Data Synchronization Architectural Pattern for Ubiquitous Learning Systems[*]

Samah Gad
Department of Computer Science and Applications
2202 Kraft Drive, Blacksburg, VA 24060
Virginia, USA
samah@vt.edu

ABSTRACT

This paper presents a domain specific architectural pattern for ubiquitous learning systems developers. The pattern provides inexperienced developers with guidelines and the main building blocks to build ubiquitous learning systems. The pattern will enable them to develop reliable systems without having much background in data synchronization and ubiquitous learning systems infrastructures. There are two main data synchronization problems targeted by the presented pattern; maintaing data consistency at anytime and detecting and resolving conflicts in data. A full description and a case study are presented to illustrate the type of targeted ubiquitous learning systems. An implementation and evaluation of the pattern are done based on the introduced case study.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Patterns, Domain Specific Applications, Adaptive Systems, Distributed Architectures*

General Terms

Software, Patterns, Data, Learning

Keywords

Synchronization Pattern, Data Synchronization, Data Sharing, Architectural Patterns, Distributed Database, Distributed Systems, Mobile Learning, Ubiquitous Learning

[*]Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PSI EtA'10, 17-OCT-2010, Reno, USA Copyright 2010 ACM 978-1-4503-0544-0/10/10. $10.00

1. INTRODUCTION

Most users have multiple devices nowadays and some applications need multiple devices to work on. Users have the ability to access their data anywhere, anytime and from any device they are using. All these advances in technology and data sharing abilities lead to a revolution in ubiquitous and mobile learning. Using mobile devices in learning is something that has been done several years ago, but now the attention is going towards adapting learning materials based on data sensed from real life environments.

These type of systems needs to collect sensed data and store it locally on mobile devices used by students. The data should then be synchronized from students to some kind of a server to be shared among other students. All students who uses the system should have a consistent copy from the sensor data on their mobile devices. This is because some learning materials will be displayed based on it. To assure the consistency of data in all the participating devices in the system, a reliable data synchronization algorithm is needed.

Patterns are software engineering designs that have been observed to work well. They can be found in different contexts and provide a solution for a well-defined problem area. Patterns are classified into different groups based on their level of abstraction. This is a definition of patterns that were introduced by Tarkoma [11] pp.103.

Researchers classified patterns into different categories and levels. Examples of pattern categories that were introduced by Tarkoma [11] are; communication, resource management and synchronization, and distribution design patterns. Other researchers like Teale [12] divided the data synchronization patterns into three different levels: architectural, design, and implementation patterns.

There are number of researchers in different areas who studied different mobile applications to come up with patterns for data synchronization. These patterns can help general developers of similar systems in solving data synchronization issues. Roth proposed in [10] a mobility patterns hierarchy. It was claimed that the mobility patterns presented covers mobile applications scenario problems. The focus in the paper was on two design patterns, synchronization and remote proxy patterns. Another research by Teale [12], focused on data movement patterns. Patterns were shown from different abstraction levels, architecture, design, and implementation. Because Teal introduced these patterns while working at Microsoft, the implementation level was focused on implementing the design specifically using Microsoft SQL Server.

Data synchronization is considered one of the key concepts behind most computing systems and networking protocols available nowadays. Data synchronization can appear in many research fields, like distributed systems, mobile database [7], pervasive computing [5], mobile learning [14], data grids [13], and peer-to-peer content distribution [4].

There are two high-level choices when designing a data synchronizer for mobile applications. These choices are; when to synchronize and how to synchronize. There are two ways to know when to synchronize; manually by the user and automatically by the synchronizer. In the presented pattern it will be done automatically.

For how to synchronize, there are two styles of synchronization; pessimistic and optimistic. Pessimistic style means that there are several copies of the data, but only one can to be modified. Synchronization of the modifications is done by copying the modified copy. Modifiability status may also be transferred in the synchronization. The optimistic style means that there are several copies of data each can be independently modified[11]. It is used more than the pessimistic one. One of the major issues in using this style is how to keep track of data updates. There is a number of mechanisms which are used to do this like, version vectors [8] [9], hash histories [6], version stamps [3], interval tree clocks [2], and bounded version vectors [1]. In the presented pattern the optimistic synchronization where used because each mobile device will have a copy of the data stored locally and it can be modified by the user which in this case will be a student.

1.1 Problem Description and Motivation

There are a lot of similarities between sensor based ubiquitous learning systems, because most of them need to collect data and store it locally on mobile devices and synchronize this data to a server. some of the problems attached to this process are unknown by inexperienced developers. The developers end up using some naive solutions to solve these problems and in this kind of learning systems a reliable data synchronization algorithms is needed.

There are few domain specific patterns available in literature. The available patterns are too general to give guidance for developers and designers. The need for a domain specific design pattern will cut the time spent in the software development and design in upiquitous learning systems.

This pattern is targeting two main problems. The first problem is about how to maintaing data consistency between the mobile device and the server at anytime. The second problem is designing a synchronization engine that detects and resolves conflicts in data while transmitting it from the mobile device to the server and vice versa, when the same replication set is possible to be updated at both mobile devices and servers.

The rest of this paper is structured as the following; section two will be about the proposed solution and a full description of the pattern. Section three will present a case study. Section four, is about the implementation and evaluation of the pattern. Finally, section five concludes the paper and give an idea about the future plans and possible extensions for this research.

2. PROPOSED SOLUTION AND PATTERN DESCRIPTION

In this paper, the focus is on an architectural patterns that is independent from any programming language and development tools. The main contribution in this paper is introducing a domain specific architectural pattern for ubiquitous learning systems that adapts learning contents based on collected sensor data. The pattern synchronization engine will be based on optimistic synchronization. It will detect conflicts based on a modified version of Version Stamps [3]. It was modified to fit the types of system targeted here in this paper.

The presented pattern inherits some of the strengths from two Microsoft patterns, Master-Master row level replication pattern and Master-Subordinate Snapshot Replication. It appears that these two patterns inherits the basic building blocks of the data synchronization pattern introduced by Tarkoma [11] pp.103.

Master-Subordinate Snapshot replication pattern features will be used when a new student is joining the class. This decision was taken because this pattern can create a point-in-time copy of defined data. The copy consists of the entire set of data that is to be replicated, not just the changes since the last replication.

Master-Master Row level synchronization pattern features were used, because here the mobile device and server are able to change in replication sets and mobile devices can add data to the replication set while not connected to the server.

2.1 Pattern Elements

The pattern consists of five main elements; sensors, communication layer, mobile device, another communication layer, and server. The mobile device and the server both have a database and a learning application that contains a synchronization engine. One of the communication layers is responsible for the communication, sending data, between the sensors and the mobile device. The other one is responsible for the communication between the mobile device and the server, see Figure 1.

Each one of the synchronization engines in both the mobile device and server have a specific role for each side at the beginning. The server database contains the replication set that is to be transmitted. The mobile device has the database where the transmitted replication set to be written. Both server and mobile device have databases that contains a replication set. The schema of the server database and the mobile device database are the same in both the mobile device and the server. The database in both has only one table that has all the sensor data, which means that there is no referential integrity enforcement needed by the database, and there are no follow-up operations, such as triggers or cascade deletes during the replication.

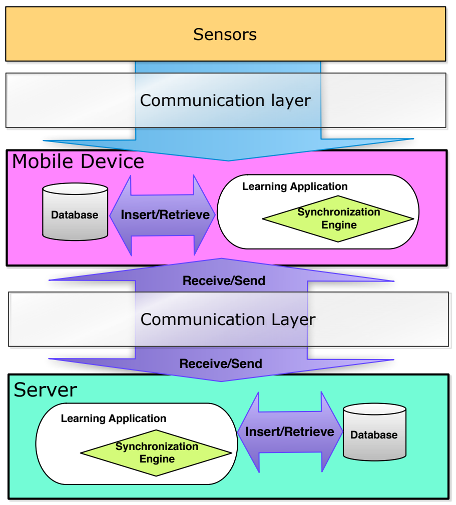

Figure 1: Building Blocks of the Pattern.

The sync engine in this pattern is no difference than a standard data synchronization pattern. It has the following basic responsibilities; keep track of the changes in the local copy of the data, exchange data with other synchronization engines when connectivity is available, detect conflicts, and resolve them. The major difference is that it initiates the synchronization automatically based on the user location and follow the domain specific policies.

2.2 Pattern Applicability

There are specific situations and problems when a developer will need this pattern. Data that needs to be replicated should be a row in a table. The replication set is a table in the database, this table consists of sensor data. The replication set is to be copied from a single source to a mobile device, and possibly to more than one mobile device. The data consists of entire rows, not just changes that have occurred to rows since the last replication. Any changes made to the replication set at the mobile device that may have occurred since the last transmission will be overwritten by a new transmission. The replication set is possible to be updated at both the mobile device and the server. Data on all the participating mobile devices including the server should have the same copy of the data. Conflicts needs to be detected and resolved at a specific time and place.

Based on the specific application domain targeted here, there are some design consideration the developer should take inconsideration. The transmission frequency and the initiation of the transmission will be based on where the student is. The initiation of the synchronization from server to mobile device will be done automatically whenever the student reaches school. The initiation of the synchronization from the mobile device to the server will also done automatically when the student leaves a specific place.

2.2.1 Conflicts Detection and Resolution

In the pattern there are two synchronization engines, one is at the mobile device side and the other one is at the server side. These two synchronization engines work together to achieve two synchronization cases. First case is Terminal-To-Host, which is synchronizing changes in replication set from the mobile device to the server.

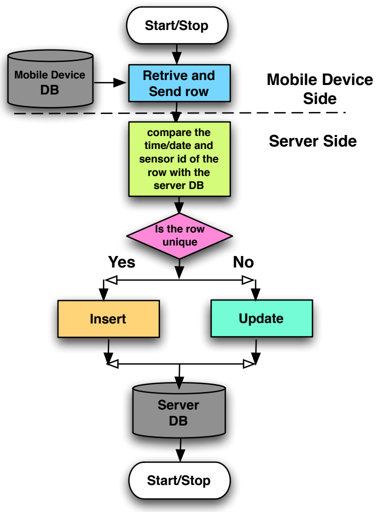

Figure 2: Algorithm for Conflict Detection and Resolution.

Conflicts should be detected before actually changing anything in the data on the server. The conflicts will be a result of inserts on the mobile device side. Inserts here means that new data was collected from sensors through mobile devices. These conflicts will occur when two or more mobile devices actually read from the same sensor at the same time, which will cause a uniqueness problem. Conflicts that result from update and deletion will not appear here since students and teachers are not allowed to delete or update the data that have been collected from sensors.

To detect conflicts in this case, the sensor id in addition to the time and date values of the rows coming from the mobile device will be compared to the rows in the server and if it matches any row in the server database the policy in this case would be that the incoming data will overrules the server data and so, the row sent to the server will overwrite the one stored on the server. In case the row coming from the mobile device did not match any row in the server database, it will be inserted into the server database directly. See the flowchart of this algorithm in Figure 2.

The second case is Host-to-Terminal, which is synchronizing a copy of the data from the server to the mobile device. This case of synchronization does not need conflict detection and resolving because in this case the mobile device need a replica from the data stored on the server. The data coming from the server will overwrite the data stored locally on the mobile device. With the server updated by the last Terminal-to-Host synchronization, all mobile devices will pull a unified copy from the server.

3. CASE STUDY

In this section a real life case study is presented. The system is a ubiquitous learning system that will be used in a K-12

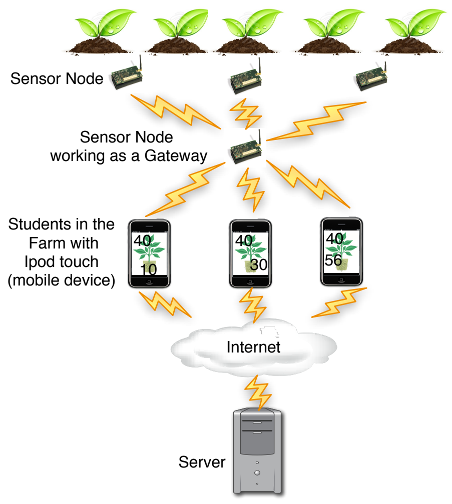
Sensor Node
Sensor Node working as a Gateway
Students in the Farm with Ipod touch (mobile device)
40
10
40
30
40
56
Internet
Server

Figure 3: Case Study.

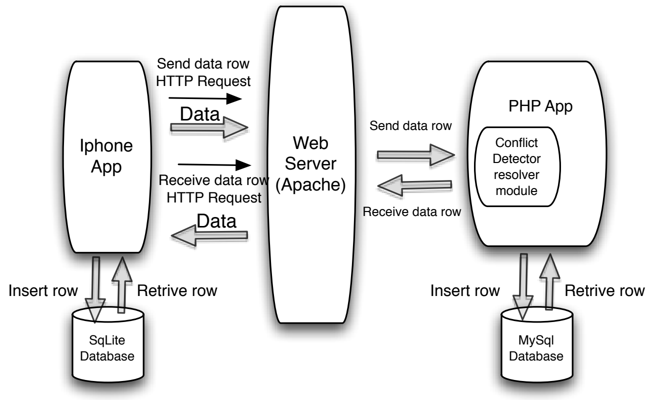

Figure 4: Implementation Architecture.

soil management class. In this class, the students should learn about how they can determine if the soil is healthy for the plant or not. Data about soil can be read using special sensors. Usually, teachers take student to farms and start showing them how to read data about soil and what this data means.

What the learning system will do in this case would be facilitating the learning process by using mobile devices to analyze and visualize the sensor data. Sensors that can sense data about soil like, soil moisture, ph level was planted in a farm that the student can visit later. Each one of the students will have a mobile device with him or her. When a student come close to a gateway sensor node, the mobile device will start reading soil data that were gathered by soil sensors. Each time the mobile device receives data it will store it locally in a database, see Figure 3.

The data collected by each student should be synchronized to a server in order to have all the data collected in one place (database). The data stored on the server database, will be shared among all the students later and used as an example in classroom.

There are some challenges that should be taken into consideration for this system to work successfully without interrupting the learning process. First, all students should have consistent replicas from the sensor data on their mobile devices, because in the classroom they will visualize the data using some applications and the teacher will give some explanation on this visualization. It is like the book the students use to study in classroom, all students should have the same version of the book otherwise they will not follow the teacher correctly.

Second, conflicts in sensor data should be taken care of because if more than one student reads sensor data from the same sensor node at the same time there will be redundancy in the data stored on the server and this will make it hard to visualize multiple versions of the same data.

4. IMPLEMENTATION AND EVALUATION

For the implementation of this pattern, the case study mentioned before was used. An iPod touch will be used by students and the server will be an Apache web server. The communication between the mobile device and the server will be through the internet using the HTTP protocol, see Figure 4.

IPod touch was chosen because other capabilities in the iPhone and iPad with obviously higher cost are not needed. The database engines that were used were SqLite on iPod touche and MySQL at the web server end.

The mobile device synchronization engine is an iPhone application built using Objective C. The server synchronization engine is a PHP application that is running on an Apache web server. The communication between the iPod and the Apache server will be based on HTTP protocol requests and responses.

If the mobile device is trying to synchronize with the server the changes that have been made to the locally stored replication set, it will send data to the server through an HTTP request. The changes here would be rows added to the mobile device replication set, read through the sensors. These added rows will be sent row by row to the server. When the row arrives to the server, it will be compared with the rows in the server database to make sure that it is unique. The comparison will be based on two columns in the table; the sensor id and the time and date columns. If the row is unique, it will be inserted to the server database directly. If not the new row will overwrite the matched row in the server database.

A primary evaluation for the implementation of the pattern was done. It showed that applying the presented pattern to the case study resulted in a reliable system and a consistent data on both the mobile device and the server. The evaluation was done based on around 100 simulated readings from sensors and sample possible conflicts were tested. The conflicts tested were around 15 simulated sensor readings and 100% of them were detected and resolved based on the presented conflict detection and resolution algorithm

5. CONCLUSION AND FUTURE WORK

Having a domain specific architectural pattern will definitely help and guarantee the reliability of the developed systems. This is because the pattern provides clear and simple guidelines. These guidelines will cut time spent in system design and development by inexperienced developers.

In this paper a design pattern for solving the data synchronization problem associated with ubiquitous learning systems that uses sensor data in adapting learning contents was presented. A full description of the pattern elements and pattern applicability was stated. A case study and implementation of were done to prove the applicability of the pattern.

In the future, an evaluation of the pattern implementation from the resource management and time perspectives will be done. Applying the pattern on similar applications that is not related to learning is also considered for future work.

6. ACKNOWLEDGMENTS

I would like to thank Eli Tilevich, assistant professor in the computer science department at Virginia Tech for the valuable and broad background he exposed us to in a research course that I took with him on software abstraction.

7. REFERENCES

[1] J. B. Almeida, P. S. Almeida, S. Almeida, and C. Baquero. Bounded Version Vectors. *DISC: International Symposium on Distributed Computing*, 2004.

[2] P. Almeida, C. Baquero, and V. Fonte. Interval Tree Clocks. *Principles of Distributed Systems*, pages 259–274, 2008.

[3] P. S. Almeida, C. Baquero, and V. Fonte. Version Stamps: Decentralized Version Vectors. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 544, Washington, DC, USA, 2002.

[4] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[5] Y.-W. Huang and P. S. Yu. Lightweight Version Vectors for Pervasive Computing Devices. *Parallel Processing Workshops, International Conference on*, 0:43, 2000.

[6] B. B. Kang, R. Wilensky, and J. Kubiatowicz. The Hash History Approach for Reconciling Mutual Inconsistency. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 670, Washington, DC, USA, 2003.

[7] Y. Li, X. Zhang, and Y. Gao. Object-Oriented Data Synchronization for Mobile Database Over Mobile Ad-hoc Networks. In *ISISE '08. International Symposium on Information Science and Engieering*, volume 2, pages 133 –138, dec. 2008.

[8] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Transaction on Software Engineering*, 9(3):240–247, 1983.

[9] D. Ratner, P. Reiher, and G. J. Popek. Dynamic Version Vector Maintenance. Technical Report CSD-970022, UCLA, June 1997.

[10] J. Roth. Patterns of Mobile Interaction. *Personal Ubiquitous Computing*, 6(4):282–289, 2002.

[11] S. Tarkoma. *Mobile Middleware: Supporting Applications and Services*. John Wiley and Sons Ltd, 2009.

[12] P. Teale, C. Etz, M. Kiel, and C. Zeitz. Data Patterns. Microsoft Corporation, June 2003.

[13] S. Venugopal, R. Buyya, and K. Ramamohanarao. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. *ACM Computing Surveys*, 38(1):3, 2006.

[14] V. Vincent Tam and B. Yin. Investigating Data Synchronization in a Mobile Learning Network with Handheld Devices. In *ITRE2003: Proceedings of International Conference on Information Technology: Research and Education*, pages 296 – 300, August 2003.