A Lightweight Approach to Contact Data Synchronization in Mobile Social Networks

Nikolay Tkachuk[1], Alexey Vekshin[1], Konstantyn Nagorny[1] and Rustam Gamzaev[1]

[1] National Technical University "Kharkov Polytechnic Institute", Frunze str. 21, 61002 Kharkov, Ukraine
tka@kpi.kharkov.ua, {alexeyvekshin, k.nagornyi, rustam.gamzayev}@gmail.com

Abstract. Data synchronization is one of the most critical issues in "always-available" software mobile applications development. In this paper the new approach to resolve this problem in social networks is proposed, which is based on iPhone platform, and utilizes legacy data storage based on MS .Net WCF (RESTFull services) and MS SQL Server. This application provides import of client's contacts data (e.g. from iPhone's address book) created previously in another application (MS Outlook, etc.) into social networks, and supports their updating by further synchronization process. The advantage of proposed approach is it's universality and lightweight, because it does not need to implement any special software adapters and interfaces.

Keywords: data synchronization, mobile application, XML-mapping.

1 Introduction: Research Actuality and Aims

Nowadays many customers need to access information any time anywhere. For this purpose different mobile software- and hardware platforms can be used, which allow to store data in various formats and to provide diverse access interfaces. There is an obvious necessity for centralized data storage and access them in such distributed systems. Thus, data synchronization (DS) issue is one of the most significant problems in development of "always-available" software mobile applications. In this paper we propose an approach to solve a DS problem during development of a mobile client application in social network which is based on iPhone platform and uses legacy data storage based on MS .Net WCF (RESTFull-services) and MS SQL Server. Our system implements the possibility to import into social network the client's contacts data (e.g. from address book), which were created previously in another application (e.g. MS Outlook), and provides their modification with further synchronization. The paper is structured in following way: Section 2 depicts briefly some modern trends in this re-search domain, in Section 3 our approach is represented, the appropriate software solution and its complexity estimation are discussed in Section 4. Finally, Section 5 concludes the paper and gives a short outlook on some future works in this research.

2 Contacts Data Synchronization Issues in Social Networks: Some Modern Trends

Nowadays there are few leading mobile platforms, e.g. iOS, Windows Mobile, Symbian, Android, and most of business customers require solutions which cover all these technologies. In any case the requirement to have contacts data synchronized with remote servers or with other devices is strictly required even for desktop applications, but definitely this is a critically important issue for modern mobile software systems, especially for social networks. This requirement has to be met together with some additional special constraints of mobile applications such as e.g. performance sufficiency and battery life [1].
On Fig. 1 a typical scheme of several interactions in social network is presented. As it is shown on this diagram, each client application: iPhone, MS Outlook, etc., and system's server as well have own storage with contacts data. The centralized database on the server-side contains contacts data from all interacting clients. One of all tasks to be realized in this system is the DS procedure between different clients and centralized database on the server.

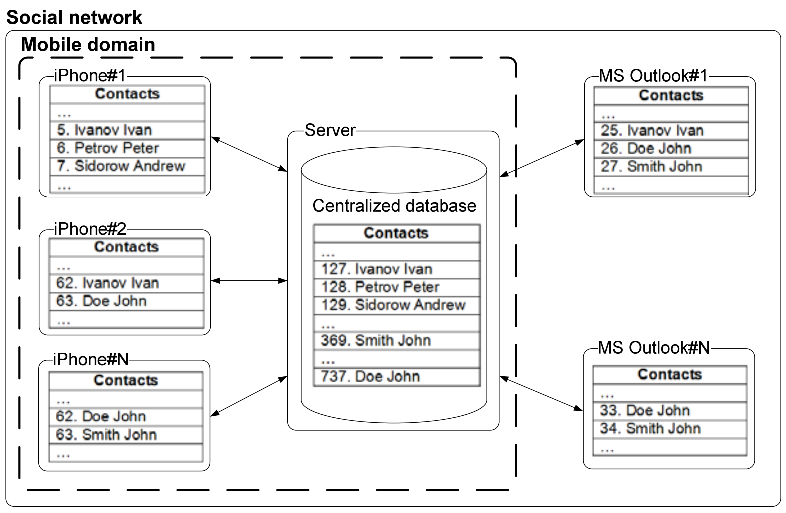

Fig. 1. Typical scheme for DS processes in social networks.

There are some special tools to solve DS problems. For example, SyncML [2] is a specification standard for common DS framework, but the main domain of SyncML is DS for mobile devices which are intermittently connected to network ser-vices. SyncML was specially designed for case, if data to be synchronized have different formats or are used in different software systems. Therefore SyncML's specification is too expensive from the development's efforts point of view, because it contains a lot of redundant options for different network systems development.

ActiveSync [3] is the software tool which allows to synchronize contacts and PIM-data [4] between mobile devices (e.g. mobile phone, communicator, pocket PC) and server which is running on Microsoft Exchange Server platform. This software requires connection between PC and devices via USB-cable, Bluetooth or IR-port.

MobiLink [5] is the Sybase DS technology from Sybase iAnywhere product line. It is session-based synchronization technology for data exchange between relational data-bases and non-relational data storages. Mobile client using MobiLink technology was implemented for Windows Mobile, but it is not possible to use this solution on iOS platform.

One of open source DS tools is Funambol service [6] with client applications for mobile devices and personal computers, it allows to synchronize contacts, PIM- data, emails and social networks data.

Summarizing features of mentioned above technologies we can conclude that it is too difficult to adopt them by real-life mobile application development taking into account problems with different data exchange formats, communication protocols, etc. Also usage of such tools is actually related with redundant data storage on client's application, and also needs to cleanup non-actual data on the server side. Moreover usage of existing technologies for the DS is expensive from the development costs point of view in rather small mobile software projects.

3 Formal Definitions and Algorithm for Contact Data Synchronization

Proposed DS approach can be represented in a formal way using the following definitions.

Definition 1. Data synchronization (DS) is a process, which is given by the tuple

$$DS = \langle XMF, \quad Ph, \quad I \rangle, \tag{2}$$

where XMF is contact identifiers mapping file (see for details Def. 2);

Ph is a set of synchronization phases (see for details Def. 5);

I is a set of queries to a synchronization server (see Def. 6).

Definition 2. Mapping file (XMF) is represented by the tuple

$$XMF = \langle CM, \quad LST \rangle, \tag{3}$$

where *CM* is a set of contact identifiers (see for details Def. 3);

LST is a set of synchronization timestamps (see for details Def. 4).

Definition 3. Contacts mapping (CM) is contacts identifiers mapping, represented by subset of Cartesian product

$$CM \subset SrvAb \times ClnAb, \tag{4}$$

where $SrvAb = \{s_k\}, k = \overline{1,m}$ is a set of contacts from server's address book;

$ClnAb = \{a_j\}, j = \overline{1,n}$ is a set of contacts from client's address book.

Definition 4. Last synchronization time stamp (LST) is a time of last contacts synchronization,

$$LST = \{t_l\}, \begin{cases} LST = \varnothing, if\ no\ DS provided\ yet \\ LST = \{t_1\}, if\ DS\ already\ provided \end{cases}. \qquad (5)$$

Definition 5. Synchronization phases (Ph) are procedures to be performed in synchronization process, represented by a following set

$$Ph = \{p_i\}, i = \overline{1,4}, \qquad (6)$$

where each phase p_i is a subset of create(C)-, read(R)-, update(U)- and delete(D)-operations:

$$p_i \subseteq \{C,\quad R,\quad U,\quad D\}. \qquad (7)$$

Definition 6. Request to synchronization server (I) – is a set of data types, which are transferred from client to server:

$$I \subseteq Q \times W, \qquad (8)$$

where $Q = \{q_v\}, v = \overline{1,N}$, N – set of natural numbers;

$W = \{w_z\}, z = \overline{1,3}$, e.g. $W = \{timestamp, contact\ identifier, contact\}$.

Algorithm which is proposed includes following four main phases.

Phase I: fetch modified contacts in SrvAB; modify contacts in ClnAB, update XMF;

Phase II: upload locally modified contacts to a server; retrieve new contact from SrvAB_ID; update XMF with obtained IDs.

Phase III: obtain from XMF the removed ClnAB_IDs; fetch contacts from server with these IDs; *user action request:* restore/delete contacts; *if restore:* insert contacts in ClnAB and update XMF; *if delete:* remove contacts from SrvAB and XMF.

Phase IV: send IDs from XMF to a server; find non-existent contacts in SrvAB by retrieved IDs, return them to a client; *user action request:* restore/delete contacts; *if restore:* upload contacts to a server, update XMF with IDs; *if delete:* delete contacts from ClnAB, update XMF; finally show synchronization report.

In Fig. 2 a sequence diagram of proposed approach in UML 2.0 notation is shown. This view describes a sequence of interactions between User, Client application on mobile device, and Synchronization server. Main synchronization phases are represented as a sequence of method calls or messages. Synchronization process starts after user interaction with mobile device, while synchronization in progress a mobile application displays dialog's confirmation messages, they allow to get control information for user. Interactions between Client application and Synchronization server are method invocations in RESTFull service.

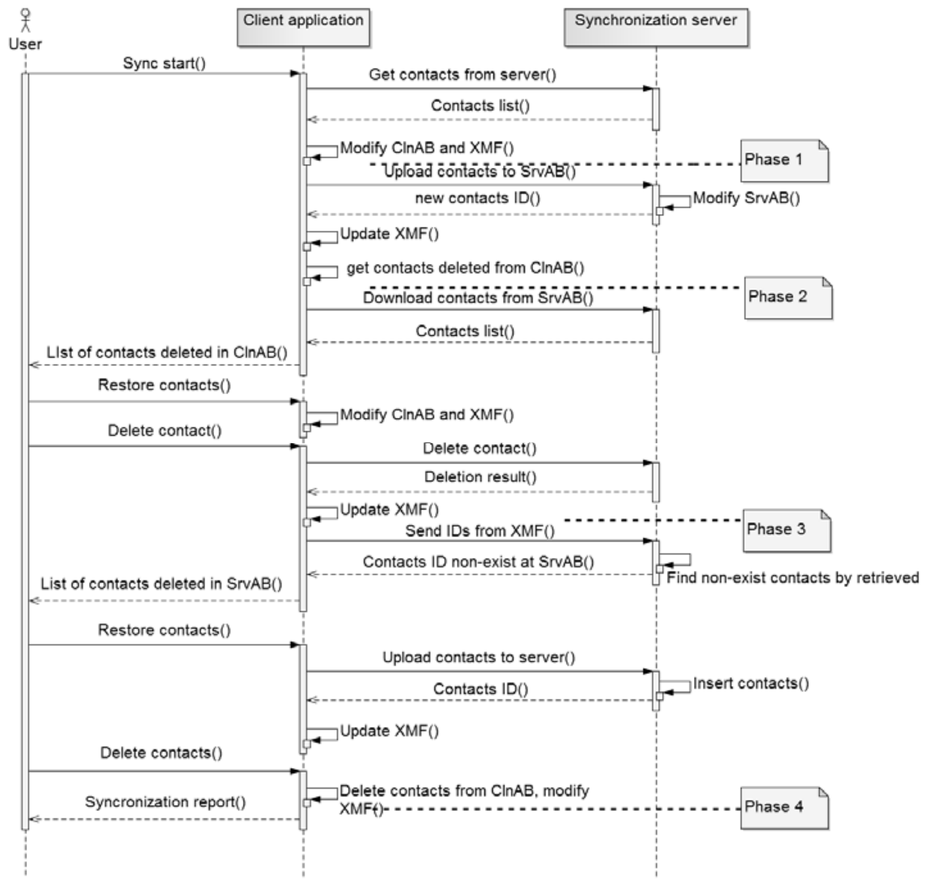

Fig. 2. Synchronization interaction sequence.

In order to provide DS process the appropriate data models, methods and software tools have to be elaborated.

4 Prototype Implementation and Complexity Estimation of Proposed Approach

On Fig. 3 the deployment diagram of typical mobile application including new software components is shown. Main nodes at this diagram are: the Synchronization server and the iPhone as a mobile device. At iPhone side a mobile application is deployed, which consists of following components: 1) *ViewController* is an application controller, which handles events, invokes *Model* and *ABProvider*; 2) *Model* is a component, which implements business logic of mobile application; 3) *ABProvider* is a component to provide access to iPhone’s local address book and to implement CRUD-operations; 4) *SyncServiceProvider* is a component to access remote REST-service and to utilize preparing request and parsing response; 5) *MappingFile* is a component accesses XMF. At the Server side the *SyncServer* node is presented with following components: 1) *SyncService* is a RESTFull-service implemented using C# and WCF technology to access server data with CRUD-operations; 2) *Centralized* database is a central storage of contacts data. As communication protocol the HTTP is used.

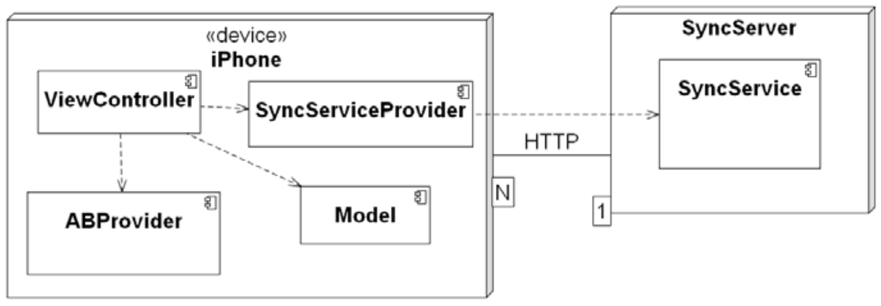

Fig. 3. The main nodes and components of proposed approach.

In a case of usage proposed approach next several components have to be implemented: 1) *XMF* mapping file, to store identifiers; 2) *SyncServiceProvider* synchronization service client. That is why from our point of view the proposed approach has less complexity as compared with another DS tools, e.g. like SyncML and Funambol (see Section 2).

In order to compare these approaches correctly, we need to describe typical software components for each tool in the same notation, and to estimate their complexity in some way.

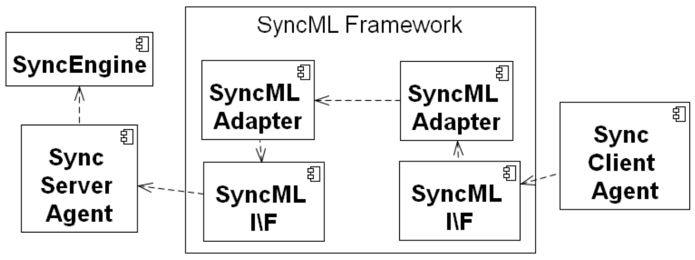

Fig. 4. SyncML-based DS approach components.

E.g., in case of SyncML framework there are some components are needed to implement for DS process: 1) 2 SyncML-adapters; 2) SyncML-engine; 3) SyncML client agent; 4) 2 SyncML I\F (API to SyncML-adapter); 5) SyncML server agent.

The SyncML based DS-architecture is shown in Fig. 4 as the UML component diagram.

In case of Funambol tool it is also needed some components to be implemented: 1) a SyncML-adapter, 2) Input and Output synclets (Java adapter classes); 3) Synchronization Sources (BTW: additionally some back-end classes have to be used in this approach, but they should not be taken into account for our comparison). In Fig. 5 the components of Funambol-based solution are shown [7].

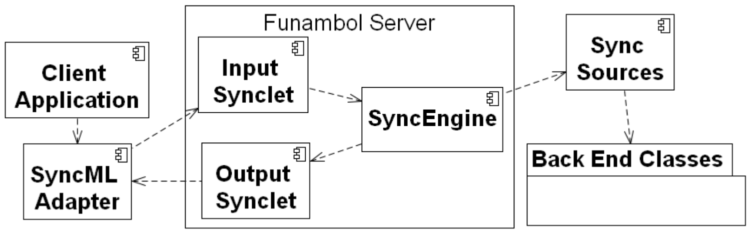

Fig. 5. Funambol-based DS approach components.

As usual frameworks have to be compared by their functionality and structures (see e.g. in [8, 9]), so to calculate complexity it is possible to calculate number of components, divided in some groups. In our case there are two groups of components: the components which have to be implemented, and the components which are already available. To calculate complexity the following formal expression can be used:

$$C = \sum_{i=1}^{z} \rho_i \cdot c_i \ , \qquad \textbf{(9)}$$

where: z is the number of component groups;

ρ_i is the weighting coefficients for components of i-th group;

c_1 is the number of components in i-th group.

The following test case uses weight coefficients calculated with Analytic Hierarchy Process (AHP) method [10], the appropriate coefficients are: $\rho_1=0,17$, $\rho_2=0,83$. With respect to expression (8) there are the following final values of complexity estimation:

1. for proposed approach: $C_{proposed} = 0,17 \cdot 2 + 0,83 \cdot 0 = 0,34$;
2. for SyncML framework: $C_{SyncML} = 0,17 \cdot 7 + 0,83 \cdot 0 = 1,19$;
3. for Funambol tool: $C_{Funambol} = 0,17 \cdot 4 + 0,83 \cdot 1 = 1,51$.

Results of complexity estimation expose that the proposed approach has less complexity than SyncML and Funambol both.

5 Conclusions and Future Work

We have presented the lightweight approach for contact data synchronization in mobile social networks, which allows to reduce development costs and to elaborate reusable software solutions. Of course, there are some problems in our approach which were not discussed in this paper. For example, we did not take into account the fact that contact data in any social network are surely private information, therefore a correct synchronization procedure has to provide an appropriate data security options, etc. Another critical issue in the proposed approach is an intensive data exchange process between a lot of client applications and centralized data storage, and it can lead to "bottleneck" effect in the synchronization framework. That is why in future work we are going to solve these problems, and additionally to improve our approach in the way of advanced analysis of contacts data to be synchronized in order to prevent possible semantic errors and data missing.

References

1. Mobile Platform Benchmarks. A Methodology for Evaluating Mobile Computing Devices. Daniel McKenna. Transmeta Corporation (2000)
2. SyncML Data Synchronization Protocol, http://www.openmobilealliance.org/Technical/release_program/ds_v1_2_2.aspx
3. Microsoft Active Sync, http://msdn.microsoft.com/en-us/library/aa913903.aspx
4. The Return of the PDA, http://memex.org/thereturnofthepda.html
5. Designing Mobile Applications: Why Sync Is Central, Sybase iAnywhere, (2007)
6. Funambol Project, http://www.funambol.com
7. Funambol Documentation, https://www.forge.funambol.org/download/documentation.html
8. Compare JavaScript framework, http://www.ibm.com/developerworks/web/library/wa-jsframeworks/
9. Best Web Frameworks, http://www.bestwebframeworks.com/
10. Saaty, T.L.: The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation. McGraw-Hill (1980)