A Lightweight Distributed Implementation of IMS LD on Google’s Android Platform

Imran A. Zualkernan
American University of Sharjah, UAE
izualkernan@aus.edu

Sina Nikkhah
American University of Sharjah, UAE

Mohammad Al-Sabah
American University of Sharjah, UAE

Abstract

Learning activity based standards like the IMS Learning Design (IMS-LD) have recently emerged. Typical implementations of this standard rely on client-server architectures. This is problematic for resource limited mobile devices that may not have reliable Internet access. This paper presents the design and implementation of a system that takes a sub-set of the IMS LD design specification involving multiple roles and creates native code for mobile devices that collectively run the learning design using SMS messages for synchronization. An implementation targeting the Google’s Android mobile platform is shown.

1. Introduction

IMS-LD is an emerging standard for specifying and running learning designs [1],[2]. In the past there have been some concerns about the semantics of the various tags in this standard [4]. Subsequently, formal semantics for this standard have been proposed [5]. IMS-LD has also been used as the basis for advanced collaborative learning design flow patterns [6].

The IMS-LD standard separates the learning content from learning scenarios that can be subsequently reused. Major building blocks of the standard are represented by roles (who), activities (what), and environments (where). Each role has its own learning objects and services. The standard uses a theater metaphor where a learning scenario is represented by a play composed of various acts. Each role is tied to a series of activities within an act. An act is completed either under a time-limit or when all the roles have completed the activities assigned to them within this act. The next act starts after the current one finishes. The play ends after all the acts have completed.

The IMS-LD standard can be implemented at three levels. Level A provides an implementation for a simple sequence of time-ordered learning activities that are performed by learners and instructors who in turn, use various learning objects and/or services. The Level B implementation, on the other hand, introduces control into the learning sequence where conditions are used to customize the learning flow according to specific circumstances, preferences or learner characteristics. This conditions specifying control are based on either internal or external properties. Internal properties like a learner’s score only persist through a single run of the learning scenario while external properties are retained across various runs. The Level C implementation of IMS-LD adds a messaging component through notifications where new activities can be triggered automatically in response to events in the learning process. This allows for learning designs that are not entirely pre-configured but emerge as a result of participant interactions.

The IMS-LD standard is typically implemented using a client/server architecture where an IMS-LD server is coupled with a front-end client responsible for rendering learning activities to the various roles. For example, Reload player [7] and SLeD [8] are two IMS-LD clients that can be coupled with a JBoss-based CopperCore IMS-LD server [9]. The SMILE LD player [10],[11] is an open-source IMS-LD client allowing delivery of activities and courses through mobile and wireless devices. The SMILE Learning Design PDA player supports a number of features including roles, enrolment of multiple roles/actors, such as individual learners, groups of learners, tutors, asynchronous messaging between different actors/users, and rendering of HTML-based content and Adobe Flash files. This client-server architecture depends on the availability and access to a centralized server. Consequently, this type of architecture may not work well in areas where Internet coverage (through GPRS/EDGE, for example) is not available or unreliable. Further, many interesting mobile learning (m-Learning) designs incorporating the environmental context of learners do not require a centralized server [12].

This paper presents a distributed architecture for rendering learning designs of a sub-set of the IMS-LD specification. This architecture presented in this paper does not rely on client-server architecture. Rather, the learning design is processed once to generate native code for various learners using different mobile devices. After the native code has been downloaded to
the mobile devices, they execute the learning activities in a peer-to-peer fashion.

2. Architecture

Figure 1 shows architecture of the system which has been implemented using the Java programming language and the corresponding toolsets. As Figure 1 shows, the system takes an IMS LD Content Package as input and generates native code for mobile devices being used by various learners playing different roles.

The current implementation of the system generates code for Google’s Android mobile platform [13]. The code transformation is accomplished by using JAXB [14] to create an Abstract Syntax Tree (AST) [15] which is based on an abstract grammar specially developed for this purpose. The Velocity template engine [16] is subsequently used to create native code which is compiled for the Google Android platform.

An important property of this grammar is that it incorporates the semantics of the relevant sub-set of the IMS-LD specification without making a commitment to its current syntactical structure which is specified using XML.

One important aspect of learning design is the synchronization between the various learners. Consider the simple Jigsaw-type learning design shown in Figure 3 where two students read some materials on their own, and subsequently write and send their views to each other. Each student needs to wait for the other to complete before proceeding to propose a solution.

Figure 2. Some productions in the abstract LD grammar

An important property of this grammar is that it incorporates the semantics of the relevant sub-set of the IMS-LD specification without making a commitment to its current syntactical structure which is specified using XML.

One important aspect of learning design is the synchronization between the various learners. Consider the simple Jigsaw-type learning design shown in Figure 3 where two students read some materials on their own, and subsequently write and send their views to each other. Each student needs to wait for the other to complete before proceeding to propose a solution.

Figure 3. Fragment of a learning design requiring synchronization

Figure 4 shows a segment of the AST generated for this specific learning design. Student A has a merge node after the first activity of Readings. The merge node has two reference sub-nodes; one contains a number and the other a reference to another actor. The number represents the synchronization point identifier. The Reference node points to Student B who needs to finish his/her activity prior to this synchronization node and inform other actors (Student A) so that he/she can proceed to the next activity. The synchronization mechanism relies on sending an SMS at every synch point. In the example above, the code for Student B
automatically sends an SMS to Student A with the message body of “synch# 0”. Hence, Student A’s next activity ‘Proposed Solution’ will not play unless it receives an SMS from Student B with the corresponding message body.

In order to deal with simple question types such as multiple choices, check boxes and user inputs results pages, IMS-LD was also extended to incorporate an XML question resource format to handle simple questions. For example, ActivityR02.XML in Figure 4 under the Activity Elements in the Proposed Solution is a representation of a question that asks the learner to enter their proposed solution in a text format. As Figure 4 shows, another synchronization point indicated by reference 1 is also present after Student A proposes a solution.

Another advantage of using the AST as an intermediate representation is that various mobile platforms can be easily targeted using the visitor design pattern [15] for code generation.

3. Detailed Example

The learning design for a detailed example is shown in Figure 5. As the Figure shows, two students are given some material to read. After reading their respective materials, they are asked some questions. After answering their respective questions, each student is asked to choose between two musical instruments. The first student needs to choose between a guitar and a drum while the second needs to select either a flute or a violin. Depending on their choice, each student learns about the chosen instrument. After both students are done learning about their respective instruments, they need to call each other to compare and contrast the instrument they have learned about. Subsequently, each student is asked to reflect on this discussion. After reflection, each student is asked further questions about the various things they have learned in their learning path.
The system generated two Google Android packages from the IMS Content Package corresponding to the learning design shown in Figure 5. One program was generated for Student A, while another was generated for Student B. It should be mentioned that the IMS LD content package, excluding resources was 26 KB while the compiled packages including the processing logic were approximately 15 KB each. These packages were downloaded into emulators as the physical phones running the Android platform were not available at this time.

Figure 6 shows the reading activity being rendered for one student on the emulator. Figure 6 also shows that part of the reading activity consists of questions and that the student is presented with a choice between choosing drums or an electric guitar.

Figure 6. Reading Activity and choice for Student B in ACT1

Figure 7. Student A calling the other after completing their lesson in ACT2

Figure 7 shows Student A calling the other student after both have finished their respective lessons. As Figure 7 shows, Student A is asked to reflect by entering their thoughts in a textbox. This text is sent to the other student in the form of an SMS.

Figure 8. Student B receiving the call and taking the second assessment in ACT 3

After both students have exchanged SMS’es, they are allowed to continue. Figure 8 shows how Student B receives the call from Student A and subsequently answering the second set of questions about the instrument they had chosen in a previous act. Finally, Figure 9 shows a student reviewing the results of their assessments through the various learning activities.

Figure 9. Student reviewing their results in ACT 3

4. Limitations

IMS LD is divided into three levels. Level A primarily deals with roles, activities and environments. Current implementation of the system supports multiple roles and activities but environments including service, monitoring and sending e-mail, etc. are not supported. Services like e-mail sending can easily be added in a future implementation. Similarly, Level B includes properties and conditions. Both of these are supported in the current implementation. All conditions like decisions points and showing and
hiding of the activities are supported. However, only internal properties are currently being handled. Since external properties persist across runs, such an implementation may be based on a peer-to-peer repository. Finally, Level C includes notifications which are currently not supported. However, these can be implemented using a similar mechanism using SMS.

5. Conclusion

This paper has presented a system that takes a subset of an IMS LD specification as input and automatically generates native code for multiple mobile devices where each device corresponds to a role. Learners can interact with each other without the need for a centralized server and use SMS as the primary synchronization and communication medium. This system can be used for a variety of interesting m-learning application requiring interaction among multiple roles. An important feature of the design of the system is the use of an AST based on a generic learning design grammar. The use of AST enables the system to be easily expanded to incorporate additional mobile platforms as targets for code generation.

References


