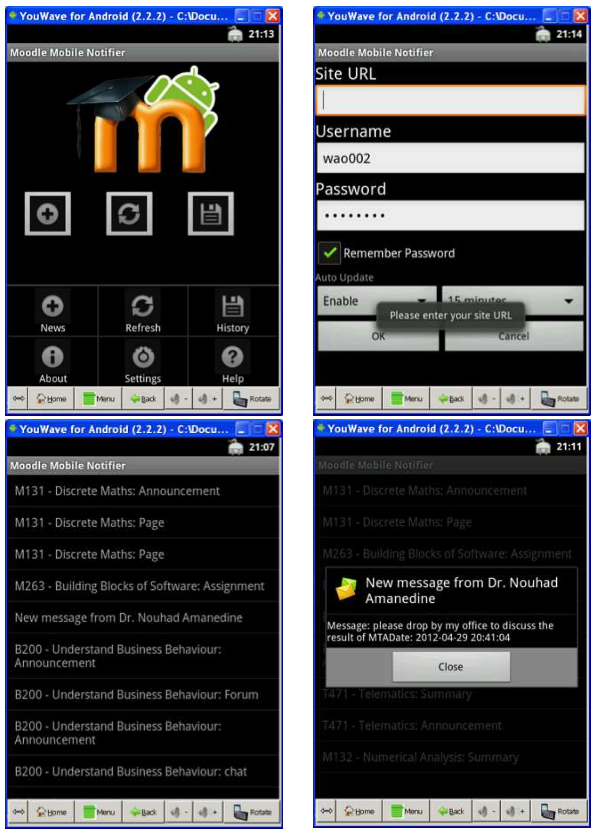


Fig. 10: getUpdates in serialized object based communication mode

Figure 11 shows some screenshots of the client graphical user interface. Simplicity, ease-of-use and user-friendliness were the main guidelines for the design of this GUI.



Site URL

Username
wao002

Password
••••••••

✓ Remember Password

Auto Update
Enable
15 minutes
Please enter your site URL
OK
Cancel

M131 - Discrete Maths: Announcement
M131 - Discrete Maths: Page
M131 - Discrete Maths: Page
M263 - Building Blocks of Software: Assignment
New message from Dr. Nouhad Amanedine
B200 - Understand Business Behaviour: Announcement
B200 - Understand Business Behaviour: Forum
B200 - Understand Business Behaviour: Announcement
B200 - Understand Business Behaviour: chat

New message from Dr. Nouhad Amanedine
Message: please drop by my office to discuss the result of MTADate: 2012-04-29 20:41:04
Close

T471 - Telematics: Summary
T471 - Telematics: Announcement
M132 - Numerical Analysis: Summary

Fig. 11: Client Application GUI

7. PERFORMANCE ANALYSIS

In section 1, two main limitations were identified: synchronization and internet resource consumption. The first limitation is solved;

By design of the proposed solution, student is capable of differentiating between old and new posts through the use of different colors.
To evaluate the impact of this work on the internet resource consumption (second limitation) and hence the power consumption of the embedded system, the following experimental test procedure has been performed.
Table 3 presents the experimental setup configuration for both software and hardware specifications of the hosts where the client and server applications were installed.
In addition, Colasoft Capsa 7 (Free edition - bundled with WinPcap) [2] was used to monitor the network traffic.
In order to evaluate the performance of this work, it has been compared against the HTTP protocol (used when Moodle is accessed through regular or embedded web browser). In this work, two different data packaging methods, text and serialized object, are applied suing the Socket API. Hence, three different data exchange methods are compared as follows:

(1) Classical web interface using the HTTP protocol (Web browser) called HTTP mode.
(2) Serialized object based communication mode called serialized object mode.
(3) Text-based communication mode called text mode.

When using HTTP mode, the browsing history of the browser is cleared before starting each test in order to ensure that there are no data used from cache. It is not possible to send and receive exactly the required data in a single request (contrary to the proposed solution), because through using the web version, one has to go through many page requests, starting from requesting the main Moodle page, then log-in, and browsing the course page. Moodle regular website consumes 671.95 kilobytes as the total download requesting all pages takes 11880 milliseconds data parsing time. This is depicted in the first row of the table 4
When using the proposed solution, the benchmarking was done on Android Virtual Device (AVD) [5] using LogCat tracer [4] integrated in eclipse, sending the same amount of data in both case for object serialized and text mode.
1500 requests were sent from client to server to test both methods (text and serialized object) of this solution. The second row of table 4 shows that the average download of the data consumption is 2.89 kilobytes and the data parsing time is 256.2 milliseconds on a client-server working using serialized objects mode. The third row of table 4 depicts that the average download of the data consumption is 2.20 kilobytes and the data parsing is time 215.8 milliseconds using text mode.
Figure 12 depicts the data consumption and data parsing time of the three compared methods (HTTP, serialized object and text mode). It can be clearly seen that both text and serialized object methods of the proposed solution are performing better than the HTTP mode. The results can be summarized as follow:

—serialized objects mode consumes 232 times less bandwidth compared to HTTP mode.
—text mode consumes 304 times less bandwidth compared to HTTP mode.
—serialized objects mode is 46 times faster compared to HTTP mode.
—text mode is 54 times faster compared to HTTP mode.

It is clear that text and serialized object modes (this work contribution) are outperforming the HTTP mode in both bandwidth consumption and execution time criteria.

Figure 13 depicts a comparison between the text and serialized object for both criteria bandwidth consumptions and execution time. It is clear that the text mode gives the better result.

8. CONCLUSION

This work proposes a Mobile Moodle Notifier system to overcome some Moodle limitations (synchronization problem, lack of date tagging, high download data consumption). The proposed system has a three tier architecture: Android based mobile application (Client tier), Server application (Server tier) and an extended Moodle database (Data tier). The benchmark tests of the proposed system shows that the data exchanged between the mobile phone and the server application is reduced to 300 times and the response time to 50 times in average, while achieving a real time synchronization between posting new information (on the Moodle) and viewing them on mobile phone.

Acknowledgment

The authors thank Vely Orneik for his valuable contribution in the development of the different elements of the MMN architecture.

9. REFERENCES

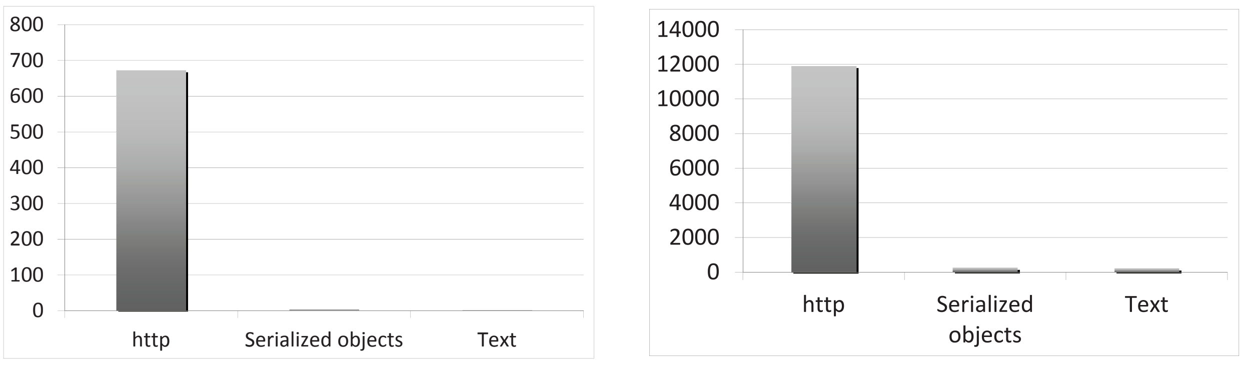
[1] James Chan. iactive mobile application. http://massmedia.hk/n2/mod/resource/view.php?id=36, 2013. Last accessed on April 8, 2014.
[2] Colasoft. Capsa free portable network analyzer(packet sniffer). http://www.colasoft.com/capsa-free/, 2013. Last accessed on April 8, 2014.
[3] Google. Google android website. http://www.android.com/, 2013. Last accessed on April 8, 2014.
[4] Google. Logcat. http://developer.android.com/tools/help/logcat.html, 2013. Last accessed on April 8, 2014.
[5] Google. Managing virtual devices. Android Virtual Device, 2013. Last accessed on April 8, 2014.
[6] Juan Leyva. Unofficial moodle mobile application. http://docs.moodle.org/22/en/umm:_Unofficial_Moodle_-Mobile_app, 2013. Last accessed on April 8, 2014.
[7] Moodle.org. Official moodle mobile application. http://docs.moodle.org/22/en/Mobile_app, July 2012. Last accessed on April 8, 2014.
[8] Moodle.org. Moodle mobile - frequently asked questions. http://docs.moodle.org/22/en/Mobile_Moodle_FAQ, April 2013. Last accessed on April 8, 2014.
[9] Moodle.org. Moodle statistics. https://moodle.org/stats/, April 2013. Last accessed on April 8, 2014.
[10] Moodle.org. Moodle website. http://www.moodle.org, 2013. Last Accessed on April 08, 2014.
[11] Ali OzGur. Moodle touch (mtouch) mobile appl. http://www.pragmasql.com/moodletouch/home.aspx, 2013. Last accessed on April 8, 2014.
[12] Hassan Sbeity. Optimizing the student lms access frequency based on shannon-theorem in developing countries. In *The Cambridge International Conference on Open and Distance Learning 2009*, 2009.

Table 3. : Software/Hardware Specifications

	Client	**Server**
CPU	Core 2 duo 2.00 GHz	Intel Core i3 2.83 GHz
Memory	2GB DDR2	10GB DDR3
Operating System	windows XP professional SP3	Windows seven ultimate 64 bit
Java version	1.7.0	1.7.0
Java Run-Time environment	build 1.7.0-b147	build 1.7.0-b147
Java HotSpot(TM) Client VM	build 21.0-b17, mixed mode, sharing	build 21.0-b17, mixed mode, sharing

Table 4. : Performance Analysis Results

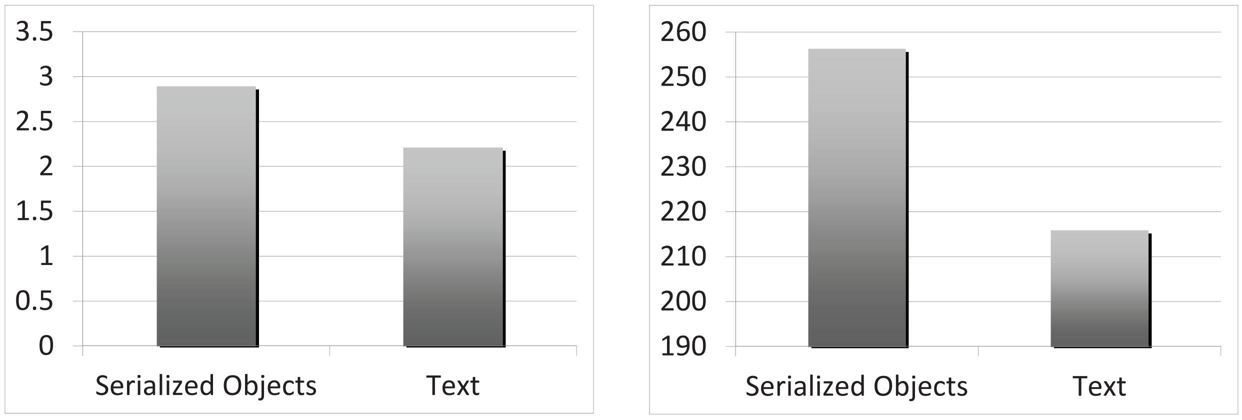
Mode	**Download Data Consumption(Kbytes)**	**Execution time (ms)**
HTTP	671.9	11880
Serialized Objects	2.89	256.2
Text	2.21	215.8



Download Consumption (KB)

Time execution (ms)

Fig. 12: Performance Analysis Results



Download Consumption (KB)

Time execution (ms)

Fig. 13: Performance Analysis Results (Text vs. Serialized Objects)