Data Synchronization and Conflict Resolution for Mobile Devices

YounSoo Kim and Hoon Choi

Dept. of Computer Engineering, Chungnam National University
Gung-dong, Daejeon 305-764, Korea
{kimys,hchoi}@ce.cnu.ac.kr

Abstract. Proliferation of personal information devices results in a difficulty of maintaining replicated copies of the user data in data stores of the various devices. Data synchronization is an automated action to make the replicated data be consistent with each other and up-to-date. The traditional consistency models and the conflict resolution schemes do not fit to the synchronization of replicated data in the mobile environment due to the lacks of simultaneous update. This paper defines types of data conflict that occurs in synchronization process of replicated data between the multiple client devices and a server along with the resolution rules for each conflict scenario in the recent-data-win policy. These rules have been implemented and verified in a data synchronization server which was developed based on the SyncML specification.

Key Words: data synchronization, consistency of replicated data, conflict resolution

1 Introduction

Advance of mobile communication devices and technologies has brought the mobile computing era. People can access Internet from anywhere, anytime by using personal information devices such as PDA(Personal Data Assistant)s, handheld PC(Personal Computer)s and cellular phones.

Proliferation of personal information devices has resulted in a difficulty of maintaining replicated copies of the user data in various devices' data storage. A user may have the same address book both in his cellular phone and PDA. If the user modifies a data item in his PDA, it causes the data being inconsistent with the replicated ones in other devices. Data synchronization is an automated action to make the replicated data be consistent and up-to-date with each other. Data synchronization needs to resolve conflicts between multiple values or operations made to the different copies of the data due to simultaneous updates on different devices.

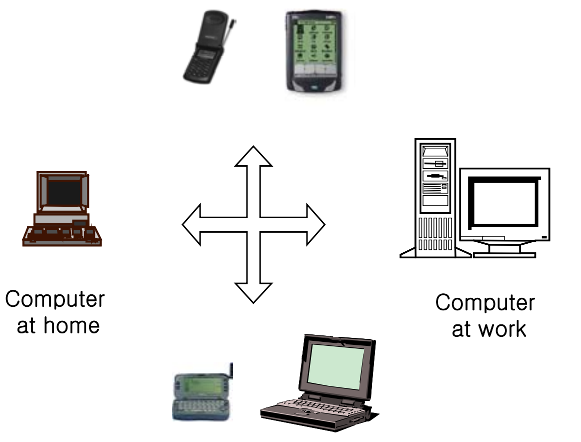


Figure 1. Data Synchronization in Mobile Environment

There are numerous studies about data conflict and consistency models in database and file system context. Balasubramaniam and Pierce [15], Ramsey and Csirmaz [16] studied about file synchronization in a replicated file system. While [15] showed the conflict resolution rules based on a condition change of a file system before and after synchronization, [16] presented the rule based on the operations applied to each replica file system. Terry [11], and Page [17] studied synchronization of replicated database. Synchronization of the distributed replicated data store, either it is the database or the file system, occurs in real-time, therefore the update in one data store propagates to the remaining data stores in real-time. In this model, the conflict mainly concerns multiple simultaneous write operations on the same data item and this is called the '*write-conflict*'.

However synchronization of replicated data of mobile devices has different characteristics. These devices are generally owned by one person and the propagation of the updates are made via one device, the server, there are seldom a simultaneous access to a data item nor the write-conflict. Also the updates made on a mobile device may not be propagated to other devices of the person until he initiates synchronization of the device to the server. Therefore update consistency model for data stores of mobile devices classifies to the eventual consistency [18] that does not require simultaneous update property. Therefore the traditional conflict resolution scheme does not seem to fit for this case.

The purpose of this study is to clarify the resolution rules for each conflict scenario. We classified the types of data conflict that may occur in synchronization processing of replicated data in multiple client devices and a server in mobile computing environment, then we identified the resolution rules for each conflict scenario. These rules have been implemented and verified in a data synchronization server which was developed based on the SyncML specification. Only the rules for recent-data-win policy has been presented in this paper due to the space limit. The paper is organized as follows. Section 2 describes the synchronization types, and Section 3 shows the conflict resolution rules, followed by implementation issue in Section 4.

2 Data Consistency Model for Mobile Device

Update patterns of replicated database are generally categorized to a master-slave model and a peer-to-peer model. In the master-slave model of replicated database, the update on the slave database is temporary and it becomes valid only after committed by the master database. On the other hand, each replica database has the same competence as master database and update in any database is valid in peer-to-peer model.

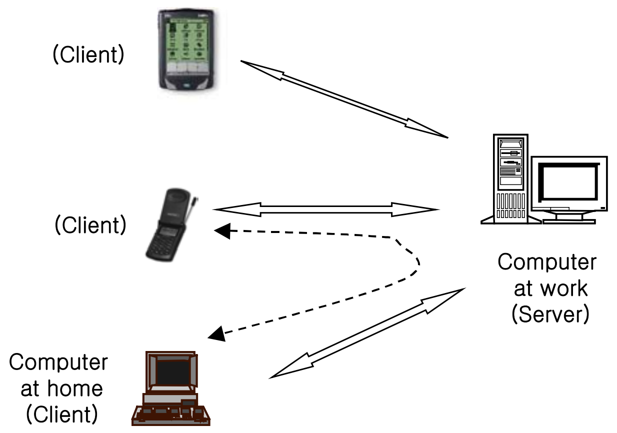
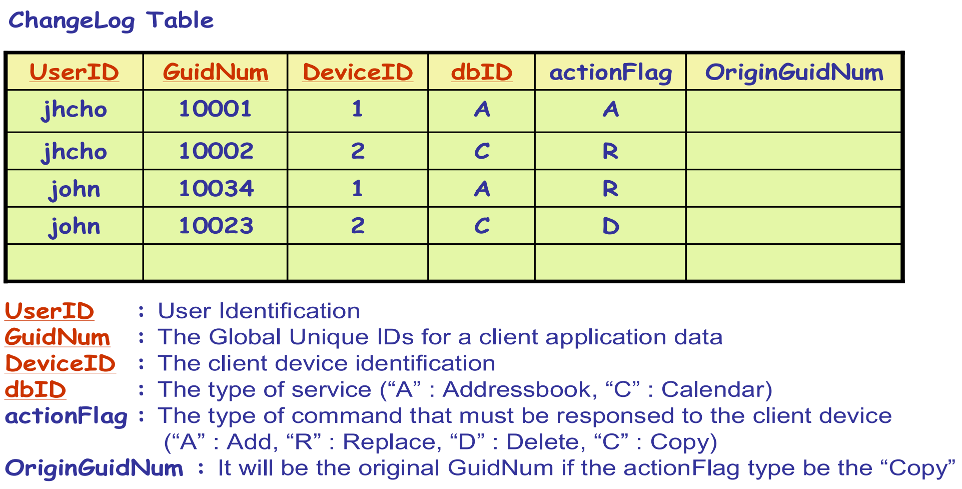


Figure 2. Master-slave model : replicated data stores in mobile environment

Because replicated data store in mobile computing environment is usually contained in a small, portable device like a PDA or a cellular phone, the data store has limited capacity and is less stable compare with desktop computers. Thus, it is a common way to take the master-slave model in which a desktop computer is used as the master data store (server) and portable devices are slaves(clients). In this model, when some data are modified at a client device, the update is available only in that device. In order to propagate the update to other devices, it communicates indirectly through the server(Figure 2). The server stores the modified data and maintains the records of modification made either by the server or by one of the clients, this record is used when the server synchronizes with other client data store. This record, the change-log information guarantees the consistency of replicated data in multiple devices (Figure 3). This paper is also based on the master-slave model.



ChangeLog Table

UserID	GuidNum	DeviceID	dbID	actionFlag	OriginGuidNum
jhcho	10001	1	A	A	
jhcho	10002	2	C	R	
john	10034	1	A	R	
john	10023	2	C	D	

UserID	: User Identification
GuidNum	: The Global Unique IDs for a client application data
DeviceID	: The client device identification
dbID	: The type of service (“A” : Addressbook, “C” : Calendar)
actionFlag	: The type of command that must be responsed to the client device (“A” : Add, “R” : Replace, “D” : Delete, “C” : Copy)
OriginGuidNum	: It will be the original GuidNum if the actionFlag type be the “Copy”

Figure 3. Information in the change-log table

Traditionally, replicated databases with a central master topology are regarded to avoid write-conflicts whereas replicated databases with a peer-to-peer topology must detect and resolve conflicts [9]. Also propagation of updates often occurs in a different time frame. An update occurred in one mobile device may not be propagated to other devices of the person until he synchronize the devices to the server. Consistency model for this case is the eventual consistency [18] that is characterized by the lack of simultaneous updates. Only if users perform synchronization operation with the server before he accesses the local data store, this model guarantees any of the client-centric consistency [19] which is applied to ordinary replicated data stores in the networked distributed systems.

The conflict that may occur during the synchronization of mobile devices is those between different operations that needs to be applied to a data item of a device by simultaneous access or different time frame, which we define the *‘semantic conflict’*. For example, suppose Device 1 has requested the server to ‘delete’ data A. After the server deletes the data A which is due to Device 1’s request, it records in the change-log table that ‘delete’ operation needs to be performed regarding the data A in other Device (e.g. Device 2, Device 3 etc.) which belong to the same user. Other device, say Device 2, may later request the server to ‘replace’ the data A with some other value without knowing that the data has been deleted by other device, this is an example of the semantic conflict. The resolution rules for such a conflict, i.e., actions to be taken to resolve the conflict should be defined at the server.

3 Semantic Conflict and Resolution

3.1 Conflict Resolution Policies

A conflict resolution policy is the criteria of which data to select as a valid one when two devices in synchronization have different values for the same data item. Table 1 shows possible conflict resolution policies. Services or synchronization applications may support all or subset of these policies. One of them may be selected by a service administrator or a device-owner.

Table 1. Conflict resolution policy

Conflict resolution policy	Meaning
Originator Win	Take the data item of the originator
Recipient Win	Take the data item of the recipient
Client Win	Take the data item of the client device
Server Win	Take the data item of the server
Recent Data Win	Take the data item which has been updated recently in time.
Duplication	Apply the requested modification on the duplicated data item while keeping the existing data.

Due to the space limitation, this paper deals only the conflict resolution rule of the Recent-Data-Win policy.

First, we need to define which point of time to look at to judge 'recentness'. Possible time points to compare can be listed as follows.

(1) User-update-time, T_U
(2) Client-access-time, T_C
(3) Server-access-time, T_S

User-update-time means the time point at which the user updated the data in the device. Client-access-time is the time point at which a client device initiated synchronization with the server. Mobile devices are not necessarily connected with the server (primary data store) all the time, they may be used independently and only be connected to and possibly synchronized with the server from time to time. That means a user may synchronize with server at the time he updates certain data, or may postpone synchronization to some time later. Finally server-access-time is the point of time that the server accesses its database to execute update requests. Roles of the client and server are relative. If the primary data store requests synchronization to mobile devices, client-access-time is the time when the server requests the synchronization.

To support Recent-Data-Win policy, data creation time or update time should be also stored with associated data item.

Adopting the user-update-time, T_U, seems most rational because it is based on user's intend to update data in his device. But system clock of the user device may not correct or intentionally tempered. The objectiveness in this case is weak and may even cause a security problem. T_C is based not by when the data is updated, but by when the data is synchronized. Adopting T_C is arguable bacause a data may be regarded as a recent one even though it might be modified quite a long time before the synchronization. Also, in correct system clocks of devices and non-deterministic communication delays may cause unfair winner.

Unless all the user's devices are well synchronized by a reliable global clock, T_U or T_C may not be trustable. Therefore, we consider the server-access-time in this paper.

3.2 Resolution Rules for the Recent-Data-Win Policy

When a record in the change-log information exists for a data item to be synchronized, semantic conflict can occur between currently requested operation and recorded operation in the change-log information. In this section, we show the conflict resolution rules for Recent-Data-Win policy to resolve conflict.

There may be many different synchronization operations but these operations can be classified into 'add', 'replace' and 'delete' type operations. For instance, the SyncML specification defines 12 commands, but they are represented by these 3 basic commands [6].

Conflict Resolution processing consists of the following two phases. Detailed actions in each phase depend on the conflict resolution policy.

Phase 1 decides which data wins among the server's value and the client's one.
Phase 2 updates the change-log information accordingly in the server.

Data stores in the server or client devices manage their data records by unique system-level identifier, not by the content of the data. For example, a data 'Tom' with system identifier 245 is different one from 'Tom' with system identifier 597. Commonly used identifiers are GUID (Globally Unique Identifier) and LUID (Locally Unique Identifier). LUID is usually used in the client device and it is meaningful only in that local data store. GUID is used in the server and is meaningful in service system wide.

When a new data item is added in a client device, a new LUID is assigned to the data by the client device and used locally in that client until the data is removed. When this new data along with the LUID is sent to the server for the synchronization, a new GUID is assigned to the data item by the server. The (GUID,LUID) pair associated with the newly added data is kept in the server.

Whichever time point of section 3.1 is used, the server compares the time associated with the data item in the change-log information with the time with the current operation, then resolves the conflict as the rules in Table 2. In the table, $op^t(o_i)$ denotes the current operation to be performed in the server on the data item o_i and $op^{t-1}(o_i)$ denotes the operation performed previously on the same data item, i.e., the data item with the same (GUID,LUID) pair.

Table 2 Resolution Rules for the Recent-Data-Win Policy

$op^t(o_i)$	$op^{t-1}(o_i)$	Action	Rule
add	add	error	T.1
	delete		
	replace		
delete	add	delete/error	T.2
	delete	delete	T.3
	replace		
replace	add	replace/error	T.2
	delete	replace	T.4
	replace		

Rule T.1 : This is the case that a client has requested to add a data item which was assigned some GUID in the server, but there has been operation $op^{t-1}(o_i)$ (either it is add, delete or replace) performed previously on this data item. This cannot happen considering GUID is unique. Either the server has assigned a wrong GUID or the client sent a wrong command that is 'add'. It is implementation dependent to decide which side is faulty and how to handle the exception, but the change-log information should not be modified in any case.

Rule T.2 : It is the case of deleting (or replacing) the data item which has been added previously, $op^{t-1}(o_i)$= 'add'. This case consists of two sub cases.
T.2.1 If there is an entry in the change-log information which is for o_i with this device's identifier, i.e., the device which has requested $op^t(o_i)$, and 'add' was the operation performed previously, in other words :

$$\text{changelog}[\text{DeviceID}(op^t(o_i))] = op^{t-1}(o_i) = \text{add}$$

then treats it as an error by the same reasoning of Rule T.1.
T.2.2 Otherwise, i.e. :

$$\text{(case 1) changelog}[\text{DeviceID}(op^t(o_i))] = \text{null} \quad \text{or}$$

$$\text{(case 2) DeviceID}(op^t(o_i)) \neq \text{DeviceID}(op^{t-1}(o_i))$$

it is not an error. This happens when a device wants to delete (or replace) o_i after itself added o_i in the previous synchronization processing (case 1). Another case that it can happen is that o_i was added to the server by some other device but o_i has not been synchronized with this device (case 2). In this case,

Phase 1 : the server deletes o_i

Phase 2 : other devices that have records for them in the change-log information have not synchronized yet, so the records in change-log can be silently deleted.

In case the server wants to replace it,

Phase 1 : the server replace o_i by executing $op^t(o_i)$

Phase 2 : add an entry in the change-log with 'replace' for other device.

Client devices will handle 'replace' in the change-log information as 'add' if o_i does not existent in its local data store.

Rule T.3 : This is the rule to handle 'delete' request after o_i has been deleted or replaced previously.

First, search the change-log information to find entries with the same device identifier as the device which requested $op^t(o_i)$:

T.3.1 : if $\text{changelog}[\text{DeviceID}(op^t(o_i))] = op^{t\text{-}1}(o_i) = \text{delete}$

Phase 1 : no action is required because it has been already deleted from the server.

Phase 2 : delete the change-log entry which is for o_i with this device's identifier

T.3.2 : if $\text{changelog}[\text{DeviceID}(op^t(o_i))] = op^{t\text{-}1}(o_i) = \text{replace}$

Phase 1: delete o_i

Phase 2 : delete the change-log entry which is for o_i with this device's identifier

Then, investigate the change-log entry which is for o_i but with different device identifier :

T.3.1 : if $op^{t\text{-}1}(o_i)$ = delete, discard the request because o_i has been already deleted. (null Phase 1 and Phase 2)

T.3.2 : if $op^{t\text{-}1}(o_i)$ = replace

Phase 1 : null,

Phase 2 : change the operation of change-log entry to 'delete'.

Rule T.4 : This is the rule to handle 'replace' request after o_i has been deleted or replaced previously.

First, search the change-log information to find entries with the same device identifier as the device which requested $op^t(o_i)$:

T.4.1 : if $\text{changelog}[\text{DeviceID}(op^t(o_i))] = op^{t\text{-}1}(o_i) = \text{delete}$,

Phase 1 : add o_i with the previous GUID in the database

Phase 2 : delete the entry in the change-log.

T.4.2 : if $\text{changelog}[\text{DeviceID}(op^t(o_i))] = op^{t\text{-}1}(o_i) = \text{replace}$,

Phase 1 : change the data record value to the requested data value

Phase 2 : delete target entry in the change-log.

Then, investigate the change-log entry which is for o_i but with different device identifier :

T.4.1 : if $op^{t-1}(o_i)$ = delete,

Phase 1 : null because o_i has been processed already in T.4.1 of the first step.

Phase 2 : modify the change-log to 'replace' because the item being requested 'delete' is not deleted yet

T.4.2 : if $op^{t-1}(o_i)$ = replace,

Phase 1 : replace o_i with the requested data value

Phase 2 : null

4 Implementation

We have developed a data synchronization server and implemented the aforementioned conflict resolution rules in the server to verify the rules. We used the SyncML protocol suites from the SyncML Initiative for the communication between client devices and the server [1,6,7,8]. The server provides vCard and Vcalendar applications. The server successfully accomplished synchronization according to various scenarios [5] with multiple client devices.

The Server Application provides an interface for a user or the application administrator to modify contents data at the server. The SyncML Toolkit encodes and decodes SyncML messages. The Sync Agent implements the SyncML protocols. Therefore, its function is application independent and can be commonly used for all SyncML applications. On the other hand, the Sync Engine implements application dependent part such as a service policy, resolution rules in case of data conflict, etc. The Session Manager manages a temporary, session related information that needs to be maintained in the server during a session. Lastly, the Open DB Interface implements interfaces to access data in database.

All the frames of the CNU SyncML Servers except the Session Manager were implemented into DLLs. We used HTTP(Hyper Text Transfer Protocol) to communicate with a SyncML client device and JNI(Java Native Interface) to load DLLs.

We verified the server based on the conformance test specification of the SyncML Initiative [1]. The SyncML conformance test scenarios consist of exchanging messages that include various commands such as Add, Replace, and Delete between clients and the server. We also used our own test scenarios intensively for synchronization tests for two application services; an address book and a calendar.

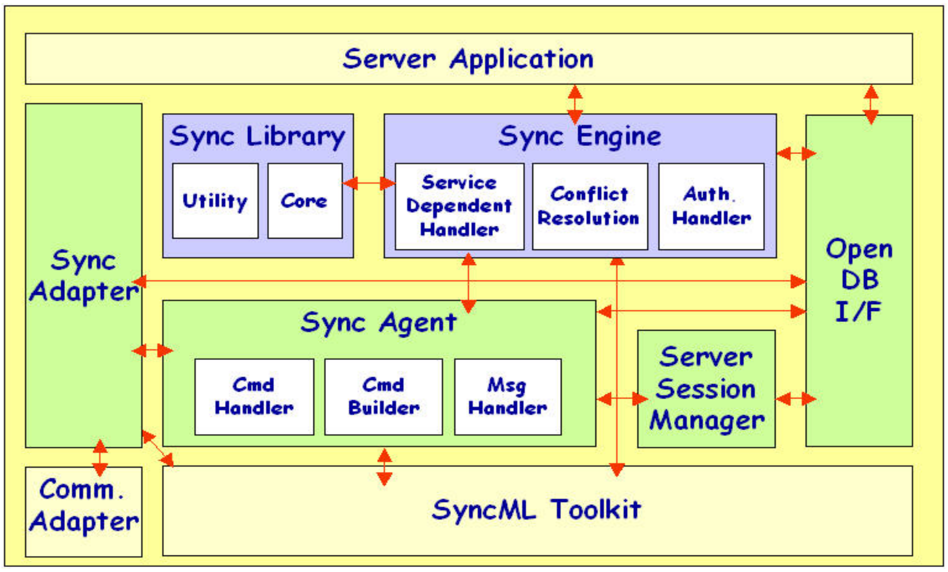


Figure 4. CNU SyncML Server Framework

5 Conclusions

As mobile communication technologies are being developed faster than ever, more people will get benefit in the mobile communication environment. Also device technology and battery power will be improved in near future. In spite of technology improvement, it is natural to think that portable, mobile devices will have less data storage and stability than desktop server computers. So, we need data synchronization to backup data in a mobile device to a reliable server computer, or to make replicated data be consistent with each other. A user may have an address book in a cellular phone and the same data in his PDA as well since he can use multiple devices.

The semantic-conflict may occur when we access the replicated data. This study showed the conflict resolution rules when multiple devices synchronize their data by the recent-data-win policy in the mobile computing environment. The conflict resolution rules and consistency model of the mobile computing has some different characteristics from traditional replicated databases in the wired network. Main characteristic is that the propagation of updates often occurs in a different time frame. An update occurred in one mobile device may not be propagated to other devices of the person until he synchronize the devices to the server. Therefore thee consistency model for this case is the eventual consistency that is characterized by the lack of simultaneous updates.

We hope that this study can contribute in implementing various synchronization applications while avoiding subtle incorrect processing of data synchronization.

References

[1] SyncML Initiative, Building an Industry-Wide Mobile Data Synchronization Protocol, SyncML White Paper, Mar. 20, 2000, http://www.syncml.org

[3] Microsoft, ActiveSync Technology, http://www.microsoft.com/ mobile/pocketpc/ downloads/activesync/

[4] Palm Computing Inc., Palm HotSync Technology, http://www.palm.com/support/hotsync.html

[5] JiYeon Lee, ChangHoe Kim, Jong-Pil Yi, Hoon Choi, "Implementation of the Session Manager for a Stateful Server," 2002 IEEE TENCON, Beijing, pp. 387-390, Oct. 31, 2002.

[6] SyncML Initiative, SyncML Representation Protocol, version 1.0.1, June 15,2001

[7] SyncML Initiative, SyncML Synchronization Protocol, version 1.0.1, June 15,2001

[8] SyncML Initiative, SyncML HTTP Binding, version 1.0.1, June 15,2001

[9] M. Dahlin, A. Brooke, M. Narasimhan, B. Porter, "Data Synchronization for Distributed Simulation," European Simulation Interoperability Workshop, 2000

[10] M. Satyanarayanan, J. Kistler, Kumar, et al. , "Coda: a highly available file system for a distributed workstation environment," IEEE Trans. On Computers, Vol. 39, No. 4, pp.447-459, 1990.

[11] D. B. Terry at al., "Managing update conflicts in Bayou, a weakly connected replicated storage system," Proc. of the 15th Symposium on Operating Systems Principle, 1995.

[12] P. J. Braam, "InterMezzo file system: Synchronizing folder collections," http://www.stelias.com

[13] PostgreSQL Replicator by Matteo Cavalleri, Rocco Prudentino PostgreSQL Replicator: Conflict Resolution Algorithms,", http://pgreplicator.sourceforge.net/

[14] M. Cavalleri, R. Prudentino, U. Pozzoli, G. Reni, "A set of tools for building PostgreSQL distributed databases in biomedical environment," Proc. of the 22th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Jul. 2000.

[15] S. Balasubramaniam and B. C. Pierce, "What is a file synchronizer?" In Proc. of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM-98), pp. 98-108, Oct. 1998. (http://www.cis.upenn.edu/~bcpierce/unison)

[16] N. Ramsey and E. Csirmaz, "An Algebraic Approach to File Synchronization," ACM SIGSOFT Software Engineering Notes, Volume 26, Issue 5, pp.175-185, Sep. 2001.

[17] Thomas W. Page et al. "Perspectives on optimistically replicated, peer-to-peer filing," Practice and Experience, Vol. 27 (12), December 1997.

[18] A. S. Tanenbaum and M. Steen, Distributed Systems, Principles and Paradigms, Prentice Hall, 2002.

[19] D. B. Terry, K. Petersen, M. Spreitzer, and M. Theimer, "The case for non-transparent replication: Examples from Bayou," IEEE Data Engineering, Vol.21, No. 4,pp.12-20, Dec. 1998.