# Synchronization with SyncML

István Balázs Sipos

Bachelor's Thesis
November 2009

Degree Programme in Information Technology
School of Technology

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

| | | |
|---|---|---|
| Author(s)<br><br>SIPOS, István Balázs | Type of publication<br>Bachelor´s Thesis | Date<br>10.12.2009 |
| | Pages<br><br>45 | Language<br>English |
| | Confidential<br><br>( ) Until | Permission for web publication<br>GRANTED |
| Title<br><br>Synchronization with SyncML | | |
| Degree Programme<br><br>Information Technology | | |
| Tutor<br><br>PELTOMÄKI, Juha | | |
| Assigned by<br><br>RAEHALME, Thomas - Abakus Softwares Ltd. | | |

Abstract

The thesis was assigned by Abakus Ohjelmistot Oy, which is a Jyväskylä based company with the business idea of helping companies with mobile and web solutions. The goal of the thesis was to build a functional demonstration of the synchronization between SyncML compatible mobile client and to provide a common interface for the server-side functionality to connect any data source for the synchronized content. The synchronized content is the Abakus calendar (called aCal).

The Funambol Data Synchronization Server was created to help synchronization based on SyncML standards. The thesis discusses the basics in the design of SyncML and Funambol as well as the development of Funambol Connector.

Important theoretical matters as conflict resolution, synchronization types, id mappings are presented. At the development of Connector can be seen the usage of DAO and Facade design patterns as well as Hibernate, Spring ORM and Maven, which are also described through detailed explanations, figures of the architecture and code.

Keywords

SyncML, Funambol, Java, Synchronization, Hibernate, Spring, Maven.

Miscellaneous

# CONTENTS

## FIGURES

## ABBREVIATIONS

DTD Document Type Definition

GUID Global Unique Identifier

HTTP Hypertext Transfer Protocol

IMEI International Mobile Equipment Identifier

LUID Local Unique Identifier

GUID  Global Local Unique Identifier

OBEX Object Exchange protocol

WSP Wireless Session Protocol

XML Extensible Markup Language

DAO Data Access Object

OMA Open Mobile Alliance

OTA Over-the-air programming

PIM Personal Information Manager

IoC Inverson of Control

## DEFINITIONS

Client Modification – A modification of an item, which occurs in a client database before the modification is synchronized to the server database.

GUID (Global Unique Identifier) – A number assigned to an object in a database. GUID values are never reused. Note that in practice, numbers do not have to be unique forever, they MUST only be unique as long as they exist in some mapping table (also see LUID).

LUID (Locally Unique Identifier) – A number assigned to an object in a database. LUID values are only unique locally, i.e., to a particular SyncML client database, but MAY be present on other SyncML client databases. In this protocol, the SyncML client device assigns to each object a locally unique, non-reusable identifier, or LUID. They are unique per device and per application.

Server Modification – A modification of an item, which occurs in the server database before the modification is synchronized to the client database.

Slow Synchronization – When a data set is synchronized for the first time, or state relating to the synchronization has been lost, the whole data set must be copied from one device to the other. Since this can be a time-consuming operation, this is known as slow synchronization.

Synchronization Anchor – A string representing a synchronization event. The format of the string will typically be either a sequence number or an ISO 8601-formatted extended representation, basic format date/time stamp.

Synchronization Engine – The portion of a SyncML server that can analyze a data set and modifications to that data set made by both SyncML server and SyncML client. The synchronization engine will implement policies to enable the detection and resolution of conflicting changes.

Temporary GUID – A temporary number assigned by the server to an object in a database (see also GUID.). Temporary GUID values are valid till the map operation for the items, with which the temporary GUIDs are associated, has been received from the client. After that the temporary GUID can be erased.

Push Email – It is used to describe email systems that provide an always-on capability, in which new email is actively transferred as it arrives by the MDA (mail delivery agent or as named mail server) to the mail user agent (MUA or called email client).

# 1 Objective of the project

There has been an amazing increase in the number of personal devices. Most of the people have at least a desktop PC or/and a laptop and a cell phone. The users reach their email service (like GMail, Yahoo or Hotmail and so on) by a web application or a client (Thunderbird, Outlook) and have their own personal information (contacts, calendar, todo lists and so on), which is stored on the server somewhere or/and on PC or laptop.

Mobile phone users would like to do the same; hence the question is why the person who uses desktop computer at the workplace and laptop at home as well as mobile phone on the road is not be able to reach the same personal information?

Firstly, the user will be able because it requires only to connect to the same service provider regardless which devices are used. The main email service providers are capable of storing the typical personal information issues. This approach indicates to the users “just” have to have an Internet connection and finds out what they would like by browsing on the net.

For example, a user creates an entry in the Abakus calendar through a web application. This entry contains the exact time and the location of meeting. After two weeks this person needs to participate on the negotiation and the only thing what he brings with himself is the cell phone. The negotiation is held in a small village but she/he has no idea where exactly. She/He would already connect to net and check the company's calendar online but unfortunately, there is no connection at all.

What to do in order to avoid this unpleasant situation in the future?

The reality as the latter example showed is that the wireless devices are not always connected. Coverage is not universal yet, connections often get disconnected and roaming is expensive and difficult. The new network technologies are supposed to bring the always-connected approach to the users. However if it happen in the wireless world, it would still be not sure the users could prefer reaching personal information by network computing.

In the wired world the users prefer to keep the data and applications on the device locally instead of leaving them on a machine owned by someone else. Further, they do not want to rely on the network even if it is highly reliable. If the network computing idea did not work well in the wired world, it is even less likely to be successful on devices with limited memory and processing power.

The mobile operator's revenue model is based on charging per byte or minute so the more data is downloaded , the more the user pays. That is why users would naturally choose downloading once and synchronize over paying per each access. For example, in this case the user would not be forced to download all the data from the calendar every time when he wants to look for something.

The synchronization is significant because the data on a person’s device can be updated. The data might be synchronized according to the user's desires (manually) or specific times (scheduled) as well the application logic (application controlled). So the charge of the data downloaded is paid when really need.

## 1.1 Advantages of synchronization

An application that is capable of synchronization on the device is needed both in wireless or wired world.

The mobile applications can gain advantages of a mobile device's processor and memory and provide fast good responsiveness compared to web application. Synchronization sessions can run in the background when connectivity is available and transfer only the information that has changed since the last synchronization.

The local application that synchronizes is more effective than a network application, but the latter are more popular. The synchronization requires a local database on the device in order to keep the bulk of data, and the application itself. Certainly this has not meant problems at the computers for years, however, mobile devices nowadays can provide enough processing and storage capacity.

More powerful hardware is released; synchronization becomes more common due to the inclusion of SyncML in hundreds of millions of mobile devices.

## 1.2 Pre-study

The pre-study mostly focuses on the data synchronization part of the thesis. Since this was the area where the main part of the work was done, the following steps were taken:

- Studying of SyncML
- Evaluating minimum requirements
- Analyzing existing implementations to decide whether a SyncML server should be made or a ready one used
- Deciding if an own client should be made or a built-in SyncML client used on the phone

### 1.2.1 Minimum requirements for data synchronization

SyncML client:

- The client should be easy both to configure and use
- As much it is possible should be avoided installing client manually
- As much it is possible should be avoided designing of a new client

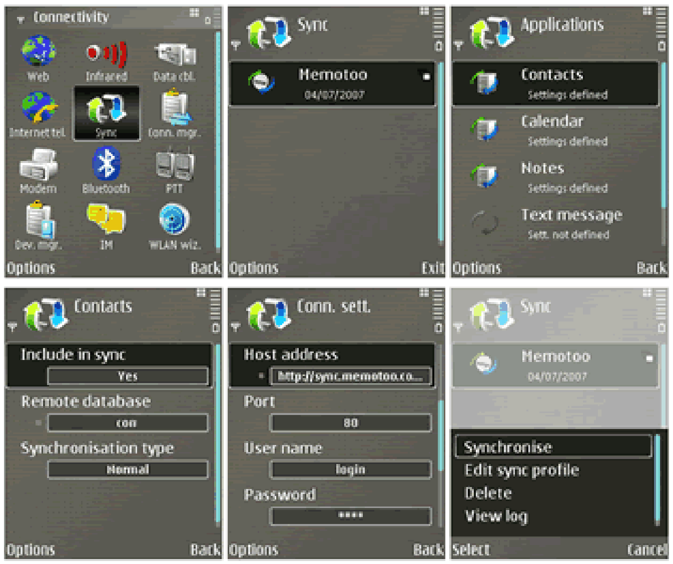
SyncML server:

- Has to work with the already existing database and database structure of Abakus
- Has to be modular and flexible
- Must be able to work under heavy traffic
- Has to be possible to optimize the server

## 1.3 Result of the prestudy

### 1.3.1 Own SyncML client or built-in SyncML client

The advantage of a built-in SyncML client (a phone is quite often said to be a SyncML supported despite the fact that there might not be a built-in client) is that there is a need to implement anything on the client side at all. These clients usually support the calendar (vCal format) and contact (vCard format) synchronization.



Include in sync: Yes
Remote database: con
Synchronisation type: Normal

Host address: http://sync.memotoo.co...
Port: 80
User name: login
Password: ****

FIGURE 1. Synchronization settings

### 1.3.2 Own SyncML server or already existing one

After analyzing several SyncML server solutions it was decided on the Funambol Data Synchronization Server (advantages later at Funambol). The built-in the client and the server use different calendar entry formats. The vCal format has to be converted to aCal format and back.

## 1.4 Synchronization with different synchronization protocols

The IT industry offers different non-interoperable data synchronization products. From these each works only with certain transports and is implemented on a few platforms. Mostly these products use distinct data synchronization protocols.

The proliferation of non-interoperable synchronization technologies complicates the life of users, device, manufacturers, service providers and application developers. In addition, the lack of a common data synchronization protocol limits the growth in use of mobile devices and the delivery of mobile data services.

## 1.5 Advantages of a common synchronization protocol

- End users: Usually the synchronization products vary from device to device. Hence it becomes hard to install, configure and operate these applications. However, SyncML allows to take devices that synchronize with a broader range of data.
- Device manufacturers:  Costly to support more than one type of data synchronization technology for the sake of the constraints of device storage capacity, power consumption and cost. They take advantage of using a common protocol that makes the device interoperable with a wider range of applications, services as well network and transmission technologies.
- Service providers: This is an intensively growing field because more and more client uses SyncML and their multiple server infrastructures with maintaining. With SyncML they are able to provide a wider and wider set of applications.
- Application developers: Instead of choosing multiple synchronization technologies if the developer votes to SyncML, it allows to develop one application that can connect to a more diverse group of devices and networked data.

(Mahmoud, 2004)

# 2 SYNCML

SyncML (Synchronization Markup Language) is the formal name for the platform-independent information synchronization standard. The project is now referred to as Open Mobile Alliance Data Synchronization and Device Management (OMA DS and DM). The SyncML standard works via the SyncML representation protocol and the SyncML synchronization protocol.

The SyncML representation protocol deals with organizing the data contents of synchronization and defines the methods for naming and identifying records as well as the XML document type used to represent a SyncML message, such as common protocol commands (Add, Alert, Atomic, Copy, Delete, Exec, Get, Map, Replace, Search, Sequence, Sync) and message containers. The SyncML messages might be transmitted over the wireless network using HTTP, WSP (Wireless Session Protocol), or OBEX (Object Exchange Protocol).

Clients request data from the server and store it on the mobile devices where they can manipulate as local copy of the retrieved data. The process where updates are exchanged and conflicts are resolved is known as data synchronization.

The data source on the server side can be a SQL Relational Database and on the mobile client side a local database according to the platform type (S60, RIM, iPhone). These client and server databases differ from each other for the sake of the capacity, thus for instance there is, LUID (local unique ID) on the client side and GUID (global unique ID) on the server side as well as a mapping table between the two types of ID.

A data synchronization protocol defines the work-flow for communication during the data synchronization session when the mobile device is connected to the network. The protocol must support naming and identification of records, common protocol commands as well as identification and resolution of conflicts.

(Hansmann, 2002, 26-33)

## 2.1 SyncML message structure

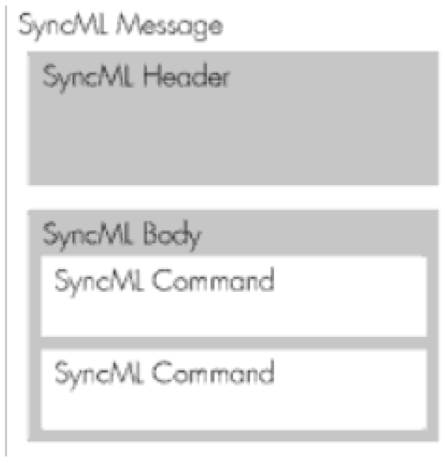


FIGURE 2. SyncML message structure

The SyncML Representation Protocol provides frame for the logical structure and format of various SyncML Messages. Every message is a well-formed XML Document with exact definition which is written in a DTD (XML Document Type Definition).

In the synchronization process the synchronizing entities logically exchange packages but physically these packages can be divided into multiple actual communicating messages. There might be many instances where the logical package is broken down into messages.

The main motivation of this approach is generally the mobile capabilities and the connection brand-width as well as the small size messages can arrive more successfully.

## 2.2 SyncML in HTTP request

As a transport layer the HTTP protocol is used in the data synchronization between the SyncML client and the SyncML server so the SyncML messages are in the body of HTTP request and response.

The post method is used to transfer the SyncML message into the HTTP request and the following information is needed to specify in the HTTP header:

- Accept:  defines the allowed MIME type in the response.
- Accept-Charset: specifies the allowed character set in the response.
- Authorization: helps the HTTP server to authorize the HTTP client.
- Cache-Control: handles the controlling between the HTTP client and HTTP server in the request/response chain.
- User-Agent: identifies the type of user agent originating the request.

(W3, 2004)

## 2.3 WBXML (WAP Binary XML)

This binary XML content format is developed by the WAP Forum and the main motivation was to reduce the transmission size of XML.

The sent SyncML message from the mobile device is encoded in WBXML. The XML needs to be converted to WBXML before responding. A deeper look is taken how it works in more details.

WBXML implements the equivalent of XML namespaces through code pages. Switching code pages equals to switching the default namespace. The WBXML encoding of SyncML uses a code page for each of the DTDs used in the protocol SyncML, SyncML Meta Information and SyncML Device Information.

The process tokenizing an XML document converts all markups of XML syntax into their corresponding tokenized format. The XML declaration, the document type declaration and all comments should be removed. Processing instructions intended for the tokenizer may be removed and all other processing instructions must be preserved. The provisioning XML is changed into WBXML by the PPG (Push Proxy Gateway) or Push Initiator.

The length of each document is one important consideration at the WBXML encoding of SyncML.

### 2.3.1 Java XML parsers

The XML parsers usually have heavy run-time memory usage. On the client side for the sake of limited capabilities it comes to the surface and requires attention from the developers. There are three main types of Java parsers:

1. The pull parser handles the document as a series of items which are read in sequence. This creates an iterator that sequentially checks the different parts of the document by repeatedly requesting the next piece. With the iterator it is possible to test the current item and inspect its attributes. From the point of visibility the pull-parsing code might be easier to maintain than SAX parsing code. The pull parsers retrieve the data in case they are requested to read the next node in the document. For example: StAX (Stream in API for XML) calls the next() method of parser iteratively and each call returns the next XML construct. The kXML parser belongs to this category.

2. A model parser completely reads the XML document in the memory and creates there a representation. Obviously, it requires more memory usage than the others.

3. A push parser pushes parsing events to the application. A SAX (Simple API for XML) defines several callbacks that are called by the parser, when events occur. SAX goes through the entire document once from the beginning to the end and sends every occurred events to the calling application.

(Gosh, 2003)

#### kXML parser

This is a pull-based parser designed to can run in embedded systems such as personal mobile devices but in this case it is used on the server.

Convert WBXML to XML and back needs a lot of tweaks for SyncML. The Funambol uses kXML 1.0 parser only it got plenty of SyncML additions.

## 2.4 Many-to-one

The many-to-one topology (central master) is applied at synchronization because the data is propagated from the central master (server) to different entities (clients). All of them has own local database.

The client does not need to determine where to send the requests because there is only one central master unlike at many-to-many topology.

Conflicts can only occur at the central server, which needs to detect and resolve them and the clients do not deal with resolving conflicts. The clients send information to the central master about the local modifications and process the change requests received from the server.

(Hansmann, 2002, 5.)

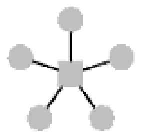


FIGURE 3. Many-to-one topology

## 2.5 How does it work in brief?

### 2.5.1 Way of the request

The client sends a request from the Sync Client Application of the Nokia device to the server. This request goes through the Sync Client Agent of the mobile device. After this point the request contains a header and a WBXML encoded SyncML message.  This request is forwarded to the Sync Server Agent and then to the Sync Server Application and Engine.

### 2.5.2 Way of the response

After completing the request on the server, it has to create an XML regarding the SyncML syntax then decode it to WBXML and response to the client. It ends if everything is okay, then the SyncML client processes and completes the defined commands in its local database.

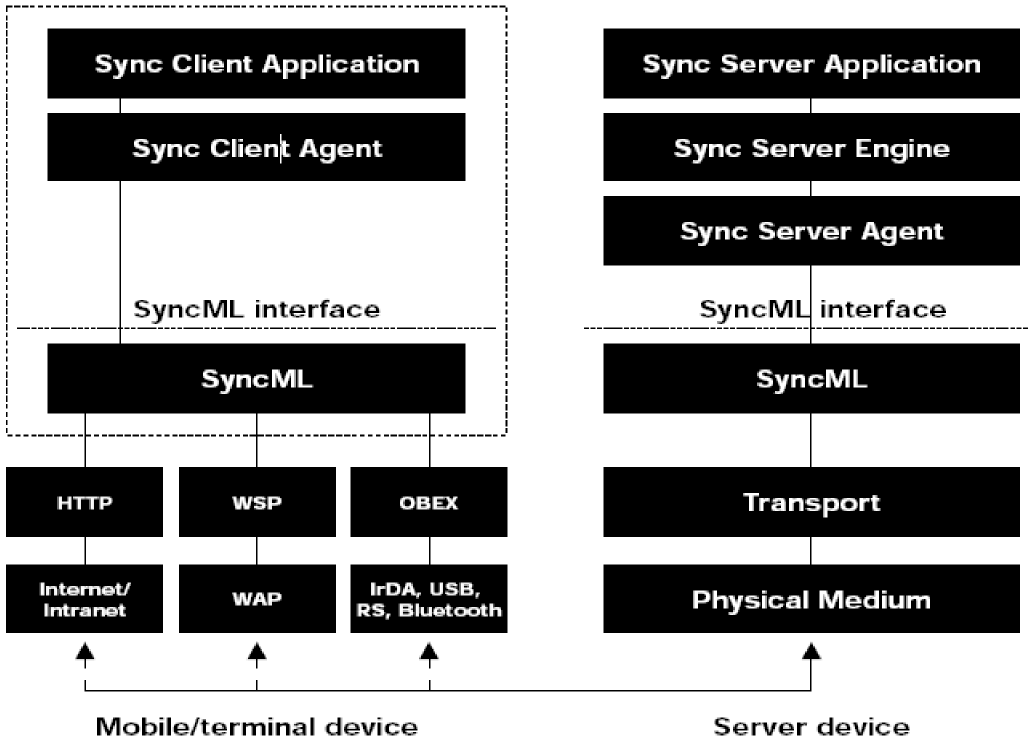


FIGURE 4. Execution flow during the synchronization session

## 2.6 Device roles

SyncML client: This contains a sync client agent which deals with sending and retrieving the data from the SyncML server. In this case it is a Nokia E71 phone with a built-in SyncML client.

SyncML server: This contains a sync server engine and the sync agent. Generally, it just waits for the client requests and after analyzing and communicating with the database, this generates the response.

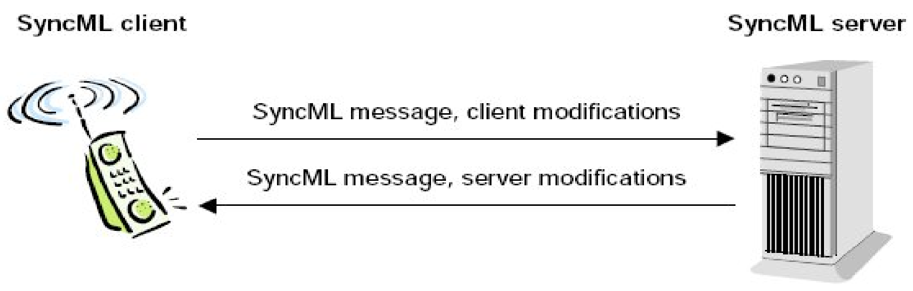

SyncML client
SyncML server
SyncML message, client modifications
SyncML message, server modifications

FIGURE 5. Synchronization between mobile device and server

## 2.7 Synchronization Initialization

Before synchronization package sending it is necessary to make a client and server initialization as shown in figure 6 below. The initialization defines exactly:

- Which database as well as protocol type on the client and on the server is desired to be used. If there are authentication credentials, the initialization helps to process on the SyncML level. Basically, this is done by using the Alert command of the SyncML Representation protocol. These must be supported by the client and the server.
- The exchange information about service and device capabilities. It is done by using the Put and Get commands of the SyncML Representation protocol and the Device Information DTD.

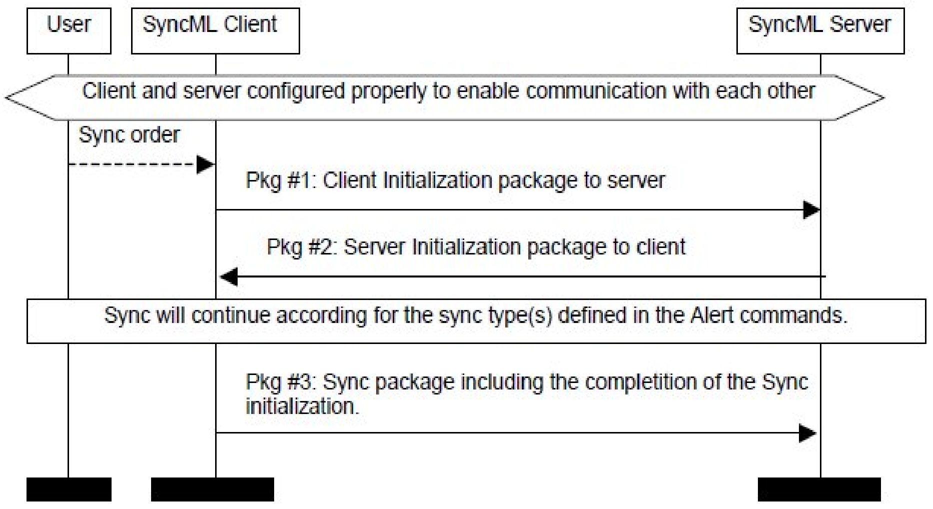
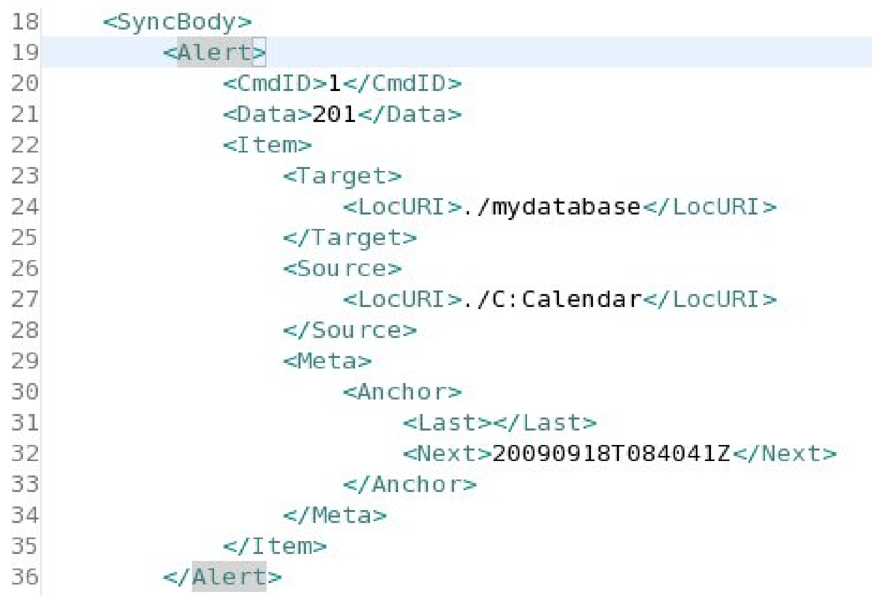

User
SyncML Client
SyncML Server
Client and server configured properly to enable communication with each other
Sync order
Pkg #1: Client Initialization package to server
Pkg #2: Server Initialization package to client
Sync will continue according for the sync type(s) defined in the Alert commands.
Pkg #3: Sync package including the completition of the Sync initialization.

FIGURE 6 Client initialization

There is a header part of the Sync Initialization request on example 7 where:

- The VerDTD tag defines the exact type of the DTD.
- The VerProto was choosable on the Nokia client, now it is 1.1.
- The SessionID must be constant during the whole session.
- One package can contain more than one message, hence these have to have MsgID (message ID).
- The Target tag consists of information about the address of the web server.
- The Source tag contains information about the IMEI of the mobile device.

(SyncML Representation Protocol, 2000, 26-33)



```
    <SyncBody>
        <Alert>
            <CmdID>1</CmdID>
            <Data>201</Data>
            <Item>
                <Target>
                    <LocURI>./mydatabase</LocURI>
                </Target>
                <Source>
                    <LocURI>./C:Calendar</LocURI>
                </Source>
                <Meta>
                    <Anchor>
                        <Last></Last>
                        <Next>20090918T084041Z</Next>
                    </Anchor>
                </Meta>
            </Item>
        </Alert>
```

EXAMPLE 1. Sync initialization request header

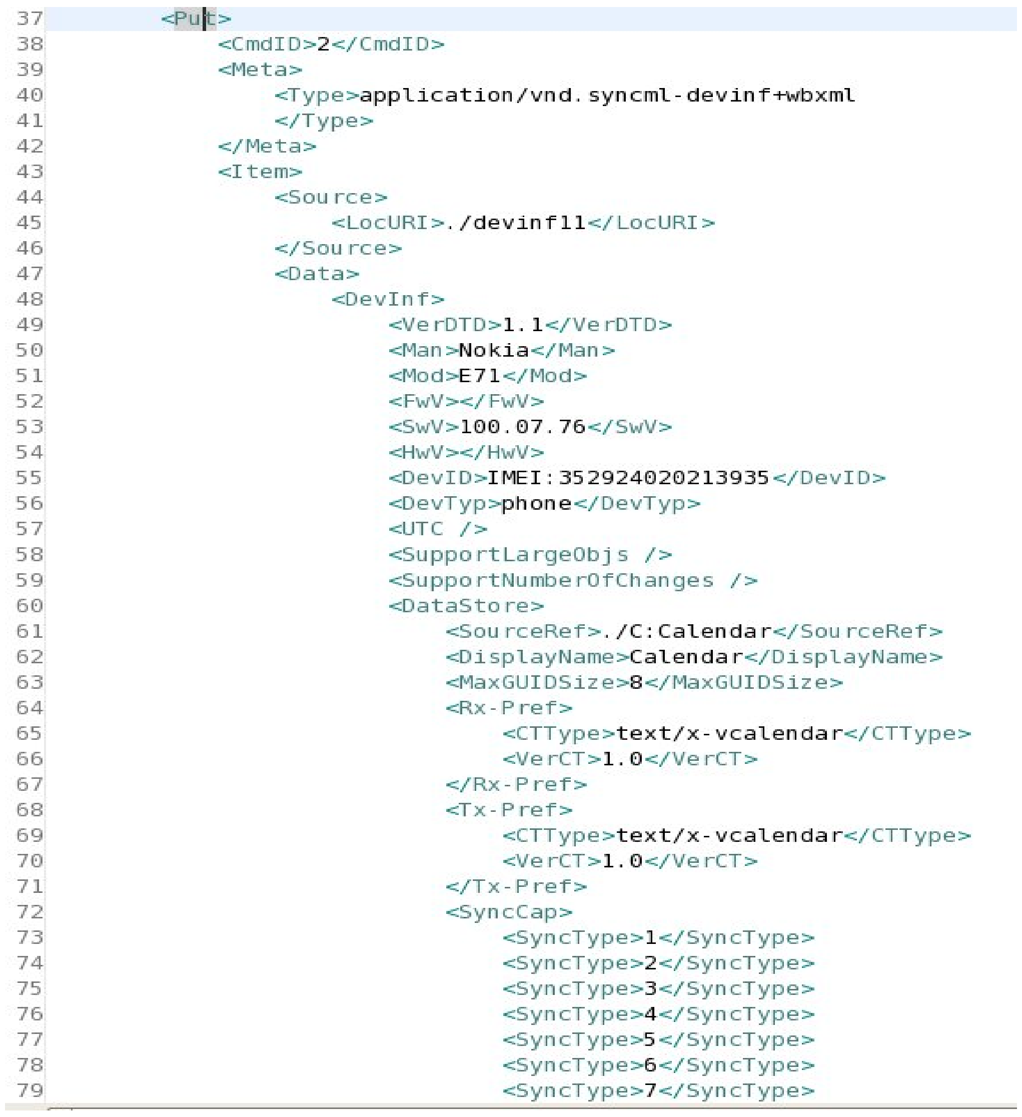
The databases desired to be synchronized are indicated in the separate Alert command. The latter helps to exchange synchronization anchors.

There are two types of anchors (Last, Next) where the Last synchronization anchor gives information about the last event when the database was synchronized from the point of the sending device. The Next anchor provides info about the current event from the point of the sending device. The receiving device has to echo the Next synchronization anchor back to the transmitting device in the Status of for the

Alert command. The Next anchor contains a time where the first part before the T (time) letter refers to the date and after it there is a time; finally, the Z letter expresses Zulu time.

If the device assures storage for the Next synchronization anchor, it can compare during the next synchronization whether the sync anchor equals with the Last synchronization anchor, and in case they are the same, the device can conclude that no errors have occurred since last synchronization.

It is important that the synchronization anchors can be updated just after finishing this synchronization session.



```
        <Put>
            <CmdID>2</CmdID>
            <Meta>
                <Type>application/vnd.syncml-devinf+wbxml
                </Type>
            </Meta>
            <Item>
                <Source>
                    <LocURI>./devinf11</LocURI>
                </Source>
                <Data>
                    <DevInf>
                        <VerDTD>1.1</VerDTD>
                        <Man>Nokia</Man>
                        <Mod>E71</Mod>
                        <FwV></FwV>
                        <SwV>100.07.76</SwV>
                        <HwV></HwV>
                        <DevID>IMEI:352924020213935</DevID>
                        <DevTyp>phone</DevTyp>
                        <UTC />
                        <SupportLargeObjs />
                        <SupportNumberOfChanges />
                        <DataStore>
                            <SourceRef>./C:Calendar</SourceRef>
                            <DisplayName>Calendar</DisplayName>
                            <MaxGUIDSize>8</MaxGUIDSize>
                            <Rx-Pref>
                                <CTType>text/x-vcalendar</CTType>
                                <VerCT>1.0</VerCT>
                            </Rx-Pref>
                            <Tx-Pref>
                                <CTType>text/x-vcalendar</CTType>
                                <VerCT>1.0</VerCT>
                            </Tx-Pref>
                            <SyncCap>
                                <SyncType>1</SyncType>
                                <SyncType>2</SyncType>
                                <SyncType>3</SyncType>
                                <SyncType>4</SyncType>
                                <SyncType>5</SyncType>
                                <SyncType>6</SyncType>
                                <SyncType>7</SyncType>
```

EXAMPLE 2. Put command of SyncML message

The Put command helps to send service capabilities to the server and the server can also ask the service about its capabilities by the Get command.

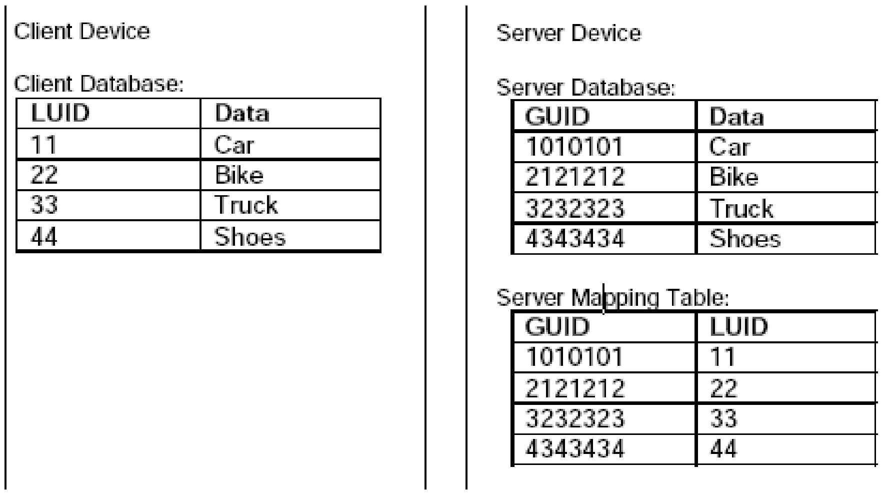
The DevInf refers to the Device Information as device manufacturer, model, IMEI and type.

The MaxGUIDSize specifies the maximum size of a global unique identifier for the ./C:Calendar in bytes for the device to able to retrieve and store it.

The SyncML allows the client and the server to have their own IDs for database items in their own local databases. These IDs on the two sides can differ from each other, thus the server has to maintain an ID mapping table for the items.

When the server adds a new item to the client, it must not send its actual GUID if the size of the actual GUID is more than 8 bytes. However, the server must use a suitable temporary GUID when adding an item to the client.

If the server had modified an existing item by the GUID after, it has to identify the item by LUID and use the mapping table. The same way when the client modifies something the server has to map the LUID to the GUID by utilizing the mapping table.



Client Device

Client Database:

| LUID | Data |
|---|---|
| 11 | Car |
| 22 | Bike |
| 33 | Truck |
| 44 | Shoes |

Server Device

Server Database:

| GUID | Data |
|---|---|
| 1010101 | Car |
| 2121212 | Bike |
| 3232323 | Truck |
| 4343434 | Shoes |

Server Mapping Table:

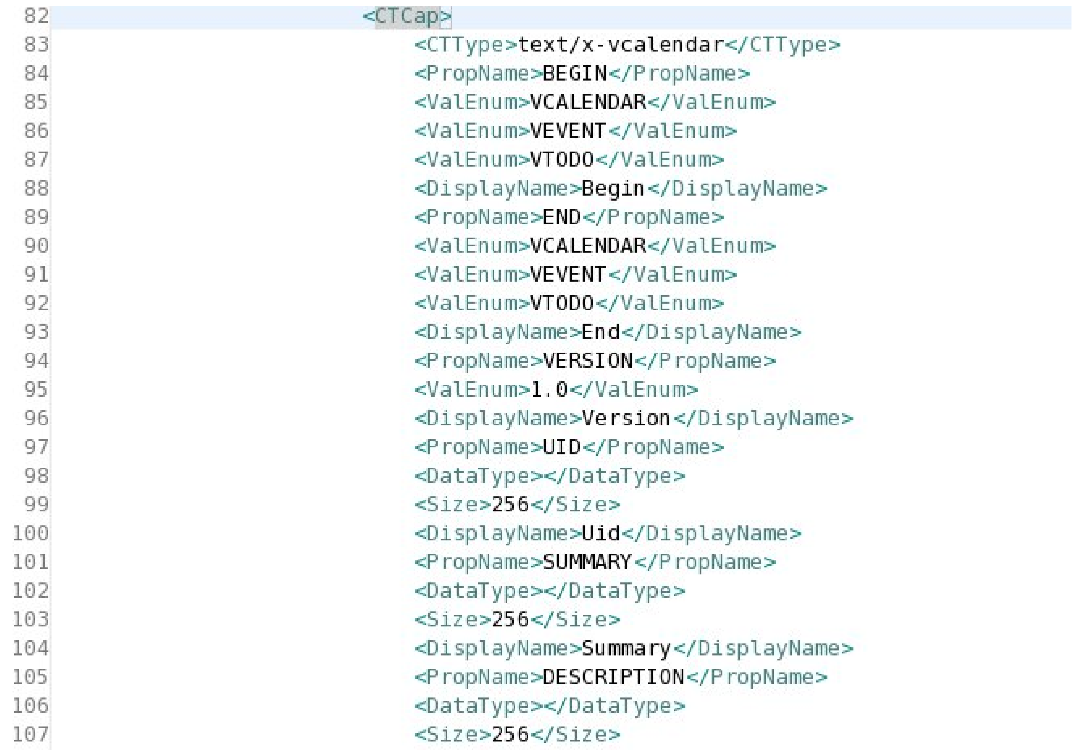
| GUID | LUID |
|---|---|
| 1010101 | 11 |
| 2121212 | 22 |
| 3232323 | 33 |
| 4343434 | 44 |

FIGURE 7. ID mappings of Data items

The Rx-Pref specifies the type text/x-vcalendar version 1.0 of a content type received by the device.

The Tx-Pref also specifies text/x-vcalendar version 1.0 of a content type transmitted by the device.

The SyncCap defines the synchronization capabilities of this device. It contains synchronization types what were already defined above and this device supports all of them.



```
<CTCap>
    <CTType>text/x-vcalendar</CTType>
    <PropName>BEGIN</PropName>
    <ValEnum>VCALENDAR</ValEnum>
    <ValEnum>VEVENT</ValEnum>
    <ValEnum>VTODO</ValEnum>
    <DisplayName>Begin</DisplayName>
    <PropName>END</PropName>
    <ValEnum>VCALENDAR</ValEnum>
    <ValEnum>VEVENT</ValEnum>
    <ValEnum>VTODO</ValEnum>
    <DisplayName>End</DisplayName>
    <PropName>VERSION</PropName>
    <ValEnum>1.0</ValEnum>
    <DisplayName>Version</DisplayName>
    <PropName>UID</PropName>
    <DataType></DataType>
    <Size>256</Size>
    <DisplayName>Uid</DisplayName>
    <PropName>SUMMARY</PropName>
    <DataType></DataType>
    <Size>256</Size>
    <DisplayName>Summary</DisplayName>
    <PropName>DESCRIPTION</PropName>
    <DataType></DataType>
    <Size>256</Size>
```

EXAMPLE 3. Calendar attributes

The CTCap specifies the content type capabilities of the device. Its children elements are the PropName and ValuEnum, DispayName, DataType and size.

The PropName has to be one from the following: text/x-vcard, text/vcard, text/x-vcalendar or text/calendar.

The ValEnum defines the supported enumerated value of a given text/x-vcalendar type property.

The size tag specifies the 256 byte length of the property or parameter.

## 2.8 Synchronization Types

There are seven types of SyncML synchronization with alert codes:

1 Two-way synchronization (200): where the client sends modification information to the server and this returns with the response. It is a normal synchronization type.

2 Slow synchronization (201): in this case every items of the database is compared step by step. The client just sends all of the data to the server and that analyzes it field by field.

3 One-way from client only (202): the client sends modifications to the server but the server does not.

4 Refresh from client only (203): the client sends all its data from the database to the server. The server may replace all data in the server database.

5 One-way from server only (204): the server sends all its modifications from the database to the client. Then the client does not response with its modifications.

6 Refresh from server only (205): The server sends all its data from the database to the client and the client might replace all data in its local database.

7 Server alerted: it refers to the server which alerts the client to do synchronization. Beforehand the server notifies the client to begin a specific type of synchronization with the server.

(Mahmoud, 2004)

### 2.8.1 Slow synchronization

A simple way to reach consistent data is to send all information from the client to the server and at the same time allow the server check all of the records with its own records field by field in order to prevent the duplicates. After this process the server possesses every record as well as the new records from the client device. Finally, these records are sent to the client, which checks them against its own records. According to the differences between the two local databases, (client, server) it modifies, adds or deletes records. For the sake of many reasons the processing of slow sync can be started on the client or the server side which indicates the need for this. In this case the client sends the alert already in the initialization phase and it is specified with alert code 201 (see alert codes above). This form of synchronization is not a efficient way to synchronization but it is needed when the server and the client have lost track of when the last successful synchronization was as well as in the initialization phase when they do not know each other's data.

### 2.8.2 Two-way synchronization

Two-way synchronization is a normal synchronization where the two sides must exchange information about the modified dates in these devices. Every time the client sends the data (synchronization request), then this is modified first with the data in the server. After all, the client gets a response and according to the content it might or might not update the database of the mobile device.

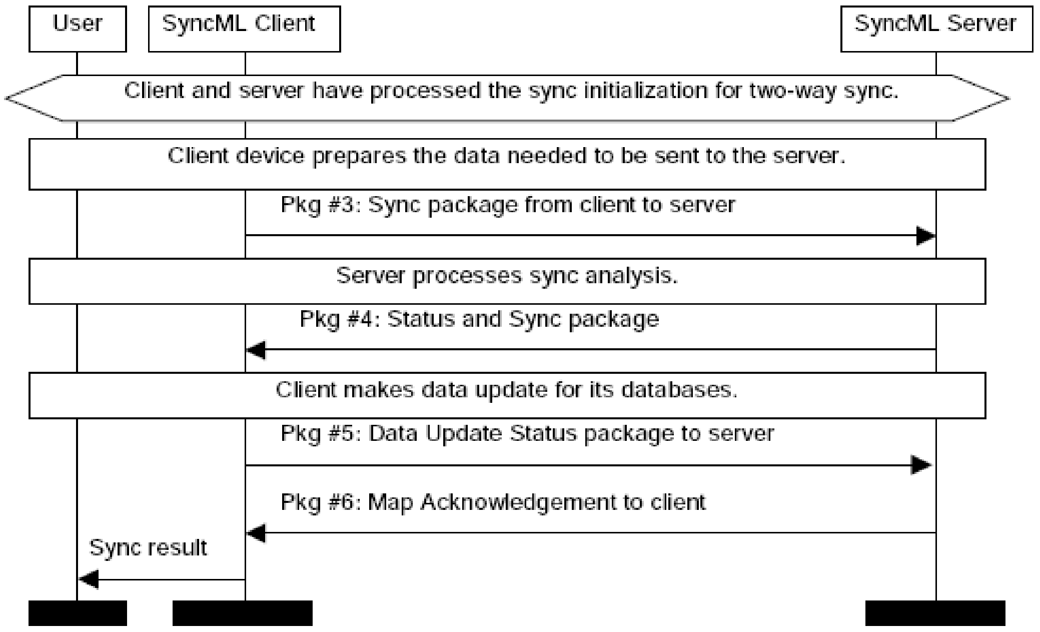

User
SyncML Client
SyncML Server
Client and server have processed the sync initialization for two-way sync.
Client device prepares the data needed to be sent to the server.
Pkg #3: Sync package from client to server
Server processes sync analysis.
Pkg #4: Status and Sync package
Client makes data update for its databases.
Pkg #5: Data Update Status package to server
Pkg #6: Map Acknowledgement to client
Sync result

FIGURE 8. Two-way synchronization

The arrows describe SyncML packages which can contain one or more messages. Every package possesses the session ID.

## 2.9 End of synchronization

The synchronization session is finished after the device is not going to send or receive anymore SyncML messages from the device and the synchronization was successful on the Sync command level as well as directly under the SyncML level, where the transport level has to be ended error freely.

## 2.10 Conflict

### 2.10.1 Conflict detection

Conflict detection is important in order to keep different data stores consistent. One relevant requirement of being able to synchronize data is to have uniquely identified records.

Conflict detection is solved by primary keys at the Relational Databases and by UID at PIMs. Each local database has to have own UIDs to identify, and, as mentioned above, they are called LUIDs (Local Unique Identifier).

The synchronization session generally begins with the client sending a list of changed records since the last synchronization, after which the server generates a list of all modifications. Finally, the server compares the two existing maps LUID/GUID (discussed previously) and detects conflicts.

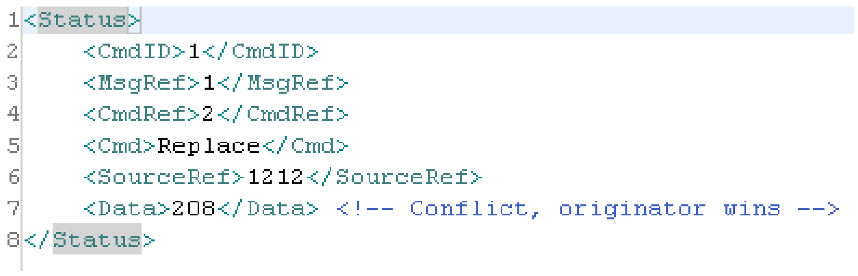
### 2.10.2 Conflict resolution

The conflict resolution is the certain action that the server has to make in order to resolve the detected conflicts. The SyncML does not define how conflicts should be resolved, instead, it provides a frame for reporting conflicts and actions taken. There are:

- Status codes for different types of conflicts (delete, update)
- Status codes for outcomes (server or client win, merge, copy)

The server deals with resolving the conflicts and does not put plus weight onto the client although the SyncML protocol would allow this. There are several already existing policies how to fix the conflicts and SyncML supports the commons with alert codes and helps to tell how it was resolved.

(Hansmann, 2002, 15)



```
<Status>
    <CmdID>1</CmdID>
    <MsgRef>1</MsgRef>
    <CmdRef>2</CmdRef>
    <Cmd>Replace</Cmd>
    <SourceRef>1212</SourceRef>
    <Data>208</Data> <!-- Conflict, originator wins -->
</Status>
```

EXAMPLE 4. SyncML status message

# 3 FUNAMBOL

Funambol is an Open Source Mobile Application Platform which includes SyncML Data Synchonization and SyncML Device Management solutions.

Funambol provides, for example, an address book and calendar for data synchronization and device management for wireless devices. The project started around 2001 with, the name Sync4j and the main goal was to find solution to a synchronization of calendar, contact and other PIM exercises.Funambol builds up a whole framework around the SyncML which includes the following interesting components (Funambol DS Server Architecture and Design Document, 2009):

- Funambol Data Synchronization Server: a mobile application server providing synchronization services for wireless clients and PCs.Funambol Device management: an Open Mobile Alliance Data Management server that remotely provides management for mobile devices.
- Funambol Module: refers to a container which is related to a Funambol DS server extension. There is a module which contains classes, configuration file, server beans which give access to a concrete database for data synchronization.
- Funambol Connector: it provides support for data synchronization with specific data sources. Gateways to file systems, databases, email systems and applications for two-way synchronization with existing data assets.
- Funambol Client: client software applications that enable users to synchronize email and PIM data (contacts, calendar, tasks and notes) between a wide range of mobile devices and the Funambol server. This is what we do not use for the sake of built-in client.
- Funambol Software Development Kit (SDK): a suite of tools to develop sometimes-connected mobile applications on devices in Java ME and Java SE

and C++, and to add data sources to the server.

Before Sync4j (Funambol), there was no existing open source Java implementation for mobile device data synchronization so a person who did not pay for an expensive software had to use already existing solutions like OpenSync that was implemented in C or decide to develop their own SyncML.

In the research phase there was a period when their JNI (Java Native Interface) and the OpenSync API were considered to be used. Later the Java approach was needed because Abakus is an Java oriented company.

The Funambol offers enough comfortable and flexible API to put its data source, business logic and specific needs for not develop again a new SyncML server with request, response handling and sync logic.

*(Learn About Funambol, 2008)*

## 3.1 Architecture

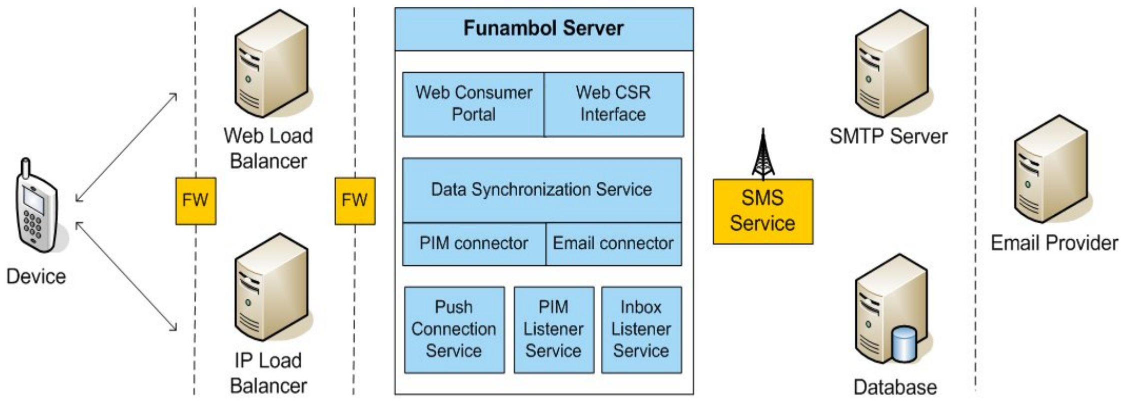


FIGURE 9. Funambol Architecture (Funambol, 2009, 19.)

A device can be on Figure 13 any physical device or client software that can communicate with Funambol Server via SyncML based on the TCP/IP protocol. Such devices could be:

- Desktop devices with working Funambol Client. In the testing period Mozilla Thunderbird's Funambol client was used because it makes debugging easier regarding the well detailed logging functionality.

- Mobile phones with a native SyncML client. The built-in Nokia client belongs to this group and it was mentioned that the main goal of the project was to realize the connection with this.
- Mobile phones with Java ME support could meet the requirements. The Funambol Mobile Client belongs to this category.

All of the devices should contain the following responsibilities:

- Providing the user UI
- Initiating the communication with the server
- Hosting the local data, which means: the client in connection with the local data source and be  able to gain and send data.
- Collecting and/or detecting the change log

## 3.2 Main parts of the Funambol Server

### 3.2.1 Web load balancer

Load balancing is a technique to distribute workload evenly across two or more computers in order to get optimal resource utilization, maximize throughput, minimize response time and avoid overload. The load balancing as a service is usually given by a dedicated program or hardware device such as a multilayer switch or a DNS server. And the web load balancer controls the incoming load amongst different nodes of the Data Synchronization Service cluster. All the nodes of the cluster can be used for each SyncML request.

### 3.2.2 Data Synchronization Service

The main features are as follows below:

- Getting and serving synchronization requests
- Synclet technology
- Hosting the synchronization engine

- Treating low level device information
- Providing interface to the back-end, remote administration interface and connection-less push

#### 3.2.2.1 Synclet

The input Synclet is a class which implements the InputMessageProcessor interface together with the method named preProcessMessage (MessageProcessingContext context, SyncML msg) where the first class allows the pipeline components to share session-scoped and request-scoped properties and the second represents the SyncML class.

The output Synclet as a class implements the postProcessMessage(MessageProcessingContextcontext, SyncML msg) method with the same parameters and functions. The concepts behind the output message processing are the same as per input message processing. The input processor component, the so called input Synclet and an output processor component, the so called output Synclet are created to help to manipulate the vCal in both ways.

(Funambol, 2009, 46-50)

### 3.2.3 The synchronization engine

This is the core element of the synchronization server, and it provides functionality for the SyncML protocol on the application level. The engine can handle three phases: initialization, data exchange and finalization. It is in charge of the followings:

- Determines what and with what should be synchronized
- Handles conflicts
- Does ID mapping (GUID and LUID)
- Detects change detections

The execution flow (see on figure 10) consists of six steps:

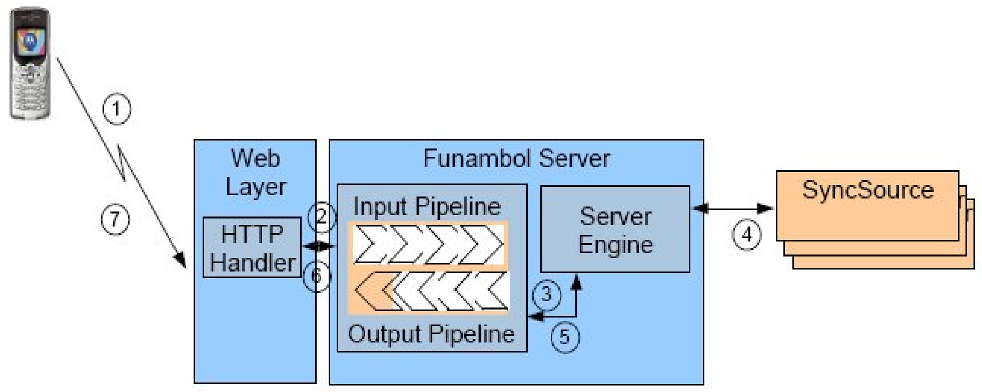


FIGURE 10. Execution flow (Funambol, 2009, 22.)

The synchronization session starts with the first SyncML message sent by the device. As this message arrives packaged in a HTML request, the HTTP handler's task is to process it right away. At this part it is important to transfer the message inside the HTTP request to a process in appropriate form.

1 The unpacked, cleaned message goes through the input message processing pipeline. Here the vCal format XML will be converted into the Abakus specific calendar format by the Synclet.

2 The changed SyncML message goes to the server engine for the sake of synchronization processing.

3 This is the place of synchronization process and in this case it requires custom SyncSources in order to reach its own Abakus data source.

4 After the result of synchronization process, the response has to meet the requirements of the built-in mobile client (calendar entry in vCal format).

5 The response SyncML message returns packaged to HTTP request by the HTTP handler and sends it to the client through the HTTP protocol.

```
<SyncML>
    <SyncHdr>
        <VerDTD>1.1</VerDTD>
        <VerProto>SyncML/1.1</VerProto>
        <SessionID>2</SessionID>
        <MsgID>1</MsgID>
        ...
        ...
        ...
        ...
</SyncML>
```

EXAMPLE 5. SyncML message in XML

The core of the Funambol in step two (in the execution flow) uses SyncML class, which is the object oriented representation of the SyncML message. Here is a hierarchical view of the message:

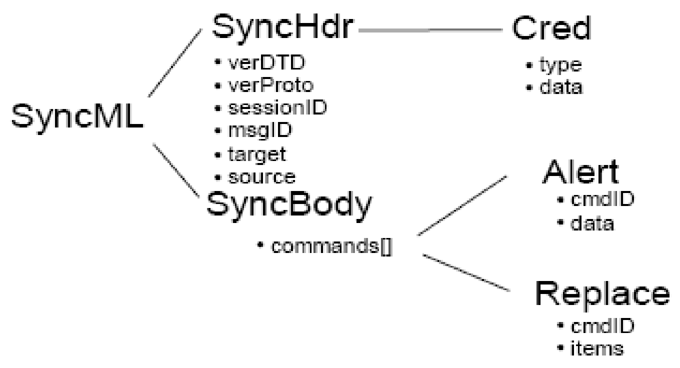


FIGURE 11. Message as a SyncML object (Funambol, 2009, 46.)

### 3.2.3.1 SyncSource

The SyncSource is the access to a data store that the mobile device will be synchronized towards. It is a very important component and offers most popular uses such as vContact, vCal, database, file system.

A SyncSource has a lifecycle and states. The lifecycle is defined according to the synchronization lifetime. Between the SyncSource states there are transitions:

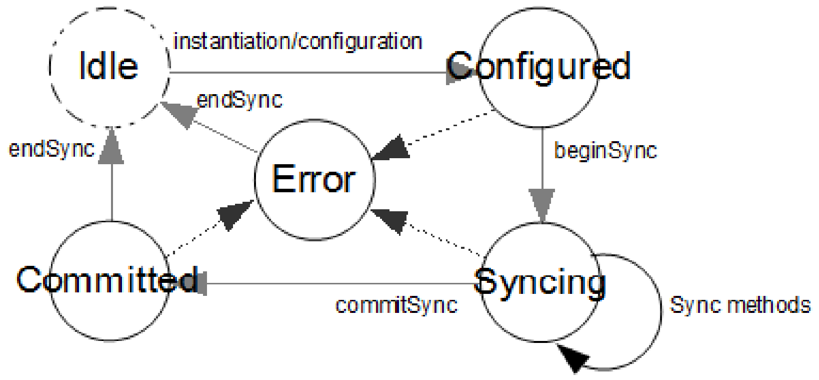


FIGURE 12. States of SyncSource

- Idle: this state of SyncSource is situated on the configuration layer as a registered SyncSource but it does not get into the memory yet, and this is why it is said to be a not real state.
- Configured: from the point when the SyncSource becomes instantiated, it is in configured state and prepared to be used by the synchronization engine. The invoking of the method beginSync() starts stepping to the state Syncing.
- Syncing: the SyncSource selects the items according to their status. The synchronization process start and in case there is no fatal error, the engine invokes the method commitSync() in order to commit the changes. If the committing is successful, the state will change to committed.
- Committed: the method endSync() is called which handles the finalization tasks. Then the latter method causes the SyncSource to turn into the idle state.
- Error: the engine will turn the SyncSource into error state if a fatal error occurs and will be reported to the client.

Funambol defines SyncSource and does not tell anything about the type of data to be synchronized. An unique sourceURI and domain-specific name identify the SyncSource.

There are two type of SyncSource interface and abstract class:

- MergeableSyncSource <<interface>>: In case a conflict raises it can be handled by merging conflict resolution strategy. For example the mobile client user updates the todo list and on the server the same todo list is updated. The merging strategy avoids the loss of information hence the two data will be merged.
- FilterableSyncSource <<interface>>: since the filter is defined in SyncML 1.2 the SyncSource supports it. There is a Record filter which says exactly which records should be synchronized and the Field filter defines the roles on the fields synchronized.

Moreover, it is possible specified if the filter to use must be Exclusive or Inclusive. In case the of Exclusive filter, the server needs to remove all items on the client not included in the filter. In Inclusive case the client has to remove all of its items not included in the filter and inform the server about delete command.

## 3.3 Implementation of the Connector

Funambol's container named module but when a module provides access to a specific back-end it is called Connector. There are needed SyncSources, Synclets and classes to the back-end which have to be implemented. There is the class diagram of the Connector see on figure 13.

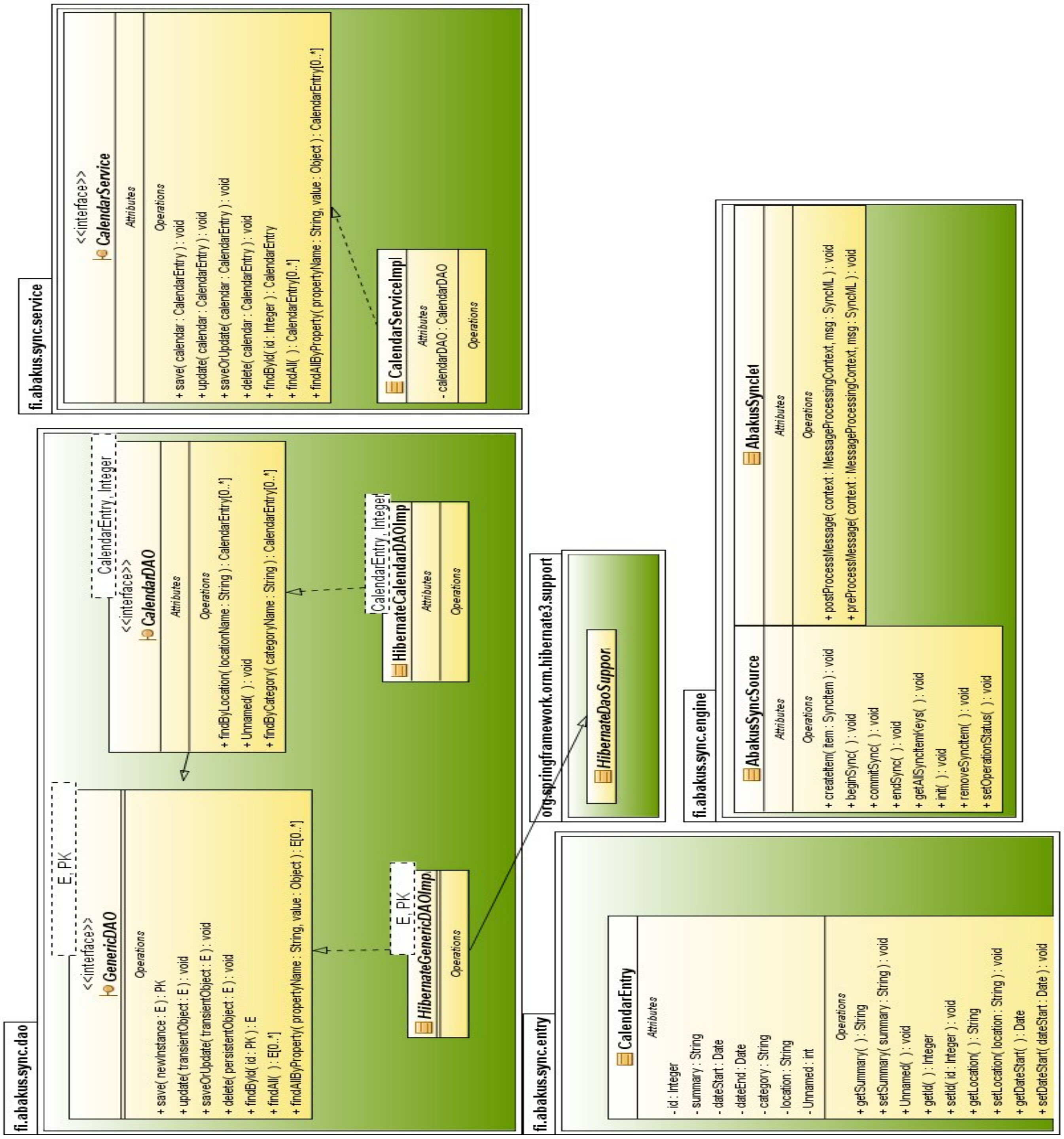

fi.abakus.sync.service
<<interface>>
CalendarService
Attributes
Operations
+ save( calendar : CalendarEntry ) : void
+ update( calendar : CalendarEntry ) : void
+ saveOrUpdate( calendar : CalendarEntry ) : void
+ delete( calendar : CalendarEntry ) : void
+ findById( id : Integer ) : CalendarEntry
+ findAll( ) : CalendarEntry[0..*]
+ findAllByProperty( propertyName : String, value : Object ) : CalendarEntry[0..*]
CalendarServiceImpl
Attributes
- calendarDAO : CalendarDAO
Operations
fi.abakus.sync.dao
CalendarEntry, Integer
<<interface>>
CalendarDAO
Attributes
Operations
+ findByLocation( locationName : String ) : CalendarEntry[0..*]
+ Unnamed( ) : void
+ findByCategory( categoryName : String ) : CalendarEntry[0..*]
CalendarEntry, Integer
HibernateCalendarDAOImp
Attributes
Operations
E, PK
<<interface>>
GenericDAO
Operations
+ save( newInstance : E ) : PK
+ update( transientObject : E ) : void
+ saveOrUpdate( transientObject : E ) : void
+ delete( persistentObject : E ) : void
+ findById( id : PK ) : E
+ findAll( ) : E[0..*]
+ findAllByProperty( propertyName : String, value : Object ) : E[0..*]
E, PK
HibernateGenericDAOImp
Operations
org.springframework.orm.hibernate3.support
HibernateDaoSuppor
fi.abakus.sync.engine
AbakusSynclet
Attributes
Operations
+ postProcessMessage( context : MessageProcessingContext, msg : SyncML ) : void
+ preProcessMessage( context : MessageProcessingContext, msg : SyncML ) : void
AbakusSyncSource
Attributes
Operations
+ createItem( item : SyncItem ) : void
+ beginSync( ) : void
+ commitSync( ) : void
+ endSync( ) : void
+ getAllSyncItemKeys( ) : void
+ init( ) : void
+ removeSyncItem( ) : void
+ setOperationStatus( ) : void
fi.abakus.sync.entry
CalendarEntry
Attributes
- id : Integer
- summary : String
- dateStart : Date
- dateEnd : Date
- category : String
- location : String
- Unnamed : int
Operations
+ getSummary( ) : String
+ setSummary( summary : String ) : void
+ Unnamed( ) : void
+ getId( ) : Integer
+ setId( id : Integer ) : void
+ getLocation( ) : String
+ setLocation( location : String ) : void
+ getDateStart( ) : Date
+ setDateStart( dateStart : Date ) : void

FIGURE 13. Class diagram

## 3.3.1 Persistence

There is a persistent store in Funambol and it is designed to be easily integrated with any type of persistence models but not bound to any persistence technology. There is a named persistent store module which helps to access the database for the operations that concern the operation of the server itself.

Basically there are three kinds of opportunities to realize the persistence:

1. Change source-code automatically
2. Change byte-code automatically
3. Do run-time reflection

Third type persistence framework Hibernate is used.

### 3.3.1.1 Layered architecture

Nowadays most of the business applications use object-oriented technology, hence it makes sense to organize classes by concern. The multilayer architecture often refers to a n-tier architecture where the presentation, the application processing and the data management are separated logically. The most popular use of multilayer architecture is the three-layer architecture.

- Presentation layer: the written code is in charge of presentation and navigation of page forms.This is what the end user will face.
- Business layer: generally would be responsible for implementing domain related business and system requirements of the user. The exact example of this layer changes widely between applications.
- Persistence layer: is a set of classes and components which deal with storing, modifying and retrieving data from data stores. This layer contains a model of the business domain entities.

In the n-tier architecture each layer communicates with only the layer directly below and specifies the function which it is responsible for. One of the main advantages is that each layer can be placed on physically different servers with only minor code changes. In addition it does no matter what each layer does inside it because the mechanic is hidden from others, and this allows changing one layer without recompiling or modifying others.

(*Java Persistence with Hibernate, 2007, 20-24*)

#### 3.3.1.2 Object/relational mapping

ORM is a programming technique which converts data between relational databases and object-oriented programming languages. As an effect it creates a virtual object database which is used within

object oriented programming language. ORM allows programmers to reach and manipulate objects without having to consider how they relate to their data sources. With the help of abstraction it manages the mapping details between a group of objects and existing relational databases or other data sources (XML repositories) and the same time it hides the changing details of related interfaces from the developers. Hibernate is a pure Java object-relational mapping and persistence framework.

The ORM is discussed below from four points of view:

- Performance: The hand-coded persistence might often be faster than automated persistence but it can be looked from another point of view. At any kind of given persistence task, many optimizations might be possible. Some are much easier to achieve with hand-coded SQL/JDBC for instance query hints. Hibernate makes it possible to do many more optimizations to be used all the time. Those people who have implemented their ORM software probably have more time to investigate performance optimizations than who did not. To sum it up, that from case to base (certain task with the database) which is the faster.
- Productivity: The Hibernate does a huge amount of work instead of the developer on the persistence layer. Therefore it allows concentrating on the

business problem, hence it reduces the development time.

- Maintainability: With hand-coded persistence an unavoidable stress exists between the relational representation and the object model implementing the domain. Changes to one always involve changes to the other. In Hibernate general object oriented things like inheritance, polymorphism, and abstraction are used and force the developer to keep implementing according to it in the business layer. Hence, it assists the code visibility which helps the maintainability. By automated object/relational persistence it even reduces the lines of code
- Vendor independence: The ORM abstracts your application away from the underlying database (together with dialect). In addition it assists in a situation where the developing takes place with a lightweight local database but deploy for production on a different database. It is usually much easier to develop a cross-platform application using ORM.

There are four levels defined for ORM quality:

1. Pure relational
2. Light object mapping
3. Medium object mapping
4. Full object mapping

At the pure relation ORM the application with the user interface is designed around the relational model and SQL-based relational operations.

At the light object mapping the entities are represented as classes that are mapped manually to the relational tables. The code is not displayed at the business logic using specific design patterns. This

approach is successful for applications which contain a smaller number of entities or with meta-data driven data models.

The medium object mapping referring to an application is designed around an object model. The SQL code is created in build time and the persistence mechanism; queries use object-oriented expression language.

The full object mapping supports sophisticated object modeling: composition, inheritance, polymorphism and persistence. Useful and efficient fetching, caching strategies are implemented to the

application. Several Java ORM tools have achieved this level of quality such as the Hibernate, Toplink.

(*Java Persistence with Hibernate, 2007, 24-31*)

### 3.3.2 DAO

The Data Access Object (DAO) design pattern provides an abstract interface to the database or just to the persistence layer as an object. The main point of this is to hide the complexity of database operations and keep the code as simple and visible as possible.

The mentioned persistence layer above is written completely by Funambol and uses JDBC calls, so there is a mapping between this layer and the real application layer. This isolation assists to separate the concerns of what data accesses the application needs in terms of domain-specific objects and data types. The DAO object should be responsible for creations, reads, updates and deletions called CRUD on the domain object. That fact it deals with handling transactions, session, connections is disbelief; these have to stay out of the DAO in order to keep the flexibility. When there are several persistence services in the application it becomes more worthy to use a certain DAO layer instead of repeating persistence methods at many places.

#### Generic DAO

The generic DAO is the type of DAO interface which are generic as well as this is the name one of my concrete interfaces.

This concentrates on the less tedious aspects of designing persistence model. With using generic the code becomes type safe so in the worst case warnings can be occurred in compile time instead of ClassCastExpection in run-time.

Now Hibernate is used but there might be come a time of change to use another persistence layer is used and by keeping all the persistence layer specific code behind the DAO pattern it could be switched out easily.

The domain-object-specific generic DAO (like HibernateGenericDAOImp) is created by implementing one of DAO interfaces and giving the domain specified domain object type with generic type parameters.

GenericDAO was created contains all the basic operations: CRUD (create, read, update, delete). It has two generic parameters: the first is an entity and the second is a primary key of a certain entity. So the interface contains method specifications with these parameters.

The CalendarDAO interface can inherit the GenericDAO 's operations, or different solution later at the CalendarDAO implements the GenericDAO and the CalendarDAO together. For this thesis, the first alternative was chosen.

At the DAO interface implementation the HibernateGenericDAOImpl abstract class was used which extends from the HibernateDAOSupport class of Spring framework.

Spring provides a kind of translation bridge between technology-specific exception and its own exception hierarchy. If an exception occurs the DAOSupport just wraps to the hierarchy. Spring is a great deal to avoid writing all those factories and “glue”.

The HibernateDAOSupport class is an abstract super class for the HibernateGenericDAOImpl which requires a SessionFactory to be provided. This SessionFactory is needed to create a HibernateTemplate.

The HibernateTemplate makes data access code easier because it provides the basic methods to the database (CRUD) and was not needed to devote time for the exception, transaction, hibernate session handling. The Spring makes it easy to transparently create and bind a Session to the current thread either by using a class at the Java code level.

*(Minter, 2008, 55-59)*

### 3.3.3 Spring

The Spring is the framework which provides integration with Hibernate, JDO, Oracle, Toplink, JPA. Spring supports resource management, DAO implementation and transaction strategies. The support packages for ORM work with Spring's generic transaction and DAO exception hierarchies. The Spring's DAO template integration style was used where the DAO can be configured through Dependency Injection and resource and transaction management of Spring.

#### Dependency Injection

Dependency Injection is a form of Inversion of Control and is also known as the Hollywood principle: “Don't call us, we'll call you!”.

Dependency injection is a style of object configuration in which the object fields and collaborators are set by an external entity instead of embedding hard coded values into program or even having that piece of program code gather the values, it needs from an external location. When the time comes for the code to run, all information it needs to make available.

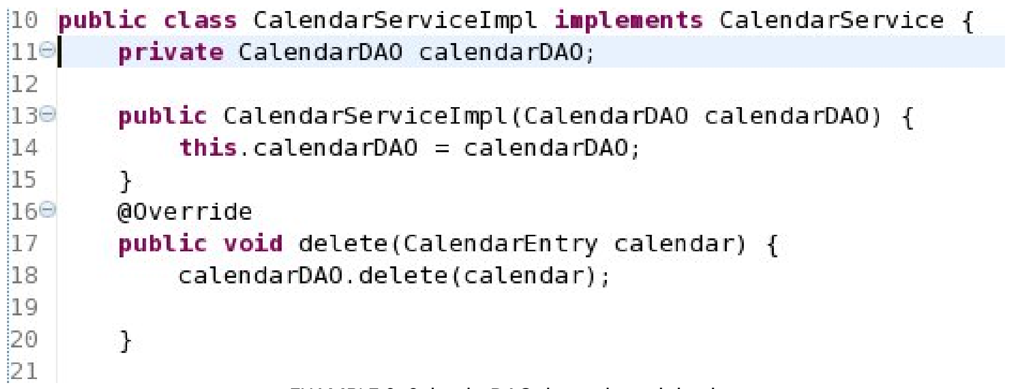
This means that the framework calls the application so the central component (IoC container) deals with populating fields in the objects with program code with currently appropriate values. After the information is injected, it becomes independent and there are no dependencies anymore on external objects.

There are existing ways how the object might get a reference from the external module by injection:

- Constructor injection, where the dependencies are provided through the class constructor (this is used in the project).
- Setter injection, when the dependent module is set through a setter method

that the framework uses to inject the dependency.

- Interface injection, when in order to get the dependency the user needs to implement an interface provided by exported module.



```java
public class CalendarServiceImpl implements CalendarService {
    private CalendarDAO calendarDAO;

    public CalendarServiceImpl(CalendarDAO calendarDAO) {
        this.calendarDAO = calendarDAO;
    }
    @Override
    public void delete(CalendarEntry calendar) {
        calendarDAO.delete(calendar);

    }

```

EXAMPLE 6. CalendarDAO dependency injection

Benefits of using Spring ORM:

- Testing: IoC of Spring framework makes it possible to change the implementations and config  locations of Hibernate SessionFactory instances.
- Transaction management: Spring provides integrated transaction management which gives proper transaction handling. In addition, transaction managers can be changed without the persistence layer specific code affecting.

*(Minter, 2008, 2-5)*

## 3.3.4 Facade

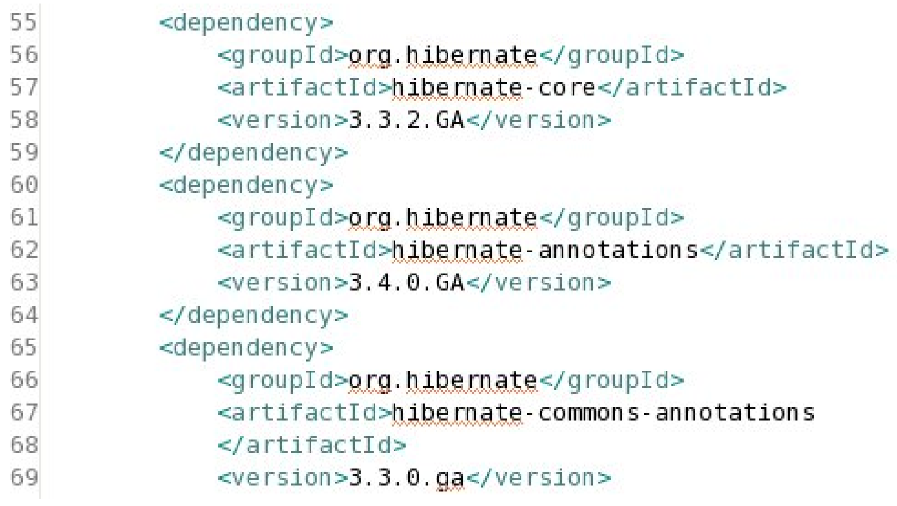
The facade is a structural design pattern that makes the subsystem easier to use and reduces the complexity. The goal is to decrease the communication and dependencies. The facade object which the CalendarService interface provides, is a single simplified interface to reach the more complex subsystem. The implementation of CalendarService (CalendarServiceImpl) knows which subsystem classes are responsible for the requests and forwards them to the correct subsystem objects which are the DAOs now. However, at the same time the subsystem objects do not store any references about the DAOs. The facade might have to translate its interface to the subsystem interfaces.

(Gamma, 1995, 272.)

## 3.3.5 Maven

The Maven is a tool primarily for Java project management and build automation. What really makes it different from the Ant is the concept. Maven found out the Project Object Model (POM) this is where the identity and structure of project are declared as well as the location where the builds are configured. According to the POM, the projects might be in relationship with each other; hence, a parent or a child project can be defined. In addition, the description of the given project is defined in an XML file so called pom.xml where instead of explicit instructions there are declaratives.

The needed Hibernate as well as Spring dependencies were decelerated.



```
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>3.3.2.GA</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-annotations</artifactId>
            <version>3.4.0.GA</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-commons-annotations
            </artifactId>
            <version>3.3.0.ga</version>
```

EXAMPLE 7. Maven dependency

# 4 CONCLUSION

## 4.1 Result

### 4.1.1 The server side

On the server side the chosen solution worked with the Funambol. The Syncml message handling was solved by Funambol which saved a lot of time compared to doing it all manually. The efforts were focused on making a so called Connector. Using Funambol Data Synchronization Server has advantages and disadvantages. Firstly, the SyncML message handling worked well and the calendar entry format too. Placing a test Connector without Hibernate went quite easily with the help of Connector skeleton class which is provided on their website. On the negative side the lack of support documentation caused a lot of dilemma. Maybe later in the future releases of the Funambol the problem will be solved. The complete implementation of SyncSource class and the integration of Hibernate classes are still missing.

### 4.1.2 The client side

On the client side the choice was using software already present on the phones instead of making an own. It saved the trouble of making a software client that would have to work well on many different phones and also the problem of getting the software out to the phones. The drawback of this is the lack of freedom because I could not affect how the client should work. For the sake of built-in client that acts as a black box , the aCal calendar format was not used on the phone , which made the study part of the solution searching harder. However, getting a software installed on the phone is also a big minus. So none of the options would have been optimal but the current one is still better of the two.

## 4.2 Personal experience

At Abakus Softwares Ltd., I had the opportunity to get experience in a foreign environment. My colleagues kept the connection with me in English and sometimes in Finnish.

After my boss defined the main goal of thesis, it seemed that I need to build an own synchronization server and thus I started to study SyncML as a possible synchronization solution candidate.

The SyncML as a standard is good and widely accepted. The structure of a SyncML message is very complicated and it results in a SyncML server with relatively complex structure. Much time was spent in order for me to able see through the structure.

I continued my task with the sending and the basic processing of SycML message by a test mobile device and an own built Servlet. Then I arrived at the real implementation of Synchronization logic which was not simple. At the same time I was looking for ready open-source projects to get help or replace the SyncML server building if the ready server seemed to work better.

I found the Funambol which completely met the minimal requirements and it did not take so long to reach the same point of development than at an own server. We switched to Funambol Data Synchronization Server and had to get to know its components.

Choosing Hibernate as an object-oriented library issued in more problems to resolve like making the Data Synchronization Server to work with the newly placed persistence libraries. I think I ran out of the time because we underestimated the peridod that was spent with research. For that very reason I did not finish the implementation of the Connector.

# 5 REFERENCES

*Bauer C., King G. 2007. Java Persistence with Hibernate*

Gamma E., Helm R., Johnson R., Vlissides J. M. 1995. Design Patterns

Funambol 2009. Developer's Guide version 7.1

Gosh S. 2003. XML-parsers

http://www.ibm.com/developerworks/library/wi-parsexml/. Referenced October 2009

Hansmann U., Mettala R. M., Purakayastha A., Thompson P. 2002. Synchronizing Your Mobile Data

W3 2004. Hypertext Transfer Protocol. http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html. Referenced December 2009

*Learn About Funambol 2008.* https://www.forge.funambol.org/learn/. Referenced December 2009

Mahmoud Q. H. 2004. Getting Started with Data Synchronization Using SyncML.

http://developers.sun.com/mobility/midp/articles/syncml/. Referenced November 2009

*Minter David 2008. Beginning Spring 2: From Novice to Professional*

SyncML Representation Protocol 2000