LEEDS METROPOLITAN UNIVERSITY

INFORMATION AND ENGINEERING SYSTEMS

SCHOOL OF COMPUTING

Enhancing Sync4j – An Open Source SyncML Project

Chintan Shah

March, 2003

Submitted in accordance with requirements for the degree of

Masters of Science in Mobile and Distributed Computer Networks

Abstract

Today's business applications and tools available in the market imply the growth of the mobile computing as a business requirement. Laptop computers, Personal Digital Assistants, mobile phones, etc are being used by business people to improve the way of doing business from where you are. Applications and data can be downloaded to mobile hand held devices from the office. It becomes essential to held data on both the sides (hand held devices and office) with the consistent state. Synchronization concept is used to achieve consistency. Various proprietary synchronization protocols are available in market, but poses problems to end users, developers, service providers for their inter operability and over head in maintenance. SyncML is an industry wide initiative to come up with a standard common synchronization protocol. Numbers of SyncML complaint products are found in the industry, but none of them are open source. Sync4j is an answer to it. Sync4j provides an extensible architecture with the capability of plugging in your own Sync engine. Sync4j is having all the core implementations of SyncML protocol, but lags some of the SyncML features like one-way synchronization, synchronization without separate initialization and complete support of sync anchors. This project looks into enhancing the above features within Sync4j.

Acknowledgement

There are many people, whom I wish to thank for their appreciated support throughout this M.Sc project.

My thanks go to Stefano Fornari, CTO of Funambol and project in charge of Sync4j, who has given me a chance to work on such marvellous project. He has provided me with the necessary help whenever I needed, along with never ending encouragement to achieve the goal.

I would like to thank my supervisors – Colin Pattinson and Joao Ponciano, for always being there to offer me help with guidance and feedback.

Special thanks go to my Family members for their love & financial support throughout my numerous academic years. Without them, I would not be what I am today.

I would also like to thank all those people who have directly or indirectly provided their unerring support throughout the course of this project, without whom none of this would have been possible.

Chintan Shah

Contents

List of Figures

The candidate confirms that the work submitted is his own and that appropriate credit has been given where the reference has been made to the work of others.

Report Layout

This whole report has been organized structurally into different chapters each reflecting about one of the topics (like Research, Analysis, testing). The report begins with the introduction of meaning Data synchronization, the main objectives of under going synchronization. The concept of working of synchronization is explained in section 1.3, followed by the areas where data synchronization can be used effectively.

Chapter 2 begins with looking at the different standards or protocols available in market implementing the concept of synchronization. As there are many protocols available, there should be some common characteristics that can help in interoperability between devices and services. What benefits can be expected if there is a common standard synchronization protocol is exploited in section 2.2. SyncML, one of the synchronization protocols, is an industry wide initiative synchronization protocol aiming to be adopted as a standard protocol for data synchronization. A brief introduction to SyncML is provided in section 2.3 with its capability of being represented as data synchronization and representation protocol is looked into section 2.4. Section 2.5 looks into how widely SyncML is adopted based on its available products today in the market.

Chapter 3 studies Sync4j, one of the SyncML implementation. This chapter explores the Sync4j's architecture. The research work that has been done within Sync4j to identify the problem statement is explained in section 3.2. Developing some modules within Sync4j and integrating them into the current state solve the problem. The software development life cycle followed for this project is explained later in that section.

Chapter 4 explains the complete analysis, design and development carried out for developing the modules as solutions of the problem. Initially the high level flow of information within Sync4j is explored followed by each section explaining the general usage of the feature to be integrated. How it is currently achieved and what modifications / additions are required to build the feature. What benefits can be achieved by designing as proposed, are explained for each feature in their subsections.

The developed product is tested and results are studied within Chapter 5. The development and the testing environment are explained in section 5.1 with the way of working a test case in section 5.2. Section 5.3 shows a test case matrix is developed to conduct different test rounds over the entire product followed by explaining an entire test case with the data transfer in section 5.4. The results received over different cases with the bugs exploited are detailed in section 5.5.

Chapter 6 carries out the evaluation of the product to analyse the target achieved by developing and integrating the modules against the problem identified with chapter 2. Evaluation is carried on two stages; evaluation of the basic analysis of the features and evaluation of the developed product are done in section 6.1 & 6.2 respectively. The conclusion derived from the above whole process is detailed within section 6.4. Section 6.5 overviews the future work that can be done within this area.

Chapter 1

Introduction

"Make everything as simple as possible,
BUT not simpler"

– Albert Einstein
(1879 – 1955)

1. Introduction

The proliferation of mobile computing devices has fuelled towards the distribution of information. The paradigm mobile computing provides user with access to their data wherever they are: on the road with their Personal Digital Assistant (PDA) or laptop or personal computer (PC) inter connected to the real network. The ability to access and update information on the fly is becoming necessary for the growth of business. Rob Veitch, Dir. Of Business Development at iAnywhere Solutions Inc, supports it by saying *"Customer service and operational efficiency can be significant – and can scale up dramatically as applications are mobilized."* (Veitch, 2003)

The key to mobile and pervasive computing is the use of applications and information on the mobile devices, clubbed with synchronization to hold latest information from the desktop office computer or on the network, which will benefit users on the move from having access to a greater number of applications that are totally integrated into their corporate applications. This ubiquitous data residing on computer and hand held devices must deal with the inherent problem of *data synchronization*; that is change(s) made on the office should be reflected on the mobile device and vice versa. Karen Scherberger, an analyst at Gartner Group raises the same question: *"How do you enable the mobile worker to say, write up orders from the field with the up-to-date information from the corporate database without synchronization?"* (Blodgett, 1997) What is Data Synchronization? Is this some kind of software or hardware? Let us understand it in next section.

1.1. Data Synchronization

The word 'Synchronize" means to happen or go together like clocks to show same time. Expanding synchronization in co-ordination with data i.e. Data Synchronization – which represents the process of making two data sets identical. Let us review, what people fell and think about data synchronization?

Andreas Jonsson of Ericsson identifies synchronization as a real business need based on the statistics presented by management consultancy Booz Allen & Hamilton quoting: *"67 % of professional workers in Europe are away from their desks or work area more than 20% of time requiring them to keep appointment schedules, customer information, and task lists up to date while on the move."* (Jonsson, et.al 2001)

International Retailers founded body World Wide Retail Exchange (WWRE) comments on relationship between Data Synchronization and companies by saying *"Data synchronization is a process that aligns trading partners' information system and databases with timely and audible distribution of certified standardized master data from an originating data source to a final recipient of this information."* (World wide retail exchange, 2003)

Synchrologic, a comprehensive mobile infrastructure solutions provider, considers "Synchronization as an integral and necessary component of the mobile infrastructure. The notion of mobility in and of itself creates the need for synchronization." (Synchrologic, 2000)

This shows synchronization as a mandatory requirement for growth of business tending to be mobile, as it helps users to keep business lives well organized with required latest information being handy and accurate. Is this the only objective for conducting synchronization within business requirements? *No*. then what else..?

1.2. Objectives of Synchronization

Business is always benefited by adopting better technology which can provide required business solutions. Following objective shows how this technology concept can benefit a business organization.

a) **Keeps each System with up to date information**

   Distributed information systems can consists of multiple sites, each holding a local copy of the data required at that site. Synchronization technologies can exchange information among the sites keeping each site current with correct data.

   Example: A multi national company has its own distributed networks all over the world. If each network would be synchronized with each other, thereby every

local server would hold up-to-date equivalent information as other server residing far abroad. This in turn provides better access time and availability of business data to its distributed local users.

b) **Reduces network data flow**

Flow of data on the network can be reduced drastically by accessing the local synchronized data. Requests can be made to local server, who can in deed responds to its requests by using the data obtained through the synchronization.

Example: As stated in above example, users can access to the data from their local server instead of fetching it from the remote server, there by reducing network data transmission.

c) **Reliable data**

In spite of mobile device holding data not always being connected to the network is still relied on as it contains the latest correct information retrieved through last successful synchronization.

Example: Mobile users can use handy devices like Personal Digital Assistant (PDA) to download its latest information like calendar entries by connecting to the server. The user can then disconnect to the server, still holding the valuable up to date and reliable data on this PDA.

d) **Resolve few arising conflicts:**

In distributed systems, time clock must be in the same identical states in order to avoid conflicts arising due to difference in time. Synchronizing clocks over network allows communications to be transferred properly throughout the network. Thus synchronization of time must take place in such environments. (Coulouris, et.al 2001).

Example: When we are saving any file to a network server, we want the file's time and date stamp to be same to both the systems (user's machine & network server). If the users and network machines are in time sync, both will represent the same time & hence avoid time conflict problem.

e) **Deliver Quality of Service (QoS):**

Applications like Voice & video needs to go in synchronous with each to deliver meaningful information. If applications cannot deliver the minimum required Quality of Service, they are no longer required within business.

Example: If voice is out of sync with the video within some application like Video conference, it would not be making good sense of them as it makes difficult to interpret them.

Now the question arises: How are these objectives achieved by this concept called Data Synchronization? Let us exploit concept of synchronization to find the answers.

1.3. Concept of Data Synchronization

To maintain the consistent state of data, synchronization should be carried out after data has been altered on either of the side (Desktop / PDA). Different granularities of consistencies can be achieved, say at database level or file level. File level can be achieved using CVS (Concurrent Version System). It keeps a history of the changes made to a set of files. While updating the files, it cautiously merges files, as long as changes aren't made to the same lines of the files (CVS, 2002). Within this project we try to concentrate on synchronization of databases instead of files. Let us take an example of a contact address book database within mobile phones to get a clear picture of database synchronization. Address book consists of fields like name, company, street, home phone, office phone, e-mail, fax, etc. All these fields are within one contact person (name) and can be considered as a record of a table. After carrying out successful synchronization, both the devices should hold the same entries (as shown in figure 1).

Maximilian says the simplest way of achieving synchronization is to do replication of databases. In this case, the current database is copied over with the updated database, bringing in both the database in consistent state (Maximilian, 2002). Based on how synchronization is achieved by exchange the full / partial (modified) data, Synchronization can be broadly categorized into two categories: Slow sync and Fast sync. During slow sync, also known as full sync, all entries from one side are sent to other side. Sync analysis is carried out to find the entries are matched to avoid duplicate entries and finally update both devices with the new or modified ones. This long process is required if the last

synchronization state was lost and hence don't have any idea of data hold by either of the sides.

Fast sync involves sending only modified entries to the other side making aware of changes and brings both devices in sync. It seems to be more efficient as the whole database is not sent over possible low network bandwidth. Modified entries on server can be traced using "last modified" time stamp. Similarly dirty bit can be used by client. Dirty bit is set to a record if there is any modification is occurred on it. While undergoing synchronization all entries containing dirty bits are send, and once database are synchronized successfully, dirty bits are erased from entries. Conflicts do occur during synchronization, which is left upon the synchronization engine & business logic implemented to solve the conflicts. Manual intervention may be requested to solve the conflict if required (Buchmann, 2002).

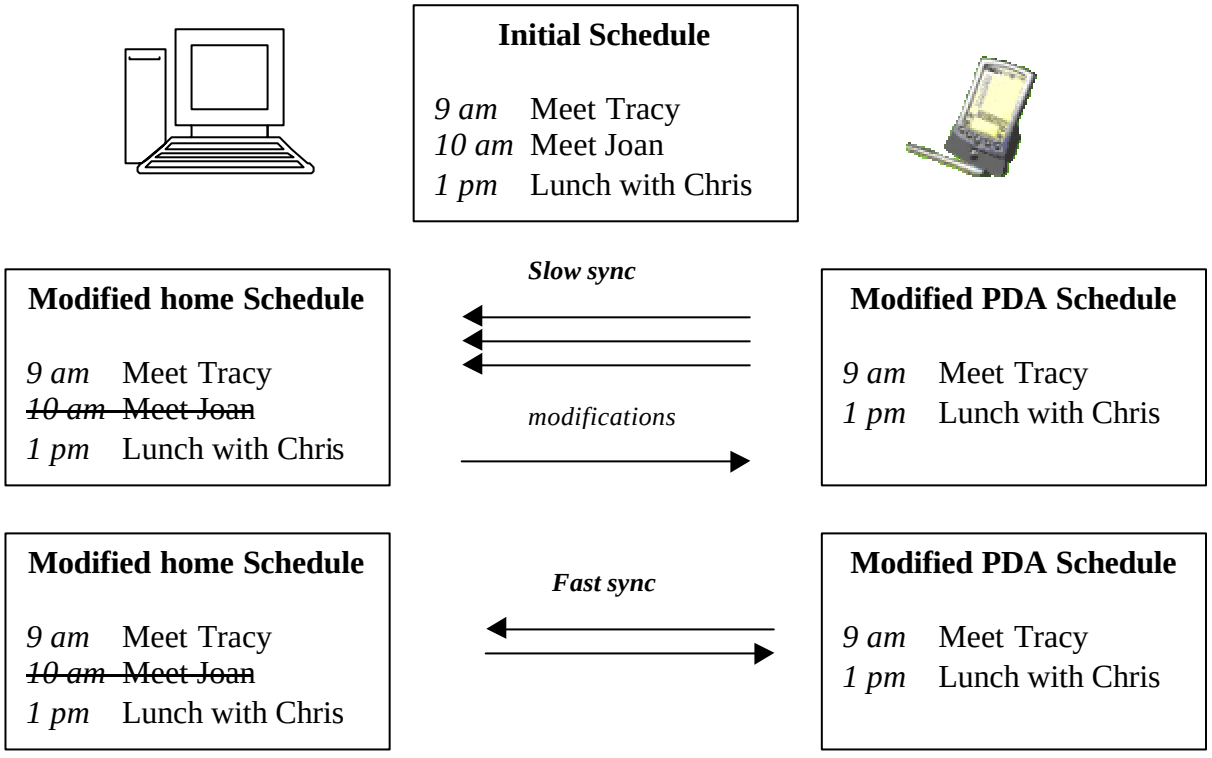


Figure 1: Comparision between slow & fast sync

Figure 1 shows the comparison between the slow and fast sync. Slow sync transfers all the entries of the database from the home pc to the PDA, while in fast sync only the

modifications are send to the PDA from home pc. The concept of above figure has been taken from Agrawal et.al (2002).

Slow sync is the process of synchronizing the entire database, whereas fast sync is the process of only synchronizing the changes. If more than two devices need to synchronize, fast sync cannot be used; as the process will reset the status flags (dirty bits), when synchronizing with one device even if they are required to synchronize with other. It can be achieved by server storing separate set of dirty bits for different devices over same database. Within slow sync, the communication and time complexities increases linearly with the number of records stored in the device, independently of the number of record modifications as all records are exchanged.

1.4. Areas of Data sync usage

Fundamentally synchronization sees that two devices undergoing synchronization gets into an identical state. This fundamental makes synchronization fits itself into various domains of usage. Now we look into few areas implementing synchronization.

a) **Web-Calendar (vCalendar):**

Changes made by a secretary to the appointments and meetings on the web-based or office network based calendar application can be updated on a mobile device of the personnel (Boss) involved. Boss can in turn send update made by him to secretary. In this case, both the user can be updated with any modifications taken place.

b) **Contacts (vCards – electronic business cards):**

Contacts can be exchanged in similar fashion to web-calendar. vCards are the electronic form of business cards. They are generally used in applications like e-mail, Web browsers, video conferencing, Personal Information Managers (PIMs), PDAs, smart card, etc.

c) **Mail messages:**

Mobile users would prefer to read and reply their mail messages on the move. They would like to have a copy of message on their desktop to see the same when they are in the office. Messages are kept in same consistent state to both the devices with which message has been read and what replies has been send.

d) **Network files:**

Files stored on a network machine can be stored in a similar identical state as other by synchronizing them. This kind of synchronization generally takes place to either mirror a site or back up data or bring network data servers into identical states.

e) **Time Clock:**

The clock time on the network must be synchronized to avoid conflicts aroused due to different times on different machines. It is better to use UTC timing to solve global time constraints.

Synchronization of clocks was used within bus priority functions defined in Helsinki to provide benefits in bus timings. More about the project can be found at (Sane, 1998).

f) **Audio Video applications:**

Various applications are there which needs audio and video within them. If within application audio speech comes late by few seconds then that of corresponding video or other way i.e. out of sync with each other, then that sequence becomes a lot difficult to understand making it no sense. Hence within such applications they both must be in synchronous with each other to make some good sense.

Chapter 2

Synchronization and SyncML

"If it [technology] keeps up, man will atrophy his limbs but the push button finger"

– Frank Lloyd Wright

2.
Synchronization and SyncML

Frank says in the above quote that technology should be kept up & must be developed, but if it keeps on progressing this way, it will limb us except the push button finger!!! The concept of synchronization should be extended with communication syntax and semantics within the ongoing session to identify itself as a protocol. This protocol should also include: (Sullivan, 2001)

1.1. Naming and identification of records
1.2. Common protocol commands
1.3. Identification and resolution of synchronization conflicts

Generally all these features are incorporated within various synchronization protocols available in the market today like Palm HotSync, Pumatech's Intellisync, Microsoft's Active Sync, SyncML, etc. Let us have an overview of each of them in the following section.

2.1. Different types of protocols

Synchronization protocols were developed by the companies as per their business or commercial requirement. Hence majority of them turned out to be proprietary protocols. Here we go through a brief introduction to few of them with their main functionalities.

2.1.1. Palm HotSync Protocol

Palm operating system is generally found in Palm PDAs or Palm pilots. These devices support synchronization through its proprietary protocol known as HotSync synchronization protocol using the interfaces of its Palm operating system. HotSync provides two modes of operations – Slow sync and Fast Sync and provides record level synchronization.

2.1.2. Intellisync Protocol

IntelliSync Anywhere from Pumatech seeks to make each type of synchronization - "Fast sync enabled" in order to minimize connection times. Its entire technology is based on central server architecture, where in all the devices synchronize with the central server. Central server stores all modifications and status flags of each device and hence sends only the modifications occurred after last successful synchronization with that device i.e. fast sync. This system enables synchronization with corporate applications like Microsoft Outlook and Exchange, between desktop and mobile including Palm PDAs, Pocket PCs and symbian devices. The major drawback of this protocol is its centralized architecture as it lacks synchronization between two devices, which needs to be done through server. (Agrawal, 2001)

2.1.3. Active Sync

Active Sync is synchronization software by Microsoft for Windows powered pocket PCs. It is based on Microsoft's operating system platforms with various user-friendly features like connecting with USB, Infrared with automatic detection of communication ports on desktop computer. It is capable of synchronizing web favourites, note documents of Microsoft Outlook, automatic detection of changes in Schedule+ or Outlook (Microsoft, 2003). This implementation of this protocol is available through its software known as Microsoft ActiveSync. Additional features of this protocol are available at Microsoft's site. (Microsoft, 2003) Overall it seems to be robust enough for synchronizing Microsoft Outlook and Pocket PCs.

2.1.4. SyncML

"*SyncML is an industry initiative to develop and promote a single, common data synchronization protocol that can be used industry wide.*" (SyncML [a], 2001) i.e. It seeks to provide an open standard for data synchronization over different platforms and devices, which is indeed promoted by hundreds of industries. Few of them are Ericsson, IBM, Lotus, Matsushita Communications Industrial Co, Ltd, Motorola, Nokia, Openwave, Starfish software and Symbian.

All of the above protocols have some special features within themselves. What are the common features (characteristics) expected from a synchronization protocol? What kinds of

applications are generally supported by standard protocol? Let us study the characteristics of a common synchronization protocol in the following section.

2.2. Characteristics of a common synchronization protocol

The goal of a common synchronization protocol must be symmetric. It should provide synchronization between any devices over any medium i.e.

- Synchronize networked data with any mobile device
- Synchronize a mobile device with any networked data.

This means that data synchronization protocol should support synchronization to be carried over any network medium (wireless, fixed line, etc) between varieties of synchronizing devices like desktop PC, automotive computers, handheld computers, mobile phones, etc. Protocol is expected to support more data types like calendar entries, contacts, e-mail, enterprise data, documents, etc. It is expected that protocol should be able to integrate with the existing transport protocols and technologies. All this can be summarised as, the protocol having following characteristics: (SyncML [a], 2001)

- Operate effectively over wireless and fixed line networks
- Support a variety of transport protocols
- Support arbitrary networked data
- Enable data access from a variety of applications
- Address the resource limitations of the mobile device
- Build upon existing technologies (Web & Internet)
- Protocol's minimal functionalities must deliver the most commonly required synchronization capabilities across the entire range of devices.

From the above discussion, question(s) arises:

Why there is a need for common standard synchronization protocol?
What benefits can we expect by having such standard protocol?
Can't the above proprietary protocols only be sufficient?

The main reason to have a standard common synchronization protocol is to have a uniform standard supported by every capable device over all platforms providing all the basic required functionalities. Majority of the protocols explained earlier in this chapter are proprietary protocols. SyncML seems to be the only protocol complying with this requirement. Is there any other alternative to SyncML? David Buchmann (2002) carried out an enquiry to find it and the answer was straight forward and clear. *'The enquiry revealed no real alternatives to SyncML. There are standards for network communication, but SyncML is the only general standardized protocol with specific capabilities for synchronization of records*". The above proprietary protocols are fair enough for special cases, eg. HotSync provides record level sync between palm & desktop pcs. iCalendar Transport Independent Interoperability Protocol (iTIP) is defined by IETF for synchronizing iCalendar entries. Industry-wide benefits can be achieved by adopting common synchronization protocol like SyncML, as explained below.

Benefits of a common synchronization protocol

By adaptation of a common synchronization protocol, everybody within this domain of synchronization is expected to benefit from it. Let us see how end users, device manufacturers, service providers and application developers will benefit from it (SyncML [a], 2001)

1) **End Users**: Users use handy devices of different manufacturers based on their application requirements. Technology and applications used to communicate between them differs depending upon supported protocols. It becomes cumbersome & expensive to configure, operate and maintain different applications supported by different devices. It would be easy for a user to maintain a single application (protocol) that can be used with all devices for basic required services.

2) **Device Manufacturers:** Every device manufacturer would like to have their devices capable of supporting the data access needs of all users and service providers, which is achieved by providing support for various protocols. Companies tend to have their own proprietary protocols capable of doing their own things in its proprietary way hindering effective interoperability between devices of other manufacturers. By adopting a common standard protocol, interoperability can be achieved on broad range for applications, services, etc.

3) **Service Providers:** Service providers do provide a support for wide range of data types, devices and its applications by bearing the over heads of installation, configuration and maintenance of different proprietary applications by facing problems of compatibility and performance. To enhance integration of broad range of services, performance; compatibility must be resolved by using a common standard protocol.

4) **Application Developers:** Applications developed for multiple protocols seems to be expensive in terms of cost and time, while compared to developing more products for a single standard protocol. Developing more services and applications for a standard protocol makes it worth by exploring its usage with additional capabilities.

Analysing the above discussions, SyncML clearly comes out to be a definite standard common synchronization protocol as also supported by David Buchmann's survey. It would be worth to explore within this protocol to see what work can be done within it to help the mobile computing industry.

2.3. Overview of SyncML

SyncML can be considered as a common language for synchronizing different devices and applications over any network. With SyncML any kind of information like calendars, contacts, task lists, and e-mails can be synchronized to make information consistent and up-to-date, despite of the location of the devices and the network.

Synchronization problems can occur for consumers, carriers and business, as proprietary systems don't talk to each other. Gregg Armstrong, chief operating officer at Starfish Software criticizes this problem of synchronization giving birth to SyncML by saying "*That's one of the reasons we initiated the SyncML consortium*" (Crouch, 2001). Generally the proliferated proprietary protocols are available over selected transports (like HTTP, OBEX) and that to restrict within few limited devices. They also seemed to have access for a small set of networked data. Looking at the absence of a single, standard synchronization protocol causing many problems for end users, device manufacturers, application developers and service providers, SyncML was initiated (SyncML [a], 2001).

The adjoining figure shows the SyncML Framework (SyncML[b], 2002). This framework helps in analysing the implementation of the system model associated with SyncML implementations. The major blocks of this framework are SyncML representation protocol, conceptual SyncML adapter and SyncML interface. SyncML data synchronization protocol is outside the SyncML framework to keep the framework easier to implement and provide interoperability. The SyncML data synchronization protocol is defined by a companion SyncML specification known as SyncProto.

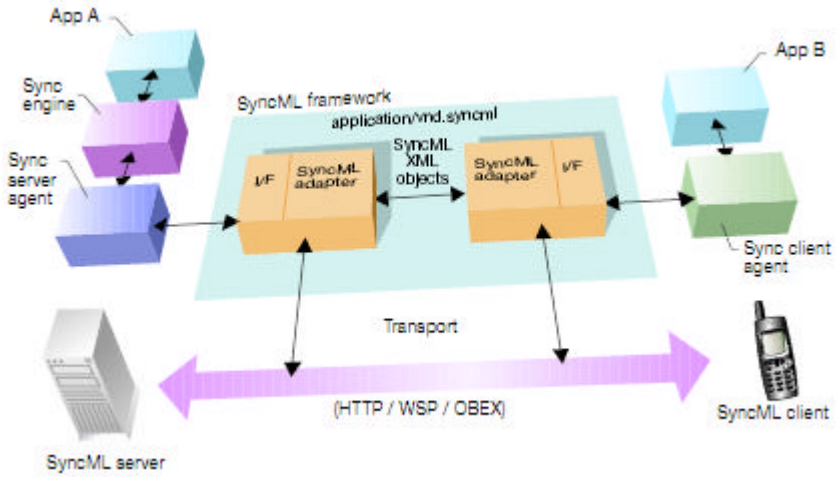


Figure 2: SyncML framework

Let us consider application ‘A’ is a networked service that provides data synchronization with other applications like ‘B’. We also assume application ‘B’ a client wishing to synchronize with ‘A’. The service and device are connected over one of the transport protocols (HTTP, WSP, and OBEX). Application ‘A’ implements “Sync Engine” process utilizing the data synchronization protocol manifesting client access to the “Sync Server” – a network resource. “Sync Server Agent” communicates between client applications and “Sync Engine” to manage access to network and to/from communications. Invoking the functions defined in the SyncML interface performs these capabilities. SyncML interface is nothing but an application (programming interface) to the “SyncML Adapter”. “SyncML Adapter” is a conceptual process which can be considered as entry / exit point for Application ‘A’ i.e. originator and recipient of the SyncML formatted objects. Actual server and client implementations may not be implemented in the discrete components explained above. The main components of a SyncML specification are:

- an XML-based representation specifications;
- a synchronization protocol; and
- transport bindings for the synchronization protocols.

SyncML can bind over wireless and wired medium with variety of communication protocols like HTTP, WSP (Wireless Session Protocol), OBEX (i.e. Bluetooth, IrDA), TCP/IP networks, and proprietary communication protocols, SMTP, POP3 and IMAP (SyncML [a], 2001). Let us now explore more about how the data is represented and synchronized using SyncML in next section.

2.4. SyncML as Synchronization and Representation Protocol

One of the unique features of SyncML is having capability of being represented as a synchronization as well as representation protocol. This feature makes SyncML more attractive to be deployed within the corporate applications. Synchronization protocol describes about using the Representation protocol for creating SyncML messages containing the initialization, synchronization and completion of the synchronization session. Hence we first begin with looking at representation protocol followed by Synchronization.

SyncML as representation protocol

A basic structure of a SyncML message can be represented as shown in figure below (Andreas, 2001). A SyncML message consists of two parts: SyncML Header and SyncML Body. SyncML Header contains the routing, session, authentication and information about the message. SyncML Body contains the various synchronization commands to be performed like Sync, Put, Get, Status, etc.

SyncML contains a set of well-defined messages represented as XML documents or as MIME (Multipurpose Internet Mail Extension). All this representation is defined using a XML DTD (Document Type Description). It represents all the information required for synchronization like commands, data, metadata, etc. The synchronization specification specifies the SyncML messages that comply with this DTD and allows meaningful communication between SyncML client and server.

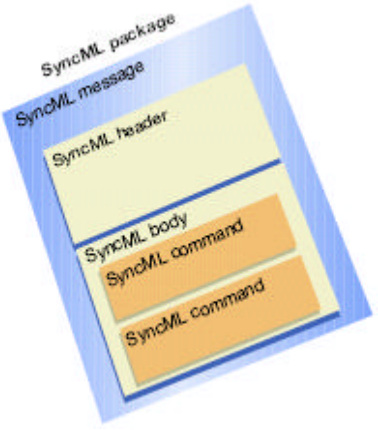


Figure 3: A basic structure of a SyncML message

For the purpose of transporting SyncML messages with different transport and session protocols supporting MIME, the content types "`application/vnd.syncml+xml`" and "`application/vnd.syncml+wbxml`" are defined for clear text XML and binary encoded XML messages (WBXML) respectively. One of them must be used for identifying the SyncML message at the transport and session level protocols that support MIME content types. This format specifies a set of XML elements, which can be divided into five categories depending upon their functionalities (SyncML [d], 2002).

- Message container elements
- Protocol command elements
- Protocol management elements
- Common use elements
- Data description elements.

The message container elements contain three base elements `<SyncML>` element with its two child elements `<SyncHdr>` and `<SyncBody>`. Data description and common use elements can occur within `<SyncHdr>` and `<SyncBody>` elements. Protocol command and management elements can only occur within `<SyncBody>` element.

SyncML as Synchronization protocol

Using representation protocol, SyncML as a synchronization protocol defines how to initiate, synchronize and terminate the synchronization including exchange of data in between the synchronizing devices (client and server). SyncML is designed considering it as a common synchronization protocol, taking care of the case where the network services and device(s) storing the data that are being synchronized are in different formats or used by different software systems. There are seven different types of synchronization defined by SyncML as stated below, depending upon the synchronization initiated by the client / server and the transfer of information between them.

1) **Two-Way Sync:** This is the most normal and widely used type of synchronization where the client and server exchanges only the data modified at their end. Client is always expected to start the synchronization by sending its modifications first.

2) **Slow Sync:** Slow synchronization can be considered as a form of two-way synchronization with an exception of client sending the entire database rather than only modified items, asking server to perform the sync analysis (field-by field basis) for the data between client and server. Server returns only the required modifications to the client. This type of synchronization is generally carried out when the device is synchronizing for the first time or sync anchors don't agree between devices.

3) **One-Way Sync from Client only:** During only this type of synchronization, client sends all its modifications to the server, but does not expect any modifications from the server. Hence, server is updated with client modifications, but client remains unaware of modification undergone at server.

4) **Refresh Sync from Client Only:** This is a special type of One-Way Sync from Client, where client exports all its data from the database to the server. Server is expected to replace (over write) its existing data with the received client database brining both the device in synchronization.

5) **One-Way Sync from Server Only:** This type of synchronization sends all modifications from server to the client and not expecting any modifications from client; and can be considered as reverse of One-Way Sync from client where in modifications are sent from client to server. Client updates itself with the modified data, but server remains of unaware of any updates on client.

6) **Refresh Sync from Server Only:** This is a special type of One-Way Sync from Server, where server exports all its data from the database to the client. Client is expected to replace (over write) its existing data with the received server database bringing both the device in synchronization.

7) **Server Alerted Sync:** Server alerts a client to perform synchronization if server intends to undergo synchronization with the client. It also tells the client with one of the above types of synchronization to be executed in between them.

For each type of above defined synchronization, there are set of common features and requirements possible. These features are can be stated as: (SyncML [b], 2002)

- Change of log information
- Sync Anchors
- ID mappings of data items
- Conflict Resolution
- Security
- Addressing
- Exchange of device capabilities
- Sync without separate initialization
- Device memory management
- Multiple messages in a package
- Busy Signalling

2.5. SyncML Implementations

How widely SyncML is adopted in the market is itself evident from the available products of SyncML in the market. The more the products are available implies its more usage and indeed the acceptance of this protocol as standard protocol for achieving synchronization. At the time of preparing the report, there were 110 SyncML compliant products as published by SyncML. More details about each compliant product with their interoperability can be found at official site of SyncML [`http://www.syncml.org/interop/interop-compliant.html`]. These compliant products are:

- 11 SyncML Data Synchronization v1.1.1 clients
- 5 SyncML Data Synchronization v1.1.1 Servers
- 3 SyncML Data Synchronization v1.1 Servers
- 58 SyncML Data Synchronization v1.0.1 clients
- 31 SyncML Data Synchronization v1.0.1 Servers

Few of them well known server in v1.1.1 category are Oracle Sync Server, Starfish True Sync, Critical Path Sync Server, FusionOne Mighty Mobile SyncML Gateway, etc. Generally company for their internal use or as a business commercial product prepares all of these products. A little work has been done within development of such products under Open Source Category. SyncML products are being developed under Open Source category

using C, C++ and Java. SyncML C Reference toolkit is provided by SyncML for implementation reference of SyncML protocols. kSync is implemented using Java, capable of working with contact lists, calendars and any user-specific data in XML format (kSync, 2002). kSync started with development of sync engine for their internal requirement coming with solution name Tequila kSync. Its open source code is available at [ksync.enhydra.org]. Sync4j is also a Java based SyncML server, but with distinct features making it more attractive to be explored within next chapter.

Chapter 3

Overview of Sync4j

"Scientific research consists in seeing what everyone else has seen, but thinking what no one else has thought"

– Unknown Source

3.
Overview of Sync4j

Sync4j is an open source project aiming to be a full-featured SyncML server based on Java platform. Sync4j is sponsored by Funambol [www.funambol.com] and sourceforge.net. It is currently hosted on services offered by SourceForge.net [www.sourceforge.net] and developed under BSD License [http://www.opensource.org/licenses/bsd-license.php]. Sean C Sullivan, one of the project managers of Sync4j, defines the targeted audience of Sync4j when it started in 2001 as: (Sullivan, 2001)

- Developers who know Java, but don't know SyncML
- Developers who know SyncML, but may not know Java
- Commercial application developers
- Open source application developers

Currently Sync4j is addressed to everybody who needs to provide or implement synchronization services. The uniqueness of this project is to build up an application base of SyncML, which can integrate itself with the users' requirements for synchronization or further development. Users can plug in their own implementation of the synchronization logic (engine), authentication policy. Let us now explore Sync4j through its architecture. The Sync4j's plug gable architecture allows the business logic of the protocol and synchronization can be developed and extended separately (i.e. without altering any modules) to meet the best requirements from it.

3.1. Sync4j Architecture

(Sync4j, 2002) official reference material from Sync4j is used to explore its architecture within this section. Sync4j implements following functional blocks:

- The SyncML protocol
- Synchronization engine

- A server waiting for response
- The interface to remote devices (transportation / communication)

The Sync4j architecture can be layered as shown below.

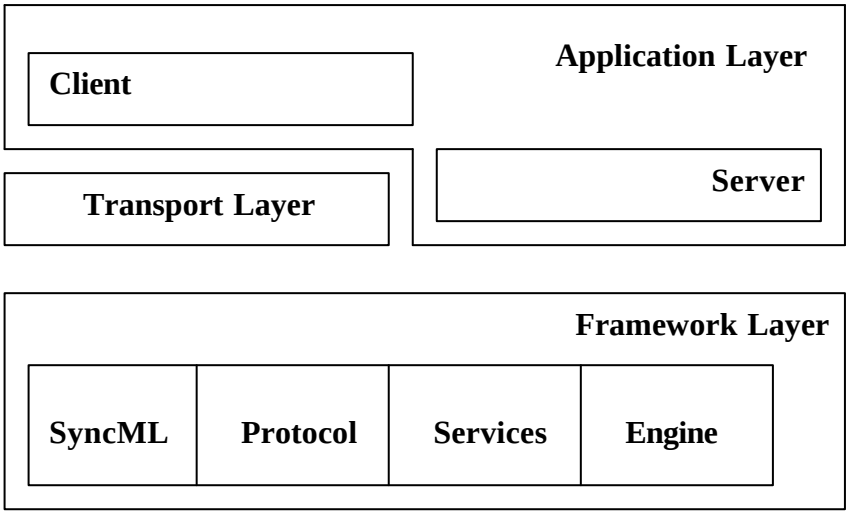


Figure 4: Sync4j layered architecture

Application Layer:

Sync4j server and client are represented through *sync4j.server* and *sync4j.client* packages. The server is implemented as Enterprise Java Bean (EJB) that can be deployed on J2EE (Java 2 Enterprise Edition) application server. There were various reasons to select J2EE application server, which are stated in (Sync4j, 2002).

Transport Layer:

Sync4j is designed to support various transport protocols as stated above. But currently it is only supporting HTTP protocol with the help of servlet implemented through *sync4j.transport.server.server.Sync4jHttpServlet.java* file. Other protocols supports will be added within its further releases.

Framework Layer:

Sync4j framework layer consists of many packages. Few of them are:

- Sync4j.framework.client → client applications
- Sync4j.framework.server → server applications
- Sync4j.framework.engine → sync engine
- Sync4j.framework.logging
- Sync4j.framework.security

  } Sync4j services

- Sync4j.framework.core
- Sync4j.framework.protocol

  } syncml protocol

SyncML protocol is implemented within *sync4j.framework.core* and *sync4j.framework.protocol* packages. Core package groups all the foundation classes that represent a SyncML message based on SyncML specifications. The classes of the framework are responsible for checking the validity of a SyncML message. The validity is carried out in terms of XML structure representing the message regardless of the context of message being processed. It sees that messages respect SyncML specifications. Protocol package adds functionality of checking all the initialization and modification requirements required during the process.

Sync4j services like logging of events and authentication are provided through *sync4j.framework.logging* and *sync4j.framework.security* packages respectively. Sync4j implements the Java Authentication and Authorization Service (JAAS) available within Sun Java development Kit 1.4 (JDK 1.4). *sync4j.framework.engine* implements the synchronization engine logic and is an important part of Sync4j architecture carrying out synchronization rules. *sync4j.framework.server* and *sync4j.framework.client* includes common classes for the development of server and client classes.

Persistent Store:

Persistent store is the concept used to store the persistent information required by the Sync4j server in order to support the protocol and provide its services. The persistent store is designed using entity beans. Persistent store is a class that stores objects within persistent

media such as database. The storage and retrieval of data through this persistent store is represented as methods: store(object) and read(object). These methods take the appropriate actions based on its behaviour about getting read or written. The core of the persistent store architecture is the interface *sync4j.server.store.PersistentStore.* It can be configured using configure (map) having *java.util.Map* parameter containing the configuration properties. The reason of implementation of such store is to:

- Centralize the database access logic in few specific classes, as opposed to spreading of database access code whenever required by classes.
- Keep the role and responsibilities of classes as much clear as possible.
- Decouple database access from other Sync4j modules

Overall have a flexible and light way to access the database for the storage purposes of the server activity.

Mainly the persistent store is used to store persistent information regarding the synchronization rocess and status to a relational database. SyncPersistentStore is configured with the following properties.

- Jndi-data-source-name: the JNDI name used to look up the datasource
- Username: the database user name
- Password: the database user password

A persistent manager (*sync4j.server.store.PersistentStoreManager*) is provided to take care of the persistent store as it grows.

Database and Principal:

Principal is a new concept just introduced within Sync4j. The main purpose of using this concept is to ease the process of identifying the user, its last sync device and the database. Figure in the following page shows the database tables used by the sync engine.

The brief description of the tables used in Sync4j is described below.

- **User** (*Sync4j_user*): Contains Sync4j user details like user name, first name, last name & e-mail address with username as primary key.

- **Device** (*Sync4j_device*): Holds the devices Sync4j can deal with. Device ID, the primary key, is the ID specified by the client within its `<SyncHdr>..<Source><LocURI>` element. Other fields of this table are device description and type of device.

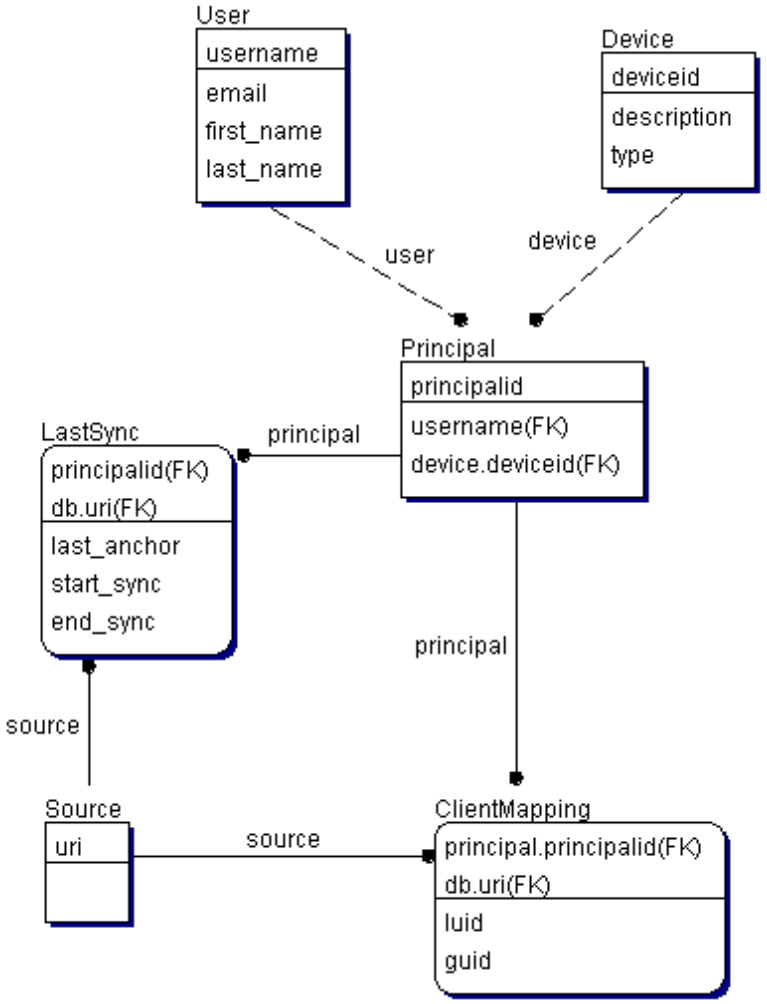


Figure 5: Engine database schema

- **Principal** (*Sync4j_principal*): It is the entity that can make a synchronization request. Conceptually the principal is a couple of user and device i.e. (username, deviceid).

- **Client Mapping** (*Sync4j_client_mapping*): It stores the Local User ID (LUID) – Global User ID (GUID) mappings for a particular database and principal

- **Source** (*Sync4j_sync_source*): It holds the known sync sources Sync4j can deal with. A source can be identified by its URI, which is used also as a key in the referring tables.

- **Last Sync** (*Sync4j_last_sync*): It stores the anchors of the most recent synchronization of the particular database by a particular principal. It also holds start and end synchronization time (anchors).

More information about the architecture including the implementation of current Sync4j sync engine can be found through the Sync4j official architecture reference material – (Sync4j, 2002).

3.2. Research within Sync4j

Research has been carried out to identify the enhancement requirements from the open source synchronization project – Sync4j and has been explained below. To reach this stage first of all the architecture and current status of Sync4j has been studied followed by learning the SyncML specifications like SyncML Sync protocol, SyncML Representation Protocol data synchronization usage, SyncML Representation protocol, SyncML device information. All this documents are available at `http://www.syncml.org`. State of Sync4j at that time was a server designed and developed with core modules like Sync engine, user connectivity through HTTP transmission protocol, authentication, and implementation of various SyncML specifications through its core & protocol packages.

It becomes important to see that within Synchronization, devices already undergone synchronization gets update only of those data's that has been altered from its previous known consistent state in order to prevent from unnecessary flow of data between two synchronizing devices. SyncML Anchors can be used to retrieve the last time stamp when successful sync occurred between those devices. Modifications occurred after this anchor is exchanged to bring both the devices in sync with each other. This also leads to requirements of either only sending or only receiving modifications or both of them from the device i.e. One-way or Two-Way Synchronization.

Synchronization is a process of bringing two devices into a consistent state. Many times, a synchronization scenario comes into play where majority of modifications take place on the server and client keeps on retrieving the updates as and when required. This feature can be stated as One-way synchronization from server. The other way round for the scenario involves various updates taken place on different clients keep on updating their server with modifications occurred at their end and not receiving any modifications from the server. This scenario can be stated as one-way synchronization from client.

The importance of these two features can be revealed using the examples of a stock market company and courier service respectively. A stock market company is having its mobile employees trying to do business of stocks on the way. They keep on downloading the latest stock rates as and when required through the server holding latest stock figures using one-way synchronization from server. Here they need to get the data updated from server about the shares whose prices are altered & not send its local updates back to the server. Hence one-way sync from server is very important in scenarios like this. When people send items through a courier service, they would like to trace the current status of the packet they have sent. The packet is scanned at various points using mobile devices to mark their current status and location. All this data is send to the central server as update from the client. Server updates itself with the information from various clients like this (One-way synchronization from client). Users who have send their packets can trace them by looking at the updated information on the server. Once-way synchronization from client seems to be very useful in such kind of situations.

Additional requirement comes of supporting SyncML anchor to identify previous successful synchronization state. Generally the devices undergoing synchronization tends to have low bandwidth connections to the network. This urges applications to make use of the bandwidth efficiently and transfer the data only required for it. There by users would like to send both the synchronization and initialization data together within same packet to save the network bandwidth & to-fro of additional overheads.

Identifying the above requirements to be integrated within the current state of Sync4j, analysis is carried out followed by development as explained in next section to achieve the benefits of the users over the existing problems.

The research family followed in the above research is deskwork involving literature search & its review, and analysis of data collected by others. Design of this research project is to be followed using action research. Action research helps us to study and change something we care about, resulting into enhancements in Sync4j. Techniques used to collect research data were documents and basic questionnaires through e-mail (Blaxter, et.al 2002). The research is incorporated with the software engineering processes for better results.

Software engineering encompasses of process, management, technical methods and tools. There are various software process models known as Software Development Life Cycles (SDLC). Some of SDLC are Linear Sequential Model, Prototyping Model, Rapid Application Development (RAD) Model, Incremental Model, and Spiral Model. Pressman (2001) identifies importance of SDLC as providing stability, control and organization of an activity. If they are not taken care of, activity may turn up into a disaster. Linear Sequential model or often called Waterfall Model is a linear approach, where in feedback of each phase is as an input to another phase.

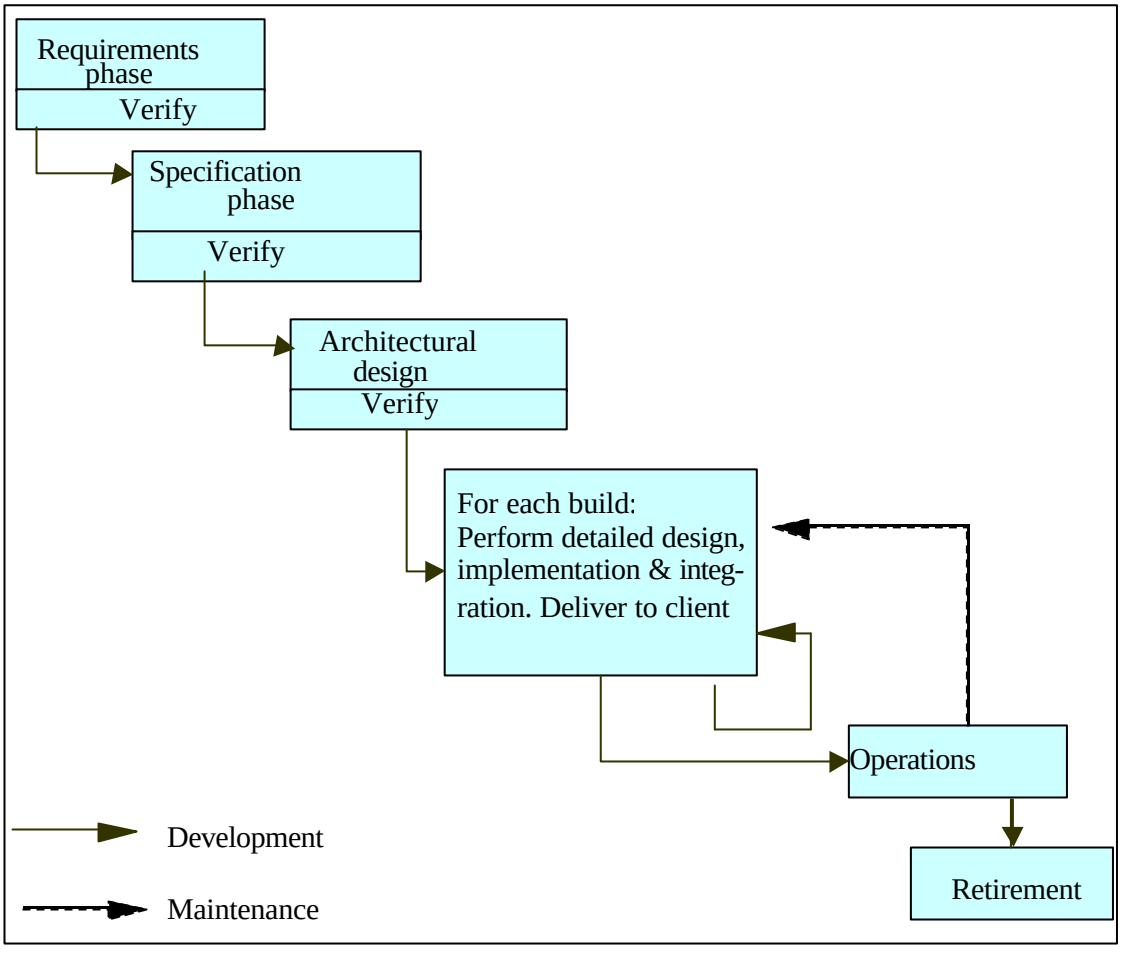


Figure 6: SDLC - Incremental Model

Incremental model is a combination of waterfall model applied in a repetitive fashion holding the iterative philosophical pattern of prototyping model. Each release within Incremental model is a mini waterfall model. The main reason for selecting Incremental model within this project is spreading of risk within itself and allows parallel working of different tasks. Risk spreads as each iteration / cycle delivers a usable product / system, current development on certain tasks can be stopped without abandoning other tasks of the project and providing better visibility of each task. (Potts, 98). Different tasks can progress simultaneously as separate waterfall model elements and see that Sync4j project is progressing as per its schedule.

This research is initially discussed with Sync4j's project in charge and university supervisor's to spot the work done and see if it was of real worth working on the researched work or required more research to be done. Receiving positive feedback to progress ahead work on this project, analysis is conducted on the features of SyncML to be incorporated within Sync4j. They are studied with the current implementation and designed to get incorporated within Sync4j. Code is developed for each feature. Following chapter looks into all this details.

Chapter 4

Analysis, Design & Development in Sync4j

"Why kick the man downstream who can't put the parts together because the parts really weren't designed properly?"

- Philip Caldwell,
American Business Executive,
CEO Ford Motors

4.
Analysis, Design and Development in Sync4j

In spite of design being good, but improper analysis will result into despair, as stated above by Philip. Sequential execution of instructions on the server and flow of information must be studied and analysed when a client tries to synchronize with the server. The analysis of this study will help in learning the flow of data within the packets, each of them carrying different tasks. The following example helps us to understand how do a client make request to the server and do server handles this.

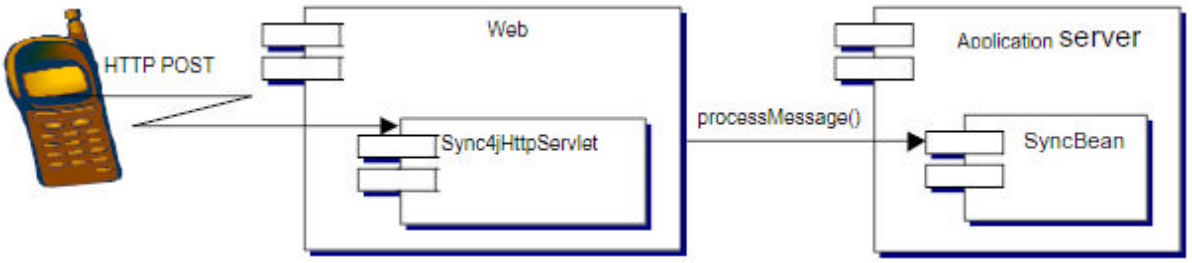


Figure 7: Example system architecture

A SyncML compliant client uses a transport protocol say HTTP to send a request for synchronization. The web server on receiving the request and fires the execution of Sync4jHttpServlet which there on handles every communication from that client for this particular session. On reading the entire message received by Sync4jHttpServlet, it invokes Enterprise Java Bean (EJB) on the SyncML server, by making a remote procedure call to *processmessage()*. Server performs the required synchronization process and prepares the response message to be sent back to the client. This response message is handed over back to Sync4jHttpServlet to forward it to the client from where it had received request for synchronization. The SyncML client can be either be a program or any kind of device.

Figure in the adjoining page shows the detailed sequence of execution of events occurring at the server. SimpleSessionHandler currently handles processing of the message received at the Server Bean through the client. SimpleSessionHandler initially prepares the environment for synchronization by setting various parameters of the server.

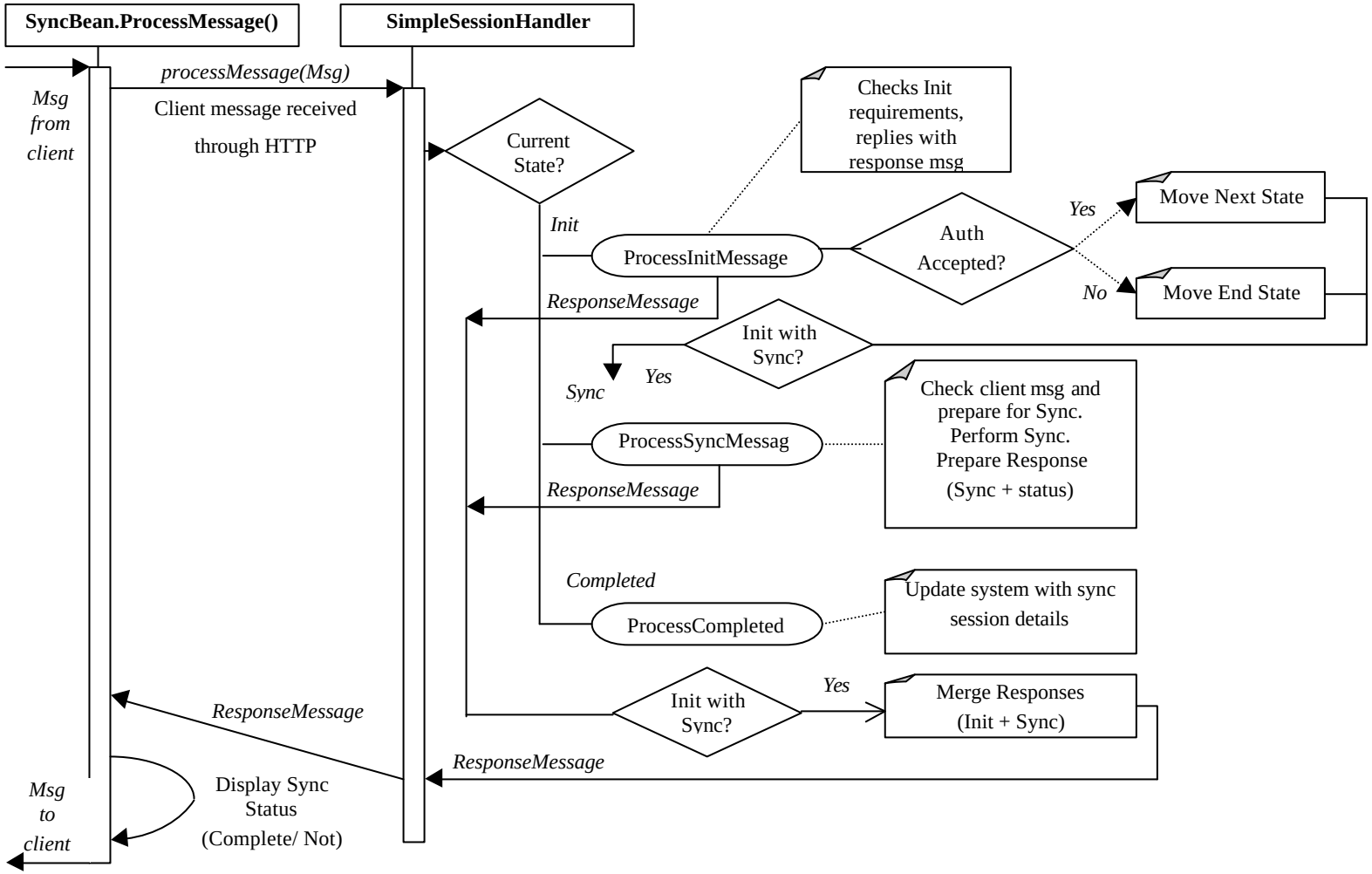


Figure 8: Flow of data in Sync4j

It begins with checking the current state of synchronization (initialization / synchronization / completed). If its initialization stage, the client message is checked for its headers, authentication and the type of synchronization requested for database(s). Server prepares a response message for the client containing status & alert information. If it was a synchronization state, clients command are studied by server format for synchronization, synchronization is carried out, and a response message for the whole process is prepared to update client with the success of request (status command) along with server modified data (sync command).

At the state of completion of synchronization, server should update its anchor to reflect the successful completion of synchronization. Finally a response message is sent from SimpleSessionHandler to SyncBean for forwarding it to the client from where server received message for synchronization. This figure also contains implementation of Sync with Initialization, which is implemented as explained in the following sections.

4.1. Analysis of SyncML & Sync4j

The previous example shows us how the server handles the client requests. We need to explore more about the sequence of packets that are exchanged for synchronization like initialization, modification, acknowledgement, mapping, etc. Two-way synchronization covers all the basic tasks of the synchronization. Hence, we analyse sequence carried within each task, giving us more information about the area, where what happens during the synchronization.

The following figure represents a MSC notation for two-way synchronization of a SyncML protocol (SyncML [c], 2002). As Sync4j implements SyncML, we first understand the logical flow of data in terms of SyncML that can be interpreted as flow of packets in Sync4j. The arrows in the figure below represent the SyncML packets. Hexagon indicates a condition that is needed to start the transaction below it. Box indicates the start of a procedure or an internal process in a device.

The process of Synchronization basically consists of two phases (a) Sync initialization and (b) Synchronization. Synchronization phase can be considered consisting of syncing data

followed by mapping of data. The purposes of sync initialization are identified as: (SyncML [c], 2002)

- To process the authentication between the client and the server on the SyncML level.
- To indicate which databases are desired to be synchronized and which protocol type is used.
- To enable the exchange of service and device capabilities.

The second phase synchronization consists of sending updates from client to server & server to client and there by getting in sync with each other. It also involves performing mapping of data on client and server using LUID – GUID concept.

During the Initialization phase, client sends the initialization packet (pkg #1) to the server. Each packet consists of Header (SyncHdr) and body (SyncBody). Within this packet, header consists of details like authentication, target location (server), version of data (verDTD) and body contains Alert command, device capabilities and service capabilities. Upon server receiving the initialization packet from client, it processes with the information provided in initialization packet like authenticating the user, checking the type of data version, format & device, type of synchronization for different databases and responds accordingly back to the client and finally completing the initialization phase.

Once the initialization phase is completed successfully, the client device starts preparing for the data that it required to be sent to the server for informing it about the changes occurred in that device after last successful synchronization. Client sends packet 3 to server with all the above prepared data within Sync commands as specified in SyncML protocol. Server analyses this data with the data stored in it with the help of Synchronization engine. The status of the modifications and other modified data from the server is send to the client from server. Client makes modifications in it as per the data received in packet 4. The status of these modifications is send back to server in packet 5. Server replies back with map acknowledgement to mark the completion of the synchronization phase. Thus in this way data is exchanged through the SyncML packets and both – client & server come in sync with each other.

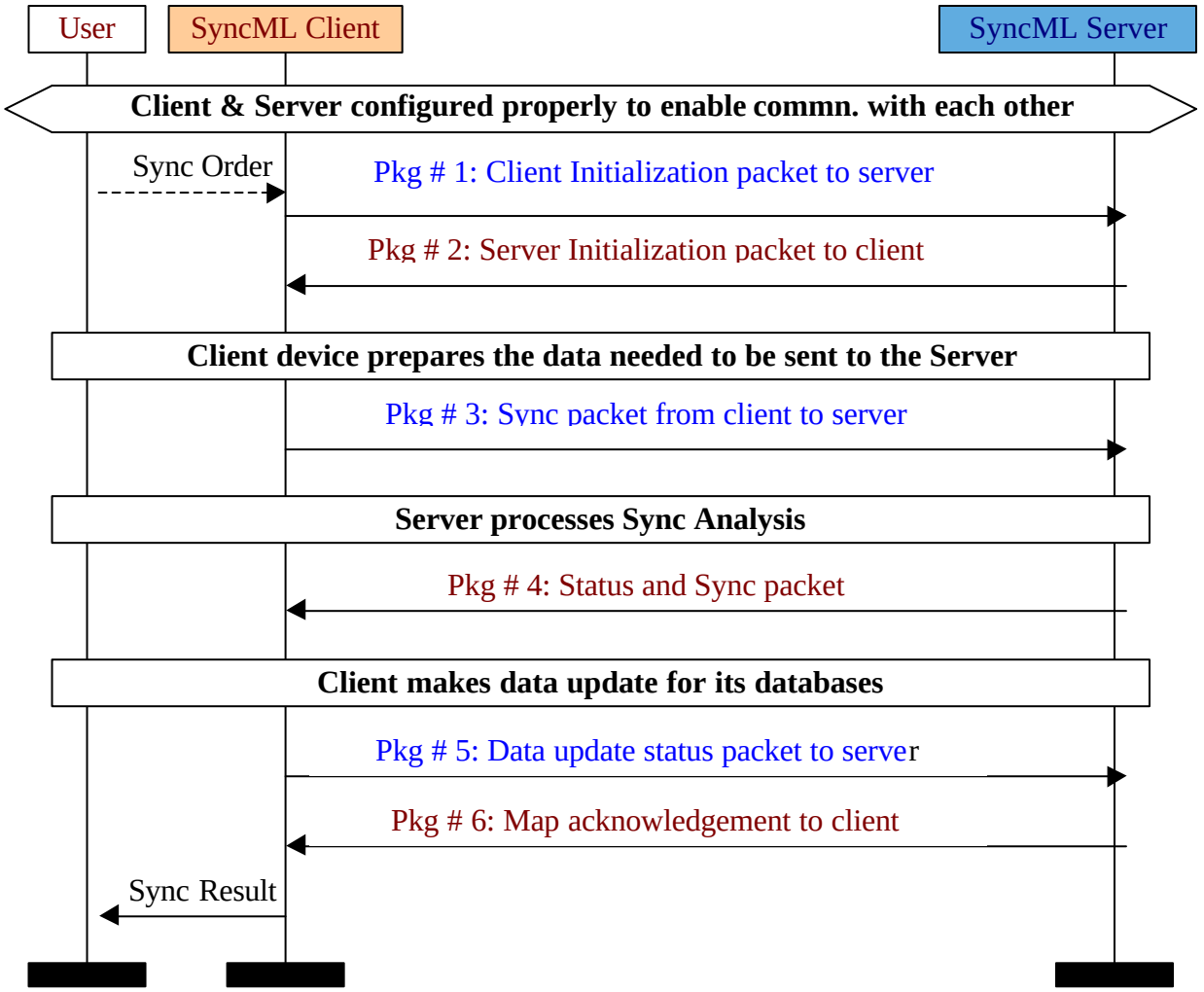


Figure 9: MSC for two-way Synchronization

Having analysed the exchange of packets between server and client, let us analyse this packets in detail to integrate it with the requirements as spotted in the research section. Following sections within this chapter analyses and designs the specified tasks within Sync4j research (see Sec 3.2).

4.2. Sync without separate Initialization

Initialization is the first phase of the synchronization process. Initialization phase is a crucial phase, as it authenticates the user (if required), provide grants to synchronize with the requested database with the type of synchronization (Slow / Fast). For more information on different types of synchronization see section 2.4. Initialization phase is depicted by packets 1 & 2 in the previous example of two-way synchronization.

4.2.1. General usage of Sync without separate Initialization

Synchronization can start without separate initialization i.e. initialization process and sync can be merged. Or in other words, initialization can be done simultaneously with sync. The major reason to do so is to decrease the number of SyncML messages to be sent over the air, there by reducing overheads of messages and make efficient use of the low network bandwidth. This feature of undergoing synchronization without separate initialization of SyncML is analysed and designed in Sync4j using SyncML protocol specifications - (SyncML[b], 2002) and (SyncML[c], 2002). The scenario as explained in the previous example gets modified in the following way. Figure 10 shows the modified version of two way synchronization with separate initialization (figure 9), briefing sync without separate initialization.

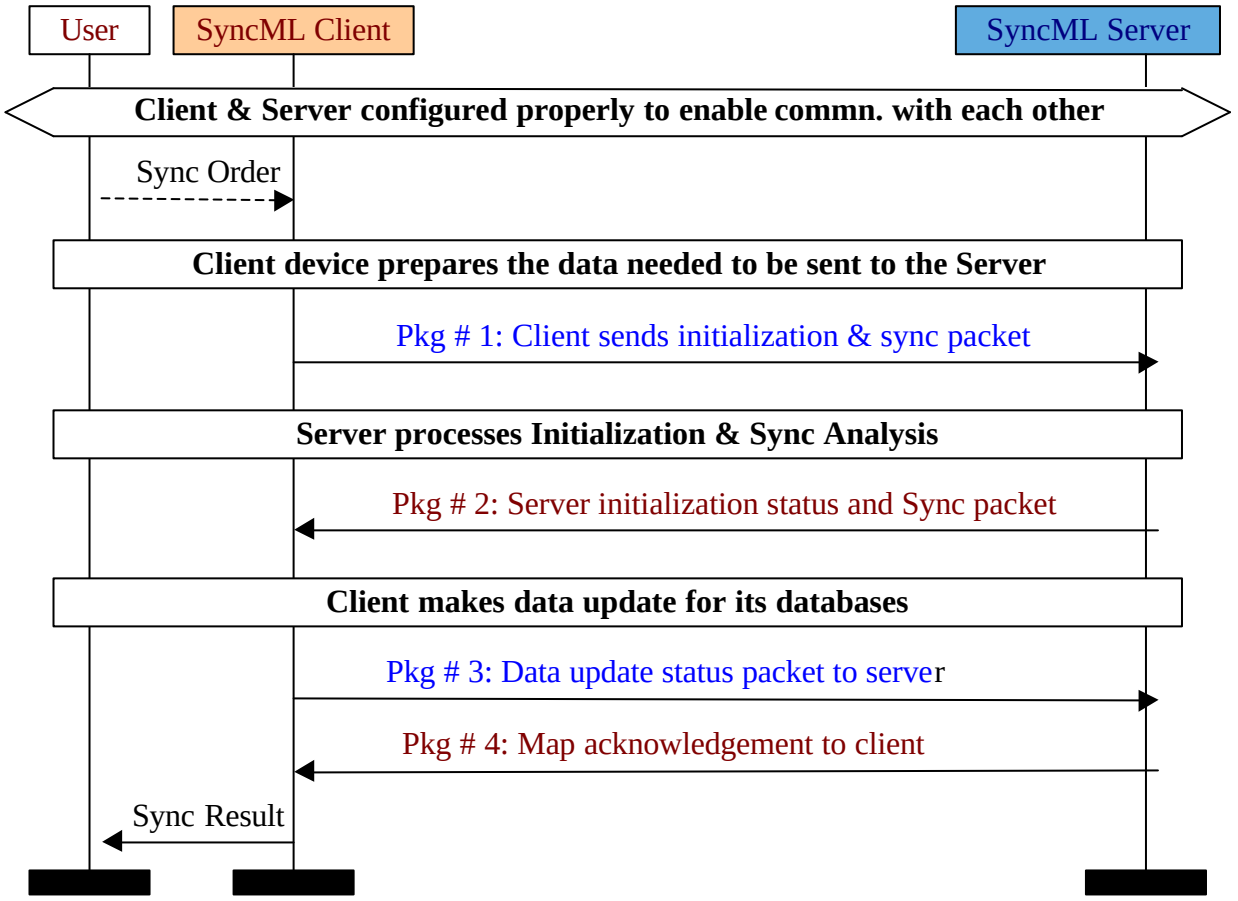


Figure 10: MSC Notation for Sync without Separate Initialization

Here the Alert command(s) from the client is sent in packet # 1 along with synchronization data, instead of packet # 3 as per figure 9. There by this packet would be containing the

initialization information like authentication, device information, data bases to be synchronized (alert commands) and its synchronization information within the sync command. This over all forms the packet # 1 containing initialization and synchronization information sent from client to server. The server completes the initialization phase by checking authentication, allowing requested type of synchronization over different databases, understanding the client device capabilities and services.

As the server has also received the data to be synchronized, it executes sync analysis through its sync engine & performs relevant updates. Server then sends update message to the client as packet # 2. This packet contains the status of authentication, type of synchronization over databases, server capabilities in response to complete the initialization phase. It also contains the data to be synchronized for the databases from the server to the client. So, packet # 2 is combination of packet # 2 and packet # 4 of previous example (figure 9).

The representation of data for the initialization packets from client to server and vice versa can be found in Chapter 4 of SyncML specifications (SyncML[c], 2002). In the following section we see the current synchronization pattern followed by Sync4j server for sync with separate initialization.

4.2.2. Current synchronization sequence in Sync4j

In this section we look into the developed code of Sync4j to understand the current working synchronization mechanism. At the moment it is supporting synchronization with separate initialization. Following steps represents the sequential execution for synchronization. Various packages referred below should be linked as sub packages of Sync4j package. Figure 8 can also be referenced for following discussion.

- All incoming client requests are received by Sync4j Web Server (*Sync4jHttpServlet*) through HTTP transport protocol.
- Sync4jHttpServlet fires a remote procedure call *processMessage()* of *SyncBean* package.
- Sessions are created for each client every time they connect for synchronization. These sessions are passed onto *SimpleSessionHandler* to handle the request received at *processMessage()*.

- *SimpleSessionHandler* checks the current status of the session and processes the message accordingly. If the Status is START, server interprets that initialization processes is to be started. If the Status is SYNCHRONIZATION, server proceeds the session ahead with the synchronization of data.
- A response message is built at the end of each process (initialization and synchronization) to reflect the status of execution of each command on the server.
- This response message is build based on the SyncML specifications. Message is converted into XML format using *processMessage().*
- Finally this response message is send back to the client through HTTP transport protocol.

Sync without separate initialization can be added into Sync4j, but adding few logical bits within the above existing structure. But what should be added? Will it affect current way of sending requests and responses?

4.2.3. Designed new Synchronization sequence in Sync4j

The current synchronization pattern of sync with separate initialization must be altered with the synchronization pattern as discussed below to make Sync4j compatible with both the scenarios of sync with and without separate initialization. Figure 11 shows the logical modifications within the flow of information to achieve the above specification with Sync4j. Blue coloured boxes in the figure represent alteration of the code within the existing ones.

This logical diagram represents the following sequence:

- *SimpleSessionHandler* during initialization phase, scans the whole received message to check if it's a message containing synchronization information? I.e. sync with initialization.
- If yes, then it sets the current state of process to SYNCHRONIZATION. But first it allows the initialization request to be processed, hence it calls *processInitMessage().* This sub routine performs all the initialization check requirements, sets databases to be synchronized, sets the type of synchronization mode on each databases (fast / slow sync) and finally prepares a response (*init_response*) message indicating various status of the commands execution requested.
- Else i.e. sync with separate initialization, current state of process is not altered.

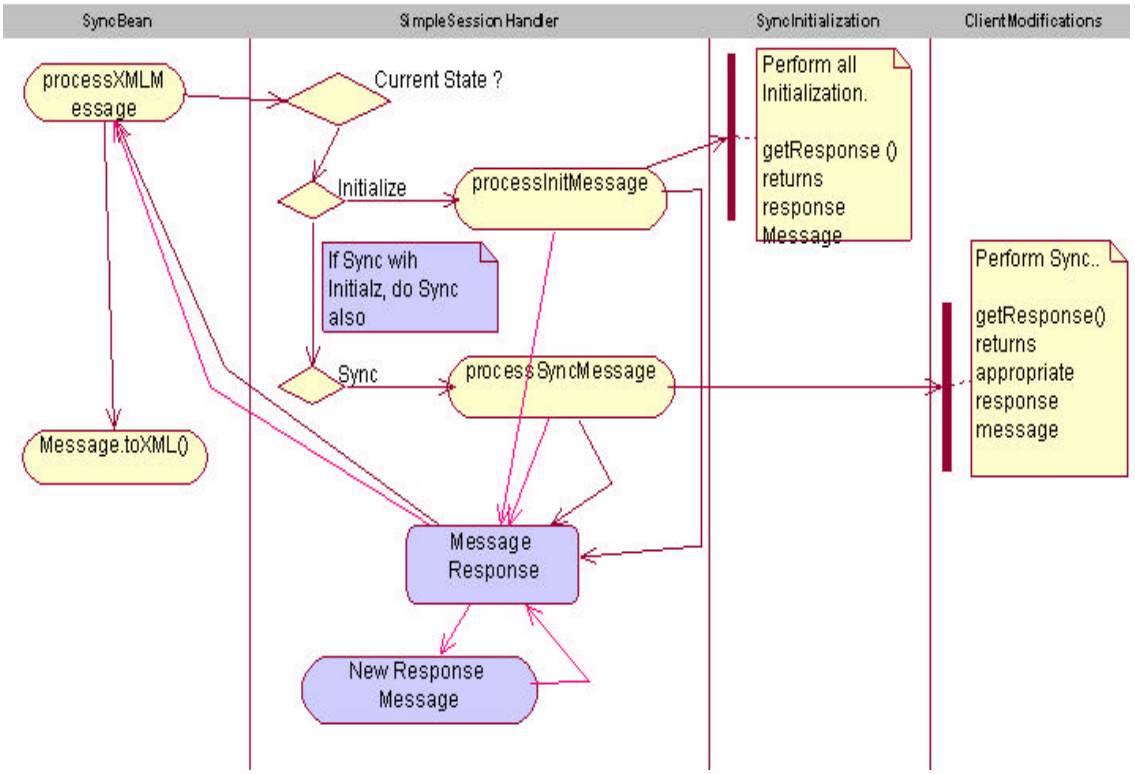


Figure 11: Design for Sync with & without separate initialization

- If the current state of process is set to SYNCHRONIZATION, the sync process is also executed sequentially. Here the synchronization information is passed to its subroutine processSyncMessage(). This sub routine performs all synchronization tasks like reflecting the modifications as presented in the <sync> element of the received message. If requested, server prepares all the updates of the server to be sent to the client, which is sent to the client in the response (*sync_response*) message along with status of sync carried out on server.
- Responses received from the initialization (*init_response*) and synchronization (*sync_response*) carried out as part of sync with initialization must be merged together to form a single message, which is send as a response message from server. Hence this two response messages are joined together to form a meaningful response message as per SyncML protocol.
- Finally, this response message is send to *processMessage()* to forward it to the client.

The above logic is inserted into the existing code, at the areas presented in the code below. Text in comments (or blue coloured text) represents the new logic within *sync4j.server.syncbean.SimpleSessionHandler*. While creating a response message from server, the final messages header should be same as that found in initialization response header as only it contains credentials which are not found in sync_response.headers. Final response body should contain response messages found in body of *init_response* and *sync_response*.

```
public Message processMessage(Message message) throws ProtocolException
      {
              - - -
              switch (currentState) {
               case STATE_ERROR:
               case STATE_START:

                    --- Process Initialization of the message ---
            // If Sync without Separate Initialization
            // set CurrentSTATE to STATE_SYNCHRONIZATION_PROCESSING
            // and proceed ahead for synchronization...

                      break;
              case STATE_INITIALIZATION_PROCESSED:
              case STATE_SYNCHRONIZATION_PROCESSING:

                   --- Process Synchronization of the message ---
                         break;
                    ---
                  }
                  ---
            //  If Sync with Initialization merge responses of
            //  Initialization & Synchronization

            Message finalResponse = null;

            if(syncWithInit == true) {
                 try {
                     finalResponse = mergeInitSyncResponse(
                                      initResponse, syncResponse);
                       } catch (RepresentationException e) {
                        throw new ProtocolException("Unexpected error", e);
                       }
              }
        return final_response;
   }
```

4.2.4. Benefits of this design

The benefits of this design can be stated as:

- It maintains the same basic sequence of code execution.
- There are no alterations in the way the initialization and synchronization information is processed or exchanged within the server.

- Response message for initialization package for sync with initialization is created by merging response messages from initialization and synchronization process.
- Message exchange between *SimpleSessionHandler* and *syncBean* is not altered, leaving the code & format of data as it is.
- Merging two features do not breach security. Synchronization on the data is performed only after the database is set by initialization phase after authenticating the user. If user is rejected, no database is set for synchronization and hence no data is send by server even if goes for synchronization task.

Let us now see the next feature to be enhanced within Sync4j as found during research (section 3.2) within the following section.

4.3. Sync Anchors

SyncML defines a concept known as Sync Anchor. Sync anchor is a way of storing information about the last Sync Event. It can be considered as a parameter that indicates the last successful synchronization session.

SyncML protocol defines synchronization anchor as "A string representing a synchronization event. The format of the string will typically be either a sequence number or an ISO 8601-formated extended representation, basic format date/time stamp." (SyncML [c], 2002).

SyncML Sync protocol specification is studied to understand more about the Sync Anchor concept and its presentation. This whole section indeed contains reference of this specification – (SyncML [c], 2002).

SyncML protocol version 1.1 has two sync anchors, Last and Next. These two anchors are always sent during the synchronization initialization phase. The last sync anchor describes the last event when the database was synchronized with this device. The Next sync anchor describes the current sync even from this device. Both the events can be represented in terms of time stamps. Both the devices (client and server) send their own anchors to each other to decide the type of synchronization to be carried over in between them.

If the device stores the Next sync anchor, it is able to compare this anchor during the next synchronization with the Last sync anchor send by another device. If both the anchors do match, it can be concluded that last sync was successful and no errors or failures has occurred in sync after that. If they do not match, the device should request a special action from another device for slow synchronization. Slow synchronization is required in this case, as the device is not aware of the successful updates send to the other side, bringing it in dilemma of sending which modifications for the other device. This enables the better utilization of the protocol itself.

Care must be taken that sync anchors must not be updated before the synchronization session is finished. The synchronization session can be considered to be finished when a device is not going to send and is not expecting to receive any SyncML message from other devices, and the synchronization was successful on the SyncML command level (i.e. no other than 200-class statuses has been returned for sync commands). Devices must also not update itself with the anchors unless the communication between the synchronizing devices is ended properly as per the transport level specifications.

As multiple devices can be synchronised with each other, the Next sync anchor on the synchronized clients (devices) must be stored until the next synchronization session in order for the devices to maximize the utility of anchors with its functionalities.

A typical sync anchor value shown below would be representing time stamp as 09:09:09 AM, 09/09/2001.

```
20010909T090909Z
```

Similarly a complete sync anchor with both the elements Last and Next can be represented in SyncML as:

```
<Anchor xmlns='syncml:metinf'>
       <Last>20010909T090909Z</Last>
       <Next>20011010T101010Z</Next>
</Anchor>
```

4.3.1. General exchange & Usage of Sync Anchors

Sync anchors of SyncML protocol enables sanity check of synchronization. Following points shows the messages where which anchors (Last, Next) are exchanged between client and server. It also highlights the way Sync anchors are used.

- Client sends its last synchronization event details to server in initialization package through Last and Next anchors.
- These anchors are sent within Alert element – requesting for synchronization of database; representing anchors related to this specific database specified within the alert element. Hence, if multiple Alert elements are present, then there would be multiple anchors within the initialization.
- Server compares the Last anchor of the requested database for this user with the Last anchor received from the client for that database. If both anchors are found to be same, it is interpreted that last synchronization between them (client & server) was successful and no error or failures had occurred since then in between them. Then server MUST allow the current process of synchronization with the requested type of synchronization from the client. If the anchors don't match, then server MUST request for either REFRESH {Status – 508} or SLOW SYNC {Alert - 201}.
- Server may wish to store the clients anchor for current session within itself temporarily for future reference within the same session.
- Server MUST respond with the same client anchor (Next) within Status element of Alert command of initialization response to acknowledge the response, if the client requests it.
- Server MUST sent current sync anchors (Last, Next) of the server for each Alert element of each database to be synchronized.
- Client acknowledges the alert command of server with its Status element containing the Next sync anchor of the server.
- On the successful completion of synchronization at SyncML and transport specification level, devices MUST update their sync anchors and not before it.

How far is Sync Anchors supported within Sync4j? Does it follow the above general exchange pattern? Let us see following sections for answers of these queries.

4.3.2. Current Sync Anchor support in Sync4j

Sync Anchors (Last, Next) are currently partially implemented in Sync4j. Let us analyse about it seeing up to what extend is their current support within Sync4j? As Sync4j is currently focusing on server side i.e. Sync4j as SyncML server, we only look into the server perspective (referenced as task from the above points) of Sync Anchors within following points.

a) Server currently is able to interpret the sync anchors with the Alert command for the databases. (Task c). Server does store them temporarily within its store for further reference (Task d). But the major concern is server is currently not supporting the main task of deciding the type of synchronization based on this anchors available.
b) Server is sending the status of the alert command, but does not contain the Next sync anchor (Task e).
c) Server is also sending Last and Next sync anchors within the alert command (Task f).
d) Server also updates its anchors upon successful completion of synchronization (Task h).

4.3.3. Design for full Sync Anchor support in Sync4j

The current support of Sync Anchors within Sync4j must be updated with the following design pattern in order to say Sync4j – A SyncML server fully implements Sync Anchors within it. Points a & b of the current sync anchor support in Sync4j must be designed and implemented. We don't need to bother about c and d, as they are already implemented.

Design of point a, involves addition of two concepts to Sync4j. First one is to retrieve the sync anchors of the server and second is to compare the sync anchors for deciding the type of synchronization to be permitted. The first concept involves retrieval of anchors based on parameters like the database, user and sync device. This task is achieved by using a concept of Persistent Store available in Sync4j. Persistent store is explained in detail in Overview of Sync4j (Chapter 3).

Once the synchronization is carried out successfully, the server MUST update its sync anchors. Sync4j implements this by storing Sync Anchors in its Persistent store. We retrieve the value of the last synchronization event for this database, user and device from the store

into a Last Time stamp object. Following code shows how the data is retrieved from the server's persistent store.

```
// ps = persistent Store
// serverLast = Last Time stamp object

try {
            ps.read(serverLast);
    } catch(PersistentStoreException pse) {
           log.severe("Unable to retrieve timestamp from store");
           log.throwing(getClass().getName(), "getSyncModeUsingAnchor", pse);
   }
```

The second concept involves comparison of this retrieved time stamp (server last time stamp) with the client time stamp. As per point c of Sec 4.3.1, based on the comparison result the synchronization mode of the database is decided. Following code shows this implementation.

```
// Check Client & server anchors to determine the type of Sync
int syncType = AlertCode.SLOW; // set default to SLOW

if((serverLast.last).equals(clientLast.last)){
           syncType = dbSync.getMethod();  // set client requested sync type
           log.fine("Using Client requested Alert Code ->"+ syncType);
 }
 else{
           log.info("Server forcing for SLOW Sync");
 }
```

The above codes are inserted into a new java file *sync4j.framework.protocol.getSyncMode.java.* The second task of providing Next Sync Anchor within the Alert command from server to client (point b of Current state of Sync anchors in Sync4j) is achieved by adding the anchor within the command. The code is traced into to find where this command is created. The code is there by altered in *sync4j.framework.protocol.syncInitialization.getResponse()* as shown below.

```
//     Within Response of Alert, Item must contain the NEXT Anchor.
    if (clientCommands[i] instanceof AlertCommand) {
        items = new Item[1];
        for(int j=0; (databases != null) && (j<databases.length) ; ++j){
```

```
if((databases[j].getTarget().getURI()).equals(targetRefs[0].getValue())){
                    AnchorMetaContent alertAnchor = new AnchorMetaContent(
                                null,                   // last anchor
                                databases[j].getNext()  // next anchor
                    );
                    items[0] = new Item(
                                null,  // target
                                null,  // source
                                null , // meta
                                new Data(alertAnchor.toXML())
                    );
               }
          }
     }
```

4.3.4. Benefits of this design

The major benefits of this design can be drawn as: (a) Design of a generic class getSyncMode implementing functions to retrieve the syncMode of the database using the anchor by just passing database as an argument. Similarly other functions to retrieve the sync mode using different parameters can be implemented over here. There by bringing similar functionalities in one place. (b) Sync anchors where ever required are referred to the temporary stored anchor – the one received from the client. Hence, code is re-used.

4.4. One-way Synchronization from Server

There are different types of synchronizations, as stated in section 2.4. One-way synchronization is a special kind of synchronization where in updates is send only from one side to another. If it is one-way synchronization from Server, all updates of server are sent to client and no updates from client are sent to server. Similarly if its one-way synchronization from client, client sends all its updates to server with no updates from server to client.

4.4.1. General usage of One-way synchronization from server

One way synchronization is very helpful in cases, where updates taken place on one side of the synchronization devices needs to be reflected other with its changes and not expecting other device to send any of its modification. Let us take an example to make this applicability more clear.

> *A busy mobile businessman carries a mobile device capable of undergoing synchronization with the office synchronization machine (server). His secretary keeps on updating his calendar with modified schedules and stores*

relevant information about the schedule. If this mobile device has undergone successful synchronization previously, then the business man only requires one-way synchronization from server to receive all the updates about his schedule on his mobile device, without need to send any modifications from his device to server.

The style of exchange of packets between the client and server remains the same as two-way synchronization with some modifications as explained below. This whole way of synchronization is analysed based on the SyncML Sync Protocol & hence may not be referenced everywhere within this section (SyncML [c], 2002).

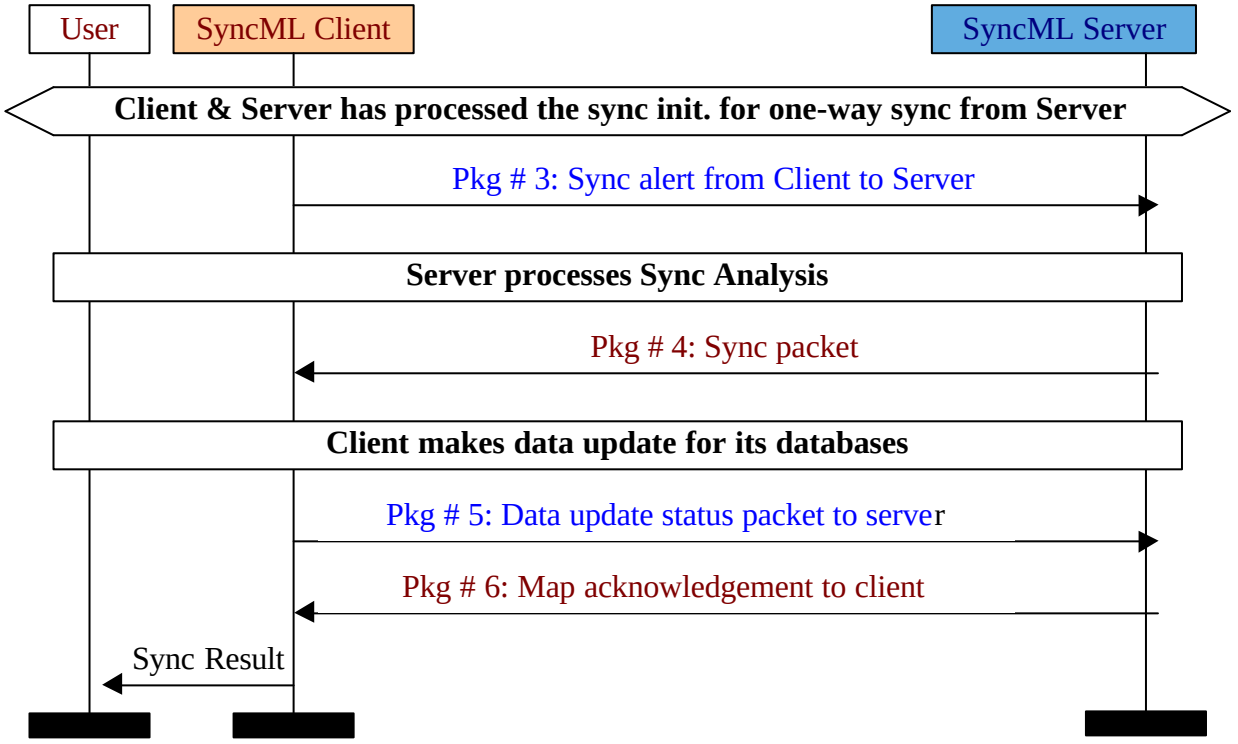


Figure 12: MSC Notation for one-way sync from server only

This MSC notation can be interpreted as explained in points below:

- Initialization in this case is same as carried out in two-way synchronization, but the synchronization requested would be one-way sync from Server i.e. Alert code 204.

- Packet # 3 contains only Sync alert command requesting for updates from server. It does not send any of the client modifications to server. Hence the sync element MUST be empty.
- Server processes sync analysis and sends the sync package to client containing modifications in packet # 4.
- Client updates itself with the received modifications from the server. Status for each updates is sent to the server through packet #5.
- Finally at the end, server sends a map acknowledgement to the client to mark the completion of synchronization (same as two-way sync).

The packet flow presented above is one SyncML session; hence all messages have the same SyncML session ID.

4.4.2. Current One-way Sync from Server support in Sync4j

Currently Sync4j is only supporting two-way synchronization. There is no support for one-way synchronization from either server or client. As per our research work, they must be added within Sync4j to enhance its utility. Let us start begin with looking at the design and development of this feature in following sections.

4.4.3. Design of One-way Sync from Server support in Sync4j

As there is no current support for this feature of SyncML within Sync4j, it must be designed starting from the scratch and integrating with the existing code. By analysing the modifications required in the execution pattern in comparison with the standard two-way synchronization, the one already implemented in Sync4j, it is found that a check must be provided to identify and process this type of synchronization. Check must verify that the sync packet # 3 received from client to server must contain only Sync alert & no updates from client to server. If there are any updates from client, they are filtered out. Rest of the other things within one-way & two-way remains the same. Hence to achieve this task, above check must be included within the existing code to enable the feature of One-way Sync from Server. Following modification in code is carried out to reflect this added specification.

```
private void checkSyncCommand() throws ProtocolException {
    List list = ProtocolUtil.filterCommands(clientCommands    ,
                                            SyncCommand.class);
```

```
            ---

            // build up a list of commands to be executed on Server
            // Exclude for ONE_WAY_SYNC_SERVER
            GetSyncMode dbSyncMode = new GetSyncMode();

            dbSyncMode.setDatabases(databases);

            SyncCommand[] tempCommands = new SyncCommand[list.size()];

            for (int i=0, j=0; i<list.size(); i++) {
                 if(dbSyncMode.getSyncModeUsingDb(
                                ((SyncCommand)list.get(i)).getTarget(),
                                ((SyncCommand)list.get(i)).getSource()
                   ) != AlertCode.ONE_WAY_FROM_SERVER){

                 clientPermittedCmds.add((SyncCommand)(list.get(i)));
                 tempCommands[j++] = ((SyncCommand)(list.get(i)));
                }
             }
            clientSyncCommands = new SyncCommand[tempCommands.length];
            clientSyncCommands = tempCommands;
    }
```

The above shown code is altered within *sync4j.framework.protocol.clientModifications.java* file. What the above code does is checks the synchronization mode for the database containing modifications from client to server with the alert code for One-way sync from Server. If it matches, those commands are not added to the client sync commands to be executed on the server. Otherwise they are appended to client sync commands. This client sync commands are then send to sync engine to perform synchronization with the server database.

An additional piece of generic code was also required to be added for determining the type of synchronization for that particular database. This piece of code takes in argument as server database and client database, and returns the alert code (Sync type) that is to be carried out on this database as agreed between them within the initialization phase.

```
    /*
     * Function to get the Synchronization mode of the database
     * @param TargetRef    Targeted database   - generally Server
     * @param SourceRef    Source database     - genearlly Client
     * @return AlertCode of the Sync carried out on this database pair.
     */

    public int getSyncModeUsingDb(Target tRef, Source sRef) {

        for(int i=0; (dbs!= null) && (i < dbs.length) ; i++) {
            if ( (dbs[i].getTarget().getURI()).equals(tRef.getURI()) &&
                 (dbs[i].getSource().getURI()).equals(sRef.getURI())) {
                return(dbs[i].getMethod());
            }
```

```
        }
        // Forcing Sync Mode to SLOW - 201

        return(AlertCode.SLOW);
    }
```

4.4.4. Benefits of this design

The benefits of this design can be drawn as:

- The feature is added into the existing structural code by inserting the required specification check and not altering the current sequence of code execution. Hence care is taken to re-use the available code and let the flow of information within the architecture be simple.

- Existing generic code is extended further. A new generic code is amended to the previously developed generic code to retrieve anchor within *sync4j.framework.protocol.getSyncMode* class.

Thus in overall, the sequence of execution of the code is unaltered leaving it intact making the flow of information within the architecture to be simple. The researched specifications are amended to Sync4j by adding few bytes of code into it after undergoing basic analysis, design and development. The modified code can be found in attached disc. Care has been taken through out the whole phase to re-use the existing available code and develop generic code that can provide useful results in further development.

With the development of One-Way Synchronization from Server, it found essential to add One-Way Synchronization from Client as well within Sync4j. How it is achieved? Let us see following section.

4.5. One-way Synchronization from Client

There are different types of synchronizations, as stated in section 2.4. One way synchronization is a special kind of synchronization where in updates is send only from one side to another. Here in one way synchronization from client, client sends all its updates to server. Server analyses the data and get itself updated with the data received from client. No modification occurred on server are reflected to the client.

4.5.1. General usage of One-way synchronization from client

One-way synchronization from client is very helpful in cases, where majority of updates taken place on one side of the synchronization devices (client) and needs to be reflected other (server) with its changes and not expecting other device to send any of its modification. Let us take an example to make this applicability more clear.

> *An express parcel service company eg. DHL, TNT … is committed to provide timely delivery of its packages. It would also like to inform its packet sending customers to track the current status of the packet, they have sent. Employees of this company can identify and store information about such packets at various intermediate stage(s) of transit using mobile devices. Central database server can be updated using one-way synchronization from client (mobile device) to reflect all the updates of the current status of packets to the server. Thus server holds latest trace of the packet in transit. Users can any time check central server, which will be showing the current status of the packet. In this case, there is no need for client to get update from server about other packets. Hence, one-way sync from client perfectly suits this case.*

The pattern of exchange of packets between the client and server remains the same as two-way synchronization with some modifications as explained below. This whole way of synchronization is analysed based on the SyncML Sync Protocol & hence may not be referenced everywhere within this section (SyncML [c], 2002).

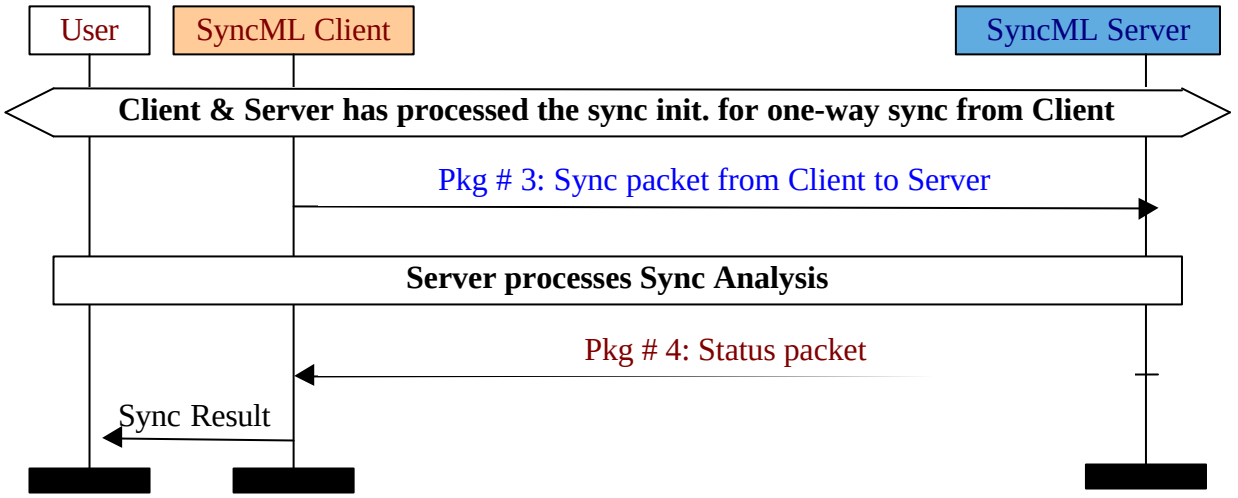


Figure 13: MSC Notation for one-way sync from client only

This MSC notation can be interpreted as highlighted in points below:

- Initialization in this case is same as carried out in two-way synchronization, but the synchronization requested would be one-way sync from Client i.e. Alert code 202.
- Packet # 3 contains Sync message from client to server. It holds all sync command containing updates occurred at client and to be reflected at server.
- Server processes sync analysis and sends the sync status package to client containing status of executed commands in packet # 4.
- Server updates itself with the received modifications from the client.

The packet flow presented above is one SyncML session, hence all messages have the same SyncML session ID. How this feature is inserted within current set of execution & features of Sync4j is explained within following sections.

4.5.2. Current One-way Sync from Server support in Sync4j

When started analysing Sync4j, it only supported two-way synchronization. There is no support for one-way synchronization from either server or client. One-way synchronization from server is added as defined in earlier section 4.4. One-way sync from client is achieved through following design.

4.5.3. Design of One-way Sync from Client support in Sync4j

As there is no support for this feature of SyncML within Sync4j, it must be designed starting from the scratch and integrated with the existed code. By analysing the modifications required in the execution pattern in comparison with the standard two-way synchronization, the one already implemented in Sync4j, it is found that a check must be integrated for this identified type of synchronization and processing accordingly. Check must verify that the sync packet # 4 sent from server to client must only contain the status of the sync commands received from client and no Sync commands to client. Sync Command is only sent by client to server in packet # 3 containing all its modified elements within its database to be reflected on the server. Rest of the other things within one-way & two-way remains the same. Hence to achieve this task, above check must be included within the existing code to enable the feature of One-way Sync from Client. Following block shows modification in code which is carried out to reflect this added specification.

```
    private List processModifications(ClientModifications modifications)
            throws Sync4jException {
              ---
        // SyncCommands send back to the client
        // No sync send to client if ONE_WAY_FROM_CLIENT

        SyncOperation[] operations = syncEngine.getSyncOperations();

        ItemizedCommand[] commands = null;

        for (int i = 0; ((syncCommands != null) && (i <
                                syncCommands.length)); ++i)
           {
            target = syncCommands[i].getTarget();

            //
            // A Sync command can be empty...
            //

            if (target == null) {
                continue;
            }
            uri = target.getURI();

            for (int j = 0; ((dbs != null) && (j < dbs.length)); ++j) {

                if (dbs[j].getName().equals(uri) &&
                    dbs[j].getMethod() != AlertCode.ONE_WAY_FROM_CLIENT ){

                    // NO SYNC from SERVER for ONE_WAY_FROM_CLIENT

                    commands = syncEngine.operationsToCommands(
                                   (ClientMapping)clientMappings.get(uri),
                                   operations,
                                   uri
                               );

                    if ((commands != null) && (commands.length > 0)) {
                        responseCommands.add(
                                new SyncCommand(
                                        cmdIdGenerator.next(),
                                        false,
                                        null,
                    ProtocolUtil.source2Target(syncCommands[i].getSource()),
                    ProtocolUtil.target2Source(syncCommands[i].getTarget()),
                                        null,
                                        commands
                                )
                        );
                    }
                }
            } // next j
        } // next i
       return responseCommands;
    }
```

The above shown code is altered within *sync4j.server.syncbean.session.SimpleSessionHandler.java* file. The functionality of the bove code is to check the synchronization mode of the database and block all sync commands to be sent for the client if the Sync Mode is One-Way from Client. Otherwise

sync command is prepared by SimpleSessionHandler for the client containing server modifications. Sync mode of the database is retrieved from the mode stored along with the database object. The mode of the database synchronization is stored during the initialization phase while confirming the type of synchronization over it.

4.5.4. Benefits of this design

The benefits of this design can be drawn as:

- The feature is quickly added into the existing structural code by inserting the required specification check while preparing the response message and not altering the current sequence of code execution. Hence care is taken to maintain the flow of information within the architecture.
- Re-use of the data available in the data objects of the class. (eg. database.getMethod() to retrive the synchronization method decided for that particular database).

Thus in overall, the sequence of execution of the code is unaltered leaving it intact; making the flow of information within the architecture even simpler. The researched specifications are amended to Sync4j by adding few bytes of code into it after undergoing basic analysis, design and development. The modified code can be found in attached disc.

All the above features are now integrated within the Sync4j. The code must be tested for verifying its integrity with the existing code to find out unexpected responses occurring during the real time behaviour of the system. Extensive tests are carried out on Sync4j to explore behaviour of Sync4j in different conditions. Results are received for this test. Does this test gives results as per expectation from the developed code? Do Sync4j behave abnormally in certain environments? Let us find out more about all this as explained in detail within following chapter of Test and results.

Chapter 5

Tests and Results

"Success is not the result of spontaneous combustion; you must set yourself on the fire first"

– Reggie Leach, Canadian Hockey Player

5.
Test and Results

Test means a trail-run through of a process or on equipment to find out how it works. Institute of Electrical and Electronics Engineers (IEEE) standard body defines testing as: (IEEE, 90)

> *"The process of operating a system or component under specified conditions, observing, or recording the results and making an evaluation of some aspects of the system or component"*

Developed work cannot be stated as perfect product from the requirements, unless product is tested against it. Results of this test are evaluated in relation to the expectation of the developed product based on the requirements. Similarly the work produced within Sync4j is tested to check the proper working of the logic implementation stated in above chapters. According to McGregor (2001) testing process includes:

- Test plans
- Test cases and data sets
- Test software and scripts
- Test reports.

The test plan involves testing of the Sync4j server with its current capabilities support in co-relation with the SyncML specifications. Test cases are designed with different parameters to test the working of the server. Server is also being tested to check its compliance with the SyncML specifications. Data sheets are prepared with the SyncML specifications. Within the test cases, following things are taken into consideration. Finally the test reports are prepared by comparing the response received from the server with the expected response.

- Synchronization with separate initialization
- Synchronization without separate initialization
- Server responses to the client's request

- Server status messages for clients commands
- Server sync messages
- Server mapping of database

The researched features are now integrated within Sync4j by analysing, designing and developing code for each of them. It is helpful to state the development and the testing environment, where the whole work has been done. This can help getting the knowledge of the tools and the software(s) used for the development and testing. Following section sees everything in brief.

5.1. Development & test environment

In this section we see an overview of the development environment used for enhancements within Sync4j. More details about each of them can be found at their official web sites as stated in brackets along with the tool / package name. Sync4j is purely a java developed application, built using XML, cygwin suite (`http://www.cygwin.com`), Jakarta Apache Ant [`http://jakarta.apache.org/ant`], Jakarta Apache cactus [`http://jakarta.apache.org/cactus`], Jdom [`http://www.jdom.org`], Junit [`http://www.junit.org`] and PostGre Sql [`http://www.postgresql.com`]. Sun's Java Development Kit Version 1.4 is used for the current development [`http://www.java.sun.com/j2sdk`]. Sync4j server is locally deployed on J2EE server (Java 2 Enterprise Edition, [`http://www.java.sun.com/j2ee`]. The developed application is hosted on Sourceforge.net [`http://www.sourceforge.net`].

Cygwin is a unix environment for windows. It consists of two parts, (a) a dll (cygwin1.dll) providing all major functionality of UNIX with its API. (b) Collection of tools available in UNIX with UNIX / Linux look and feel (Cygwin, 2002). Apache Ant is a java based built tool similar to make in unix, but without wrinkles attached in make (Ant, 2002). Cactus helps in easing the testing of server side java code like Servlets, EJBs, Filters, etc. (Cactus, 2002). JDOM is a java-centric and java-optimized for access to robust light means of reading and writing to XML (Jdom, 2002). Junit is regression testing framework to enhance the unit tests in Java (Junit, 2002). PostGre SQL is the open source database software used in sync4j to store information about server activity & synchronization (PostGre Sql, 2002).

Concurrent Version System (CVS) is used to retrieve and update the source code onto sourceforge site for Sync4j [`http://sync4j.sourceforge.net`].

5.2. Working of Test Cases

In this section, we try to explore the working of the test cases. There is a fully automated test code, where in we specify all the test cases to be tested. This code tests the cases and reports the result (failure or success). If it is success, subsequent test case defined in test code is executed for testing. But if failure is encountered, i.e. an error occurred in comparison between the expected output and received output, testing stops. Hence, here we look into how a single test case is executed with the help of following diagram.

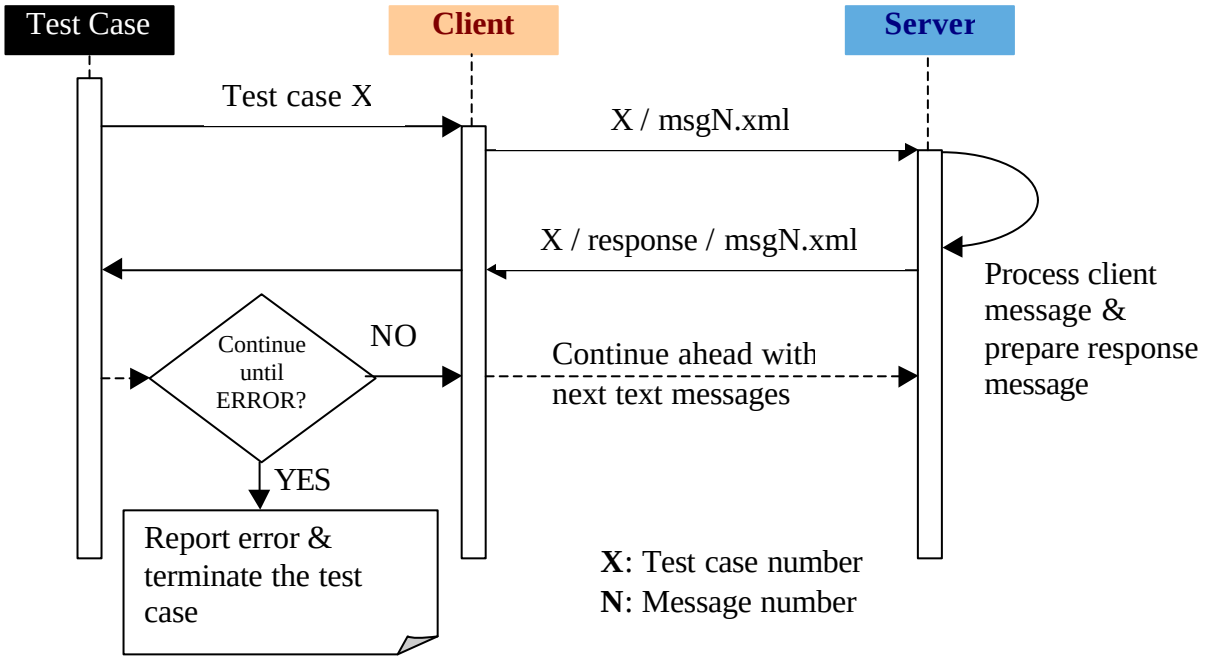


Figure 14: Working of Test Suite

The test suite executes the test cases to see whether it complies with the working of the Sync4j as expected. Test Case X represents the X test case of the test matrix – explained in next section. Each test case is represented by a test directory consisting of 3 sub directories– response, reference and error. Response, reference and error directory contains responses messages from server, reference messages build up for check with responses and any error generated respectively. All the messages sent by client are named as msg1.xml, msg2.xml ... msgN.xml and are within test X directory.

Execution of an individual test case starts with sending first message from client to server (msg1.xml). Server processes this message and prepares a response message for the client, which is msg1.xml. This message is stored in X / response directory as msg1.xml. Test suite compares this response with the expected response (msg1.xml in X / reference directory). If both the messages match with each other, test suite proceeds ahead with testing of subsequent messages. If some difference is found, an error report is created along with the location of difference in message and test suite stops further tests. This error report is stored as X / error / msgN.xml. Next section explores the numerous test cases that are built as a matrix to test Sync4j thoroughly with the SyncML specifications.

5.3. Test Case Matrix

A matrix of test cases is prepared to allow different parameters to be considered within various cases. Building of matrix, allows easy management of test cases with identifying the properties to be checked in the test case itself. It allows easy identification of the message from the test case name and number. For example, WOSI indicates Sync without Separate Initialization test case. WOSIA indicates Synchronization without Separate initialization along with Authentication. Each test case involves designing up a request using SyncML client specifications, response expected from a SyncML compliant sync server, reply to server response in SyncML format there by completing the full cycle of SyncML synchronization. Following table shows the test case matrix.

Test Code	Data	Client Capabilities	Server Capabilities	One-Way		Two-Way	Authentication
				C2S	S2C		
WOSI0001							
WOSI0002		√					
WOSI0003		√	√				
WOSI0004	√	√	√		√		
WOSI0005	√	√	√	√			
WOSI0006	√	√			√		
WOSI0007	√	√		√			
WOSI0008	√				√		
WOSI0009	√			√			

WOSI0010	√	√				√	
WOSI0011	√	√	√			√	
WOSI0012	√					√	
WOSIA0001							√
WOSIA0002		√					√
WOSIA0003		√	√				√
WOSIA0004	√	√	√		√		√
WOSIA0005	√	√	√	√			√
WOSIA0006	√	√			√		√
WOSIA0007	√	√		√			√
WOSIA0008	√				√		√
WOSIA0009	√			√			√
WOSIA0010	√	√				√	√
WOSIA0011	√	√	√			√	√
WOSIA0012	√					√	√

Table 1: Test case matrix for Sync without Separate Initialization

5.4. Test Scenario

Let us now look into one of the test case and try to understand how this test cases work. Consider test case WOSI0005; WOSI implies synchronization without separate initialization, number 0005 indicates test case number 5 and the test table matrix tells that this test case has data, client capabilities, server capabilities and one-way synchronization from client to server represented by color block.

Client sends an initialization packet to the server with its initialization data like client capabilities and alert command specifying the request type of synchronization. This is packet number 1 of the MSC notation shown in figure 10. Following code shows this message from client to Sync4j SyncML server as an XML format containing data as per SyncML specifications. The header of the message is within the <SyncHdr> tag. It contains all the information like SyncML version, server address and client's credential. <SyncBody> contains all the information about the initialization like alert command for the type of synchronization (202 for one-way sync from client to server). Client can send all its

capabilities through SyncML's <put> tag. Client capabilities like Device info can be provided within <DevInf>, type of data within the <Type>, memory available <Mem>, etc. can be provided as sub-elements of <put>.

```
<!—-
      Sync4j Test suit
      ================

      TEST: WOSI0005 Sync without Separate Initialization
      Data:         Yes


      Client Cap:   Yes
      Server Cap:   Yes
      Syc:          Yes (C2S)
      Auth:         No
-->

<SyncML>
<SyncHdr>
<VerDTD>1.1</VerDTD>
<VerProto>SyncML/1.1</VerProto>
<SessionID>12345678</SessionID>
<MsgID>1</MsgID>
<Target><LocURI>sync.sync4j.org</LocURI></Target>
<Source><LocURI>Sync4jTest</LocURI></Source>
</SyncHdr>
<SyncBody>
<Alert>
       <CmdID>1</CmdID>
       <Data>202</Data> <!—202 for C2S -->
       <Item>
              <Target><LocURI>test</LocURI></Target>
              <Source><LocURI>test</LocURI></Source>
              <Meta>
                     <Anchor xmlns=”syncml:metinf”>
                     <Last>234</Last>
                     <Next>276</Next>
                     </Anchor>
              </Meta>
       </Item>
</Alert>

<Put> <!—Sending Client capabilities -->
       <CmdID>2</CmdID>
       <Meta><Type xmlns='syncml:metinf'>text/plain</Type></Meta>
       <Item>
              <Source><LocURI>./devinf11</LocURI></Source>
              <Data>Clients configuration…
              </Data>
       </Item>
</Put>

<Get> <!—- Requesting Server capabilities -->
       <CmdID>3</CmdID>
       <Meta><Type xmlns='syncml:metinf'>text/plain</Type></Meta>
       <Item>
              <Target><LocURI>./devinf11</LocURI></Target>
       </Item>
</Get>
<Sync> <!—- Synchronization data from client to server -->
```

```
<CmdID>4</CmdID>
       <Target><LocURI>test</LocURI></Target>
       <Source><LocURI>test</LocURI></Source>

<Replace>
       <CmdID>5</CmdID>
       <Meta><Type xmlns='syncml:metinf'>text/plain</Type></Meta>
       <Item>
              <Source><LocURI>test1.txt</LocURI></Source>
              <Data>Replace received from client.</Data>
       </Item>
</Replace>

<Delete>
       <CmdID>6</CmdID>
       <Meta><Type xmlns='syncml:metinf'>text/plain</Type></Meta>
       <Item>
              <Source><LocURI>test2.txt</LocURI></Source>
       </Item>
</Delete>
</Sync>

<Final/>
</SyncBody>
</SyncML>
```

Server receives the client request and processes them. First of all, it tries to identify the user with its credential in order to check the privileges for the database to be synchronized. In this case, no user credentials are provided, hence server identifies client as a guest. A principal is assigned to this client based on (User + Device), which indeed gives the correct information about the previous synchronization by the guest user (or credited user) through this device. More information about the concept involved in principal can be found in Sync4j architecture (Chapter 3). After identifying the principal of user, its last synchronization information is retrieved with the help of anchors provided by the client. These anchors are then compared with the server anchors to determine the type of sync to be carried out for this session, as explained in Sync Anchors (Chapter 4). This particular test case undergoes one-way sync from client to server, which follows the steps explained in detail within Chapter 4. Server is requested for its capabilities through the <get> command, which is replied through the <Results> command. Server processes the clients sync commands and returns status of each command executed in <status> tag. Following table shows the expected response from the server to the client.

```
<SyncML>
<SyncHdr>
       <VerDTD>1.1</VerDTD>
       <VerProto>SyncML/1.1</VerProto>
```

```
        <SessionID>$(IGNORE)</SessionID>
        <MsgID>1</MsgID>
        <Target><LocURI>Sync4jTest</LocURI></Target>
        <Source><LocURI>sync.sync4j.org</LocURI></Source>
        <RespURI>$(IGNORE)</RespURI>
</SyncHdr>
<SyncBody>

<Status>
        <CmdID>1</CmdID>
        <MsgRef>1</MsgRef>
        <CmdRef>0</CmdRef>
        <Cmd>SyncHdr</Cmd>
        <TargetRef>sync.sync4j.org</TargetRef>
        <SourceRef>Sync4jTest</SourceRef>
        <Data>212</Data>
</Status>

<Status>
        <CmdID>2</CmdID>
        <MsgRef>1</MsgRef>


        <CmdRef>1</CmdRef>
        <Cmd>Alert</Cmd>
        <TargetRef>test</TargetRef>
        <SourceRef>test</SourceRef>
        <Data>200</Data>
        <Item>
                <Data><Anchor xmlns='syncml:metinf'>
                        <Next>276</Next>
                        </Anchor>
                </Data>
        </Item>
</Status>

<Status>
        <CmdID>3</CmdID>
        <MsgRef>1</MsgRef>
        <CmdRef>2</CmdRef>
        <Cmd>Put</Cmd>
        <SourceRef>./devinf11</SourceRef>
        <Data>200</Data>
</Status>

<Status>
        <CmdID>6</CmdID>
        <MsgRef>1</MsgRef>
        <CmdRef>4</CmdRef>
        <Cmd>Sync</Cmd>
        <TargetRef>test</TargetRef>
        <SourceRef>test</SourceRef>
        <Data>200</Data>
</Status>

<Status>
        <CmdID>4</CmdID>
        <MsgRef>1</MsgRef>
        <CmdRef>5</CmdRef>
        <Cmd>Replace</Cmd>
        <TargetRef>test1.txt</TargetRef>
        <SourceRef>test1.txt</SourceRef>
        <Data>200</Data>
</Status>
```

```
<Status>
       <CmdID>5</CmdID>
       <MsgRef>1</MsgRef>
       <CmdRef>6</CmdRef>
       <Cmd>Delete</Cmd>
       <SourceRef>test2.txt</SourceRef>
       <Data>200</Data>
</Status>

<Results>
       <CmdID>7</CmdID>
       <MsgRef>1</MsgRef>
       <CmdRef>3</CmdRef>
       <Meta>
       <Type xmlns='syncml:metinf'>application/vnd.syncml-devinf+xml</Type>
       </Meta>
       <Item>
              <Source><LocURI>./devinf11</LocURI>
                     <LocName>./devinf11</LocName>
              </Source>
              <Data><DevInf xmlns='syncml:devinf'>
                     <VerDTD>1.0</VerDTD>
                     <Man>sync4j</Man>
                     <OEM>-</OEM>
                     <FwV>-</FwV>
                     <SwV>-</SwV>
                     <HwV>-</HwV>
                     <DevID>-</DevID>
                     <DevTyp>-</DevTyp>
                     <DataStore>
                            <SourceRef>$(IGNORE)</SourceRef>
                            <DisplayName>test</DisplayName>
                            <Rx-Pref><CTType>unknown</CTType>
                            <VerCT>-</VerCT>
                            </Rx-Pref><Rx><CTType>unknown</CTType>
                            <VerCT>-</VerCT>
                            </Rx><Tx-Pref><CTType>unknown</CTType>
                            <VerCT>-</VerCT>
                            </Tx-Pref><Tx><CTType>unknown</CTType>
                            <VerCT>-</VerCT>
                            </Tx><DSMem>
                            </DSMem>
                            <SyncCap>
                                   <SyncType>2</SyncType>
                            </SyncCap>
                     </DataStore>
              </DevInf>
              </Data>
       </Item>
</Results>
<Alert>
       <CmdID>8</CmdID>
       <Data>202</Data>
       <Item>
              <Target><LocURI>test</LocURI></Target>
              <Source><LocURI>test</LocURI></Source>
              <Meta><Anchor xmlns='syncml:metinf'>
                     <Last>234</Last>
                     <Next>276</Next>
                     </Anchor>
              </Meta>
       </Item>
</Alert>
<Final/>
</SyncBody>
</SyncML>
```

The above expected response is compared with received response as explained within working of test cases. One problem within One-Way sync from server is identification of Sync Alert. Client sends an alert command (204) indicating One-Way Sync from Server. Server accepts requested synchronization mode if it follows synchronization criteria(s) like previous synchronization was successful. An acknowledgement with the Alert command is sent from server. Client replies with the expected status commands and sends a Sync command for synchronization request. Client sends following Sync Command.

```
<Sync>
       <CmdID>3</CmdID>
       <Target><LocURI>test</LocURI></Target>
       <Source><LocURI>test</LocURI></Source>
</Sync>
```

There has been confusion over how a sync alert should be sent in the second message of synchronization from client to server. Questions were asked to Sync4j community (e-mail: sync4j@yahoogroups.com) about the same, and the only feedback I got was to have empty <Sync> block only with <CmdID> and then server should respond with modification and various status commands. As <CmdID> was not permitted as an empty element without its child like <Target> or <Source>, it has been added them within <Sync> block as stated above. Server identifies this block of sync command, but replies with Null Pointer Exception while preparing memory for synchronizing the client sources with NULL items. But, if the Sync command is appended with modification commands like <Alert>, <Delete> or <Replace> no exception is thrown. Server do ignores all the client modifications & none of them are updated onto sever. This creates a problem in identifying the main reason of server behaviour for interpreting the sync alert from client for this particular type of synchronization.

The server is tested with various parameters as explained above using the test matrix, providing us with numerous results involving few bugs within the current code. We exploit more about all this in the following section.

5.5. Result of test scenario

Deviation of a message in comparison between the received response and expected response (reference) can be considered as an error or a bug within the system. To run the test

scenario, set of request and reference messages (expected responses) are built up based upon the test case matrix (Section 5.3). This section looks into various bugs explored based on definite test carried out using the test case matrix. Following bugs are exploited.

Bug # 1:	**<PUT> Command Representation Exception**

Description	Client sends its capabilities through <put> command. Server stops processing and sends response status as 400. Following root of error is traced on the server.
	`Sync4jHttpServlet: Protocol error`
	`Sync4j.framework.core.RepresentationException: parent`
	`<Data>`
	`<DevInf….`
	`----`
	`/DevInf>`
	`</Data>`
	`is missing child: <VerDTD>`

Interpretation	<DevInf> is a sub-element (child) of <Data> element. Server is unable to process <DevInf> as child of <Data>, hence giving above error message.
Possible Solution	Current code seems to support DevInf, but still error exists. Recommended to have a look at `Sync4j.framework.core.Data.java` I tested as stated by Kyungnyong, but was still received errors.

3rd Party Feedback	From User (kyungnyong) (See Appendix)
	I think there is a problem to parse <devinf> within <data>. So, I have modified the last line of Data (final Element element) method of Data class (sync4j/src/java/sync4j/framework/core/Data.java).
	the original line is:
	`data = new DeviceInfo(element).toXML();`
	and I changed like this.
	`data = new DeviceInfo(Util.extractElement(element,"DevInf", true)).toXML();`
	I think it works.

Bug # 2:	**Incorrect Sync Anchor in Alert for each database**
Description	Server sends an Alert command in response to the initialization request with the type of synchronization along with the Server anchors with the principal client. As per SyncML specifications, server must send own anchors, rather than the anchor received by client.

Interpretation	Sever anchors must be reflected instead of client anchors (currently send) in Alert command.
Possible Solution	Update the anchors sends in the initialization response.

Bug # 3:	**2 Status commands for <Delete> command**

Description	Client requests for delete of data on the server (say test2.txt) with following command.
	`Sync4jHttpServlet: Protocol error`
	`<Delete>`
	`<CmdID>5</CmdID>`
	`<Meta>`
	`<Type xmlns='syncml:metinf'>text/plain</Type>`
	`</Meta>`
	`<Item>`
	`<Source><LocURI>test2.txt</LocURI></Source>`
	`</Item>`
	`<Delete>`
	Server replies to the delete request with two status commands having same status.
	`<Status>`
	`<CmdID>2</CmdID>`
	`<MsgRef>2</MsgRef>`
	`<CmdRef>5</CmdRef>`
	`<SourceRef>test2.txt</SourceRef>`
	`<Data>200</Data>`
	`</Status>`
	`<Status>`
	`<CmdID>3</CmdID>`
	`<MsgRef>2</MsgRef>`
	`<CmdRef>5</CmdRef>`
	`<SourceRef>test2.txt</SourceRef>`
	`<Data>200</Data>`
	`</Status>`

Interpretation	There should be only one Status for the command executed. But here server sends two status messages for the same thing.
Possible Solution	Server synchronization status must be updated in order to send only one Status message for delete operation instead of two.

Bug # 4:	**No Status message for Conflicted database elements**
Description	When client sends data for modification and server finds some conflicts between those database elements, NOP (no operation) is carried out on them & client receives no status of the command requested.
Interpretation	The current code does not send status for conflicted elements leaving client

	in confusion what happened about that command.
Possible Solution	Needs to extend the response status for the conflicted elements in the sync processing server. Temporary solution can be to send a status command 500 to say unexpected error occurred at server.

Bug # 5:	**No Status message required for <Get> command**

Description	Client requests for server capabilities through <get> command. Server sends its capabilities <Results> element. It also sends a status message for the successful execution of <get> command.
	`<Status>`
	`<CmdID>4</CmdID>`
	`<MsgRef>1</MsgRef>`
	`<CmdRef>3</CmdRef>`
	`<TargetRef>./devinf11</TargetRef>`
	`<Data>200</Data>`
	`</Status>`
	`<Results> --- </Results>`

Interpretation	Results element itself is the result of the successful completion of get command. Hence, there is no need for additional status message. Even the specs don't say to send status message for <get> command.
Possible Solution	Remove the status command for the <get> command, from where it generated.

Bug # 6:	**Redundant <Type> within <Meta> of the Server capabilities**

Description	Server sends it capabilities in <Results> element. Within its <Meta> element, it contains redundant <Item> element, as shown in transcript below.
	`<Results>`
	`------`
	`<Meta>`
	`<Type xmlns='syncml:metinf'>`
	`<Type xmlns='syncml:metinf'>application/vnd.syncml-devinf+xml`
	`</Type>`
	`</Type>`
	`</Meta>`
	`---`
	`</Results>`

Interpretation	The second <Type> element with application/vnd.syncml-devinf+xml is correct. The first <type> element is redundant, and needs to be removed.
Possible Solution	Check the creation of <Type> element in <Meta> and remove the duplicity of the same <type> element.

Bug # 7:	**Double Status for <SyncHdr> and <Sync> Command for WOSI0012.**
Description	In test case WOSI0012, it is found that server replies with double status message for <SyncHdr> and <Sync> command. <SyncML> <SyncHdr> -- </SyncHdr> <SyncBody> <Status> <CmdID>1</CmdID> <MsgRef>1</MsgRef> <CmdRef>0</CmdRef> <Cmd>SyncHdr></Cmd> <TargetRef>sync.sync4j.org</TargetRef> <SourceRef>Sync4jTest</SourceRef> <Data>212</Data> </Status> -- <Status> <CmdID>3</CmdID> <MsgRef>1</MsgRef> <CmdRef>2</CmdRef> <Cmd>Sync</Cmd> <Data>200</Data> </Status> -- <Status> <CmdID>6</CmdID> <MsgRef>1</MsgRef> <CmdRef>0</CmdRef> <Cmd>SyncHdr></Cmd> <TargetRef>sync.sync4j.org</TargetRef> <SourceRef>Sync4jTest</SourceRef> <Data>200</Data> </Status> <Status> <CmdID>5</CmdID> <MsgRef>1</MsgRef> <CmdRef>2</CmdRef> <Cmd>Sync</Cmd> <TargetRef>test</TargetRef> <SourceRef>test</SourceRef> <Data>200</Data> </Status> <Final/> </SyncBody> </SyncML>
Interpretation	Server clubs the response generated by the Initialization and synchronization phase, as it is WOSI. The first four commands (CmdID 1-4) are prepared by the Init phase and CmdID 5-6 prepared by the sync phase. There is some incorrect information within the status commands, as explained below with the possible solutions.
Possible Solution	Following two things should be taken care of, while clubbing the response of init and sync to remove above bug. 1. Remove Status of <Sync> (CmdID 3) in this example from

	Initialization phase, as it contains insufficient data like TargetRef and SourceRef. 2. Remove Status of <SyncHdr> (CmdID 6) in this example from Synchronization phase, as it contains wrong data (200) as opposed with the right one in Initialization phase reponse 212 – CmdID 1.

All this bugs would be fixed within forthcoming releases, based upon the priority within the Sync4j. We need to now concentrate within next chapter looking at evaluation of the work done to complete the whole cycle of project and see how far we have achieved the problem that was spotted.

Chapter 6

Evaluation

"Evaluate what you want—
because what gets measured,
gets produced"

- James A. Belasco
American Academic & Management Consultant

6. Evaluation

Evaluation is an act of judging values of something. Encarta dictionary defines Evaluation as *"an act of considering or examining something in order to judge its value, quality, importance, extent or condition*" (Encarta, 2003). Evaluation of work done within Sync4j is also carried out on to its importance, extent and value.

Within this whole project, evaluation is carried out at two stages: (1) Evaluation of Analysis, (2) Evaluation of System. Evaluation is carried out in two stages to achieve good level of development. Again as per quote of Philip Cadwell stated in beginning of Chapter 4, if the development is excellent, but the analysis and its design is not up to the mark, this can only lead to fair results instead of excellent quality of output. Evaluation of analysis is carried out based on the formal proposal of the analysis for the enhancement within Sync4j. Evaluation of system is carried out over the whole development and current state of Sync4j.

6.1. Evaluation of Analysis

This evaluation involves looking at the proposed enhancements to be integrated within Sync4j. Proposed enhancements are One-Way Synchronization from Server, One-way Synchronization from Client, support for Sync Anchors and Synchronization without separate Initialization. These enhancements can be achieved by updating the existing code with the proposed development based on analysis explained in the earlier chapters. Proposal must be evaluated to see the effectiveness of new features within Sync4j. Evaluation is carried out by the Sync4j project in charge (Stefano Fornari). The proposal was accepted by him acknowledging that the development based on the analysis would produce the expected result. The evaluation also took into consideration for not altering the entire structure of the current process and providing generic extension to the further modules. Feedback was taken into consideration while developing the code.

6.2. Evaluation of Development

Upon complete development of new code containing enhancement features and integrating them within the existing code, it becomes vital to evaluate the entire development. This

evaluation becomes very crucial as we need to verify how far we have reached as per our expectations during the analysis stage. Are we able to deliver all that was expected? Are we able to deliver the analysed features within Sync4j in fully working conditions that can be used? Is it worth adding these features? In short we look at the importance, extent and value of the work done.

Usage of this application would involve hand held mobile devices like mobile phones and PDA. These devices would like to synchronize with their office computer or network server to retrieve the latest information and synchronise with any modifications occurred at the either end. Generally the transport medium would be OBEX (say IrDA, Bluetooth, other local connectivity) and HTTP having constraints like bandwidth, area of distance covered. Product must be developed considering such constraints to provide an effective solution achieving better synchronization through all mediums. Unnecessary flow of additional information like message headers can be reduced to achieve better utilization of the low bandwidth using feature called Sync without separate initialization, sync Anchors and One-way synchronization.

Sync without separate initialization involves merging of two phases of synchronization (Initialization + Sync) into a single message. Final produced message has a single header and body of both the phases. Sync4j did not have this capability until this feature was added to it by this project. Value of this feature can be stated as users can use either way of sending initial packets and server being capable to interpret the received style of the message. Client can send initialization packet and sync packet separately or within same packet and server processing accordingly. No major bug was traced within testing of this capability. A minor bug was found sending double status messages (bug # 7) in test and results. There are some securities considerations within this feature like: Server receives all updates from client before synchronization deciding synchronization mode. If server decides slow sync, client may need to send all of them again, causing wastage of bandwidth. (SyncML [c], 2002)

The client sends its modifications to the server before there is any possibility for the server to send its credentials (if required) to the client. There can be some possibility of masquerading.

During a synchronization event, it is recommended to send only updates taken place on devices after last successful synchronization. SyncML feature known as Sync Anchor is implemented within Sync4j to take care of it. This feature avoids the unnecessary need of sending redundant data already residing on other side. Only the modified data since last known consistent state determined through sync anchors is send during this new synchronization event. This feature is very important within synchronization as it enables in identifying the last successful synchronization between the user (user + device) and the server and take necessary decisions to allow the mode (Fast / Slow sync) of synchronization within this event. The value of the sync anchors comply with the SyncML specifications. Sync anchors was partially supported within Sync4j and was extended with the capability of deciding the sync mode based on comparison of sync anchors provided by the client with the server copy of sync anchors for this client. No bug was traced within this extended feature, except the existing one (bug # 2) – see Sec 5.5.

Many times there are needs of sending modifications from one side to another; where in the importance of one-way synchronization comes into picture. Few examples of usage of one-way synchronization are exploited in research section (chapter 3.2). With the addition of these two features (one-way sync from client / server) within Sync4j, clients are able to send or receive only updates to and from server respectively. This kind of features is generally used in specific applications, extending usage of Sync4j within such environments. It also highlights the value of adding generic features to increase the importance of Sync4j in various domains of synchronization.

Evaluating the above enhanced features within Sync4j, it seems they have achieved the expectations of the analysed requirements. The code produced for each feature (sync without separate initialization, extend support for sync anchor, one-way sync) is working well, except the one-way synchronization from server, where server proceeds by throwing a NULL pointer exception while creating memory of database to be synchronized. If sync command is added with additional commands like sync modifications from client, server behaves absolutely fine (See chapter 5). Results from the server during the test phase for the above added features are found to be in accordance with the SyncML specifications. There was no major breakage of flow of content or in-correct data. This implies better integration of the new code within the existing code of Sync4j and behaving well without any mal functioning.

Code always seems to be perfect till something goes wrong with it and a bug is spotted within itself. This exploitation is left onto for user, who can file the bugs at the project site as they come through.

6.3. Conclusion

This M.Sc project was a research cum development project to investigate and develop the enhancements required within Sync4j – an open source project implementing SyncML. SyncML is definitely the de facto open standard for synchronization. Various development forums like The Wap forum, bluetooth forum and the 3GPP group have adopted SyncML as their synchronization protocol (Rajkiran, 2001). The most interesting part of Sync4j is its openness and extensible framework allowing data synchronization services as required by corporate services by plugging in their own sync engine, synchronization sources, and the protocol support. Jonsson of Ericsson (2001) supports the concept by stating "*Users on the move will benefit from having access to a greater number of applications that are totally integrated into their corporate applications*."

This interested in exploring the need of enhancements within Sync4j, which can be used in real time applications in the way it is required. Features like one-way synchronization, full support for sync anchors and synchronization without separate initialization were required to be integrated within Sync4j to make it more robust and attractive in low bandwidth environments. To integrate this new features, first of all its SyncML specifications are studied followed by analysing the working of Sync4j. This study resulted into a platform where both the analysis can be merged to give a blue print of enhanced Sync4j with the proposed features. Blue print explained the behaviour of server in various cases and the way of achieving new features. Code was developed based on this proposed analysis using Sync4j coding standards and following the SyncML specs. Integrating these features into Sync4j makes it more robust by saving the flow of unnecessary additional data between devices and there by saving the network bandwidth and making it more attractive for usage within more domains.

6.4. Future Work

At the end of the project few things were found to be worked on further. Sync4j would like to extend its support from file format to vCalendar and vCard. It would also like to see more incorporation of new device management specs of SyncML within itself. Currently Sync4j

is supported using HTTP protocol; it can be extended with other transmission protocol like OBEX, SMTP, POP, etc. If Sync4j is growing more and more popular as its going right now, it can be deployed onto different servers like BEA, JBoss, SunOne. Finally Sync4j would like to provide more administrative interfaces to easily manage the product.

Chapter 7

Appendix

7.
Appendix

Appendix A: Glossary

CVS Concurrent Version System (CVS) is a network transparent version control system to manage development process of developers. It provides easy to use programming centric mechanism for handling multiple versions; protect from two-people modifying the same file syndrome.

EJB Enterprise Java Beans (EJB) architecture, part of J2EE, is a technology for developing, assembling, deploying and managing distributed applications in an enterprise environment.

GUID Global User Identification (GUID) is a concept used within SyncML to identify specific element within the server database. The main reason to introduce this concept within SyncML is to overcome the limitations of small names permissible for data on the hand held devices. A mapping of LUID (of devices) with the server GUID can help in identifying the elements of database.

HTTP Hyper Text Transfer Protocol (HTTP). An application level protocol in relation to TCP/IP protocol, which has been in use by the World Wide Web since 1990. It defines the set of rules for exchanging files over World Wide Web (WWW) like text, graphic, images, sound, etc. The HTTP daemon in the destination server machine receives the request, after processing the requirements, the result is returned back from the caller.

IETF Internet Engineering Task Force (IETF) is the body that defines standard internet operating protocols like TCP/IP, SMTP, etc.

J2EE Java 2 Platform, Enterprise Edition (J2EE) is a java platform designed by Sun Microsystems designed for mainframe scale computing generally for large enterprises. It is beneficial for having thin client tiered architecture.

LUID Local User Identifier (LUID) is a concept used in SyncML to identify the database elements on the local device through its short name possible with low memory. A mapping on the server (between device LUID to server GUID) is used to trace elements.

MSC Message Sequence Chart (MSC) notation is used within the data sequence chart between client and server to explore the flow of data in subsequent sequences. It helps in identifying the message with the relevant packet or sequence number.

OBEX Object Exchange Protocol (OBEX) is a multi transport protocol designed to allow one or more application layers to provide access to various network transports. It easily allows extension of support for IrDA infrared, blue tooth radio frequency and beyond.

PDA Personal Digital Assistant (PDA) is a term used for small mobile handy device capable of providing information (storing & retrieving) along with the computation of the data.

PIM Personal Information Management (PIM) involves management of the personal information hold within hand held mobile devices like contacts, calendar entries, To-Do's, etc.

POP Post Office Protocol (POP) is a client / server protocol for e-mail, where server stores the e-mails for users. Periodically users / client applications can access this mail server to check their new e-mails and could download mails. POP can be considered as "store and forward service". IMAP is an alternative to POP, where mails are mails are viewed at server and cannot be retrieved to client.

QoS Quality of Service (QoS) is the idea used to negotiate the required level of service. Generally it is agreed based on characteristics that can be measured factors like error rates, transmission rate, etc.

SDLC Software Development Life Cycle (SDLC) defines set of process to be followed for the easy management of the project as per the requirements and make sure everything progresses smoothly within the analysis, design, development and evaluation of the project.

SMTP Simple Transfer Protocol (SMTP) is used to control the transfer of e-mail messages between systems on the internet. It is based on TCP/IP protocol for sending and receiving of emails. It is generally used along with other mail protocol like POP or IMAP for receiving mails. Port # 25 of TCP/IP protocol is used for SMTP.

vCAL vCalendar (vCAL) is an ex-change format for personal scheduling information within Calendar and scheduling products. It is useful in exchange of information across a broad range of transport methods. It can be used for exchanging information between applications like Personal Information managers (PIMs), e-mail systems, web browsers, etc.

vCARD vCARD (Virtual Card) automates the exchange of information typically found on a traditional business card. It is generally used in applications like e-mail, PDAs, web browsers, smart cards, etc.

Appendix B: Source code files

This appendix provides all the source code that has been altered or added to the Sync4j project. Here only the source code of the Sync4j server is provided. Test cases have also been produced, but as there are many files, they are not included within this appendix. They could be found in accompanying disk. Additional files can be found in the accompanying disk.

1. File: sync4j.server.syncbean.session.simplessionhandler.java

```
/*
 Copyright (c) 2002 sync4j project
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions, and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions, and the disclaimer that follows
    these conditions in the documentation and/or other materials
    provided with the distribution.

 3. The name "sync4j" must not be used to endorse or promote products
    derived from this software without prior written permission.

 4. Products derived from this software may not be called "sync4j", nor
    may "sync4j" appear in their name, without prior written permission.

 In addition, we request (but do not require) that you include in the
 end-user documentation provided with the redistribution and/or in the
 software itself an acknowledgement equivalent to the following:

     "This product includes software developed by the
      sync4j project."

 THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 DISCLAIMED.  IN NO EVENT SHALL THE SYNC4J AUTHORS OR THE PROJECT
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 */

package sync4j.server.syncbean.session;

import java.security.Principal;
```

```
import java.util.logging.Logger;
import java.util.logging.Level;

import java.util.Properties;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.HashSet;
import java.util.HashMap;
import java.util.Arrays;
import java.util.List;
import java.util.Map;

import sync4j.framework.core.*;
import sync4j.framework.protocol.*;
import sync4j.framework.database.Database;
import sync4j.server.engine.Sync4jEngine;
import sync4j.framework.engine.SyncOperation;
import sync4j.framework.engine.SyncEngineFactory;
import sync4j.framework.engine.source.MemorySyncSource;
import sync4j.framework.logging.Sync4jLogger;
import sync4j.framework.security.SecurityConstants;
import sync4j.framework.security.Sync4jPrincipal;
import sync4j.framework.server.session.SessionHandler;
import sync4j.framework.server.error.ServerException;
import sync4j.framework.server.LastTimestamp;
import sync4j.framework.server.ClientMapping;
import sync4j.framework.server.error.MappingException;
import sync4j.framework.engine.SyncEngine;
import sync4j.framework.engine.SyncItemState;
import sync4j.framework.security.JAASOfficer;

import sync4j.framework.security.jaas.CredentialHandler;

import sync4j.framework.server.store.PersistentStore;
import sync4j.framework.server.store.PersistentStoreException;

/**
 * This class represents the handler for a SyncML session. It coordinates and
 * handles the packages and messages as dictated by the protocol.
 * <p>
 * The entry point is <i>processMessage()</i>, which determines which message is
 * expected and what processing has to be done (depending on the value of
 * <i>currentState</i>). If an error accours, the session goes to the state
 * <i>STATE_ERROR</i>; in this state no other messages but initialization can be
 * performed.
 * <p>
 * In the current implementation separate initialization is required.
 * <p>
 * <i>SessionHandler</i> makes use of a <i>SyncEngine</i> for all
 * tasks not related to the handling of the protocol.
 * See <i>sync4j.framework.engine.SyncEngine</i> for more information.
 *
 * @see sync4j.framework.engine.SyncEngine
 *
 * @author Stefano Fornari @ Funambol
 * @author Richard Wafer
 * @author Chintan Shah
 *
 *  $Id: SimpleSessionHandler.java,v 1.23 2003/02/12 08:37:54 chintanshah_79 Exp $
 *
 */
public class SimpleSessionHandler implements SessionHandler, java.io.Serializable
{

    // ------------------------------------------------------------- Constants

    public static final int STATE_START = 0x0000;
```

```
    public static final int STATE_END = 0x0001;
    public static final int STATE_ERROR = 0xFFFF;
    public static final int STATE_INITIALIZATION_PROCESSING = 0x0010;
    public static final int STATE_INITIALIZATION_PROCESSED = 0x0011;
    public static final int STATE_SYNCHRONIZATION_PROCESSING = 0x0012;
    public static final int STATE_SYNCHRONIZATION_PROCESSED = 0x0013;
    public static final int STATE_SYNCHRONIZATION_COMPLETION = 0x0014;


    // ------------------------------------------------------------ Private data

    private int currentState = STATE_START;
    private int nextState = STATE_ERROR;

    /** Is package containing Sync with Initialization.
    */
    private boolean syncWithInit = false;

    /**
     * Gets the current state
     *
     * @return the current state
     */
    public int getCurrentState() {
        return currentState;
    }

    private long creationTimestamp = -1;

    /**
     * Gets the creation timestamp of the session
     *
     * @return the creation timestamp
     */
    public long getCreationTimestamp() {
        return creationTimestamp;
    }

    private transient Logger log = Sync4jLogger.getLogger();

    /**
     * Timestamp (in milliseconds) of when the synchronization started
     */
    private long startTimestamp = -1;

    /**
     * Timestamp (in milliseconds) of when the synchronization ended
     */
    private long endTimestamp = -1;

    private transient SyncInitialization syncInit = null;
    private transient ClientModifications modifications = null;

    /**
     * When action command received from client set to true
     */
    private boolean syncCommandFromClient = false;

    /**
     * This Map contains all the mapping for the databases involved in the
     * synchronization for the current logged principal.<br>
     * The database names are used as keys and the corresponding mapping
     * contains the luig-guid mapping. The map is created and initialized
     * in <i>getClientMappings()</i>
     */
    private Map clientMappings = null;

    /**
```

```
     * Contain the client device Id for the current session
     */
    private String clientDeviceId = null;

    /**
     * To retrieve SyncModes of databases based on different parameters
     */
    private GetSyncMode syncMode = new GetSyncMode();

    /**
     * The databases that have to be synchronized. It is set in the initialization
     * process.
     */
    Database[] dbs = null;

    // ------------------------------------------------------------- Properties

    /**
     * The session id - read only
     */
    private String sessionId = null;

    public String getSessionId() {
        return this.sessionId;
    }

    /**
     * The command id generator (it defaults ro a <i>CommandIdGenerator</i>
instance)
     */
    private CommandIdGenerator cmdIdGenerator = new CommandIdGenerator();

    public void setCommandIdGenerator(CommandIdGenerator cmdIdGenerator) {
        this.cmdIdGenerator = cmdIdGenerator;
    }

    public CommandIdGenerator getCommandIdGenerator() {
        return this.cmdIdGenerator;
    }

    /**
     * The cmdIdGenerator must be reset each time the process
     * of a message is starting
     */
    private void resetIdGenerator() {
        this.cmdIdGenerator.reset();
        syncEngine.setCommandIdGenerator(this.cmdIdGenerator);
    }

    /**
     * The message id generator (it defaults ro a <i>SimpleIdGenerator</i>
instance)
     */
    private SimpleIdGenerator msgIdGenerator = new SimpleIdGenerator();

    /**
     * The Last message Id from the client
     */
    private String lastMsgIdFromClient = new String();

    /**
     * The factory for the <i>SyncEngine</i> object
     */
    private SyncEngineFactory syncEngineFactory = null;

    /**
     * The <i>SyncEngine</i>
     */
```

```
    private Sync4jEngine syncEngine = null;

    /**
     * Authenticated credential. If null no credential has authenticated.
     */
    private Credential      loggedCredential = null;
    private Sync4jPrincipal loggedPrincipal  = null;

    // ------------------------------------------------------------ Public methods

    /**
     * Sets the property <i> syncEngineFactory</i> and creates the instance of
     * the <i>SyncEngine</i> objects.
     *
     * @param syncEngineFactory the factory
     */
    public void setSyncEngineFactory(SyncEngineFactory syncEngineFactory) {
        this.syncEngineFactory = syncEngineFactory;
        try {
            this.syncEngine = (Sync4jEngine) syncEngineFactory.getSyncEngine();
        } catch (Sync4jException e) {
            log.throwing(getClass().getName(), "setSyncEngineFacytory", e);
        }
        if (log.isLoggable(Level.FINEST)) {
            log.finest(syncEngineFactory + " sync engine factory set");
            log.finest(syncEngine + " sync engine created");
        }
    }

    public SyncEngineFactory getSyncEngineFactory() {
        return this.syncEngineFactory;
    }

    public SyncEngine getSyncEngine() {
        return this.syncEngine;
    }

    /**
     * Indicates if the session is a new session
     */
    private boolean newSession = true;

    public void setNew(boolean newSession) {
        this.newSession = newSession;
    }

    public boolean isNew() {
        return this.newSession;
    }

    // ------------------------------------------------------------ Constructors

    /**
     * To be deserialized.
     */
    public SimpleSessionHandler() {
    }

    /**
     * Creates a new instance of ProtocolHandler
     *
     * @param sessionId the id of the session
     */
    public SimpleSessionHandler(String sessionId) {
        this.sessionId = sessionId;
        this.creationTimestamp = System.currentTimeMillis();
    }
```

```
    /**
     * Creates a new instance of ProtocolHandler
     *
     * @param sessionId the id of the session
     */
    public SimpleSessionHandler(SyncSessionIdentifier sessionId) {
        this(sessionId.getValue());
    }

    // ------------------------------------------------------------ Public methods

    /**
     * Returns true if the sessione has been authenticated.
     *
     * @return true if the sessione has been authenticated, false otherwise
     */
    public boolean isAuthenticated() {
        return loggedCredential != null;
    }

    /**
     * Returns true if the session has not been authenticated because of an
     * expired account.
     *
     * @return true if the account is expired
     */
    public boolean isAccountExpired() {
        JAASOfficer officer = (JAASOfficer) syncEngine.getOfficer();

        return officer.isLoginExpired();
    }

    /**
     * Processes the given message. See the class description for more
     * information.
     *
     * @param message the message to be processed
     *
     * @return the response message
     *
     * @throws ProtocolException
     */
    public Message processMessage(Message message) throws ProtocolException {
        Message response = null;
        Message initResponse = null;
        Message syncResponse = null;

        if (log.isLoggable(Level.FINE)) {
            log.fine("Current state: " + currentState);
        }
        if (log.isLoggable(Level.FINEST)) {
            log.finest("About processing message: " + message.toXML());
        }

        //
        // Reset the cmdIdGenerator has specified in the spec
        //
        resetIdGenerator();

        //
        // Each time a message is received for a particular session adjust the
message ID
        //
        msgIdGenerator.next();

        //
        //  We maintain the message Id from client
        //
```

```
        lastMsgIdFromClient = message.getHeader().getMessageID();

        //
        // Initialize the device ID from the client request
        //
        clientDeviceId = message.getHeader().getSource().getURI();

        try {
            switch (currentState) {
                case STATE_ERROR: // in case of error you can start a new
initialization
                case STATE_START:
                    startTimestamp = System.currentTimeMillis();
                    moveTo(STATE_INITIALIZATION_PROCESSING);
                    login(message.getHeader().getCredential(),
                          clientDeviceId                    );

                    if (isAuthenticated()) {
                        readPrincipal(loggedPrincipal);
                    }

                    response = processInitMessage(message);

                    if (isAuthenticated()) {
                        readPrincipal(loggedPrincipal);
                        moveTo(nextState);
                    } else {
                        moveTo(STATE_END);
                    }

                             //
                             //     Checking for message with Sync with
Initialization
                             //     If yes, set CurrentSTATE to
STATE_SYNCHRONIZATION_PROCESSING
                             //     and proceed ahead for synchornization...
                    //
                             if(checkSyncInit(message)) {
                                    currentState =
STATE_SYNCHRONIZATION_PROCESSING;
                                    syncWithInit = true;
                                    initResponse = new
Message(response.getHeader(), response.getBody());

                                if (log.isLoggable(Level.FINE)) {
                                           log.fine("Sync message without separate
initialization");
                                   }
                             } else {
                                    syncWithInit = false;
                                    if (log.isLoggable(Level.FINE)) {
                                           log.fine("Sync with separate
initalization");
                                    }

                             break;
                             }

                case STATE_INITIALIZATION_PROCESSED:
                case STATE_SYNCHRONIZATION_PROCESSING:
                    moveTo(STATE_SYNCHRONIZATION_PROCESSING);
                    response = processSyncMessage(message);
                             if(syncWithInit == true){
                                    syncResponse = new
Message(response.getHeader(), response.getBody()) ;
                             }

                    if (response.getBody().isFinal()) {
```

```
                    moveTo(STATE_SYNCHRONIZATION_PROCESSED);
                    moveTo(STATE_SYNCHRONIZATION_COMPLETION);
                }
                break;
                //Maybe the message wil not be in multipart for the completion
            case STATE_SYNCHRONIZATION_COMPLETION:
                response = processCompletionMessage(message);

                if (response.getBody().isFinal()) {
                    moveTo(STATE_END);
                    endTimestamp = System.currentTimeMillis();
                }
                break;

            default:
                logout();
                throw new ProtocolException("Illegal state: " + currentState);
        }
    } catch (ProtocolException e) {
        log.throwing(getClass().getName(), "processMessage", e);
        moveTo(STATE_ERROR);
        throw e;
    } catch (PersistentStoreException e) {
        log.throwing(getClass().getName(), "processMessage", e);
        moveTo(STATE_ERROR);
        throw new ProtocolException("Peristent store error", e);
    } catch (Throwable t) {
        log.throwing(getClass().getName(), "processMessage", t);
        moveTo(STATE_ERROR);
    }

     //
     //  If Sync with Initalization merge responses of
     //     Initialization & Synchronization
    //
     Message finalResponse = null;

     if(syncWithInit == true) {
        try {
            finalResponse = mergeInitSyncResponse(initResponse, syncResponse);
        } catch (RepresentationException e) {
            throw new ProtocolException("Unexpected error", e);
        }
    } else {
            finalResponse = response;
    }

    //
    // If not more messages are expected from the client, just move to the
    // end state
    //
    if ((currentState != STATE_END) && noMoreResponse(finalResponse)) {
        moveTo(STATE_END);
    }

    if (log.isLoggable(Level.FINE)) {
        log.fine("About returning message: " + finalResponse);
    }

    return finalResponse;
}

/**
 * Processes an error condition. This method is called when the error is
 * is not fatal and is manageable at a protocol/session level. This results
 * in a well formed SyncML message with an appropriete error code.
 * <p>
 * Note that the offending message <i>msg</i> cannot be null, meaning that
```

```
     * at least the incoming message was a SyncML message. In this context,
     * <i>RepresentationException</i>s are excluded.
     *
     * @param the offending message - NOT NULL
     * @param the exception representing the error condition - NOT NULL
     *
     * @throws sync4j.framework.core.Sync4jException in case of unexpected errors
     *
     * @return the response message
     */
    public Message processError(Message msg, Throwable error)
    throws Sync4jException {
        SyncHeader msgHeader = msg.getHeader();

        Item[] items = new Item[0];
        int status = StatusCode.SERVER_FAILURE;

        if (syncEngine.isDebug()) {
            items = new Item[1];

            items[0] = new Item(
                           null, // target
                           null, // source
                           null, // meta
                           new Data(error.getMessage())
                       );
        }

        if (error instanceof ServerException) {
            status = ((ServerException)error).getStatusCode();
        }

        StatusCommand statusCommand =
            new StatusCommand(
                cmdIdGenerator.next()               ,
                msgHeader.getMessageID()           ,
                "0" /* command ref */               ,
                "SyncHdr" /* see SyncML specs */     ,
                new TargetRef(msgHeader.getTarget()),
                new SourceRef(msgHeader.getSource()),
                null /* credential */               ,
                null /* challenge */                ,
                new Data(status)        ,
                new Item[0]
            );

        String serverURI =

syncEngine.getConfiguration().getStringValue(syncEngine.CFG_SERVER_URI);
        SyncHeader syncHeader = new SyncHeader (
                                    new DTDVersion     ("1.1"       )         ,
                                    new ProtocolVersion("SyncML/1.1")         ,
                                    msgHeader.getSyncSessionIdentifier()      ,
                                    msgHeader.getMessageID()                  ,
                                    new Target(msgHeader.getSource().getURI()),
                                    new Source(serverURI)                     ,
                                    null  /* response URI */                  ,
                                    false                                     ,
                                    null /* credentials */                   ,
                                    null /* metadata */
                                );

        SyncBody syncBody = new SyncBody(
                                new AbstractCommand[] { statusCommand },
                                true /* final */
                            );

        if (currentState != STATE_ERROR) {
```

```
            moveTo(STATE_ERROR);
        }
        return new Message(syncHeader, syncBody);
    }

    /**
     * Called by the <i>SessionManager</i> when the session is expired.
     * It logs out the credential and release aquired resources.
     */

    public void expire() {
        logout();
    }

    /**
     * Called to interrupt the processing in case of errors depending on
     * extenal causes (i.e. the transport). The current implementation just move
     * the session state to the error state.
     * <p>
     * NOTE that the current implementation simply moves the state of the session
     * to <i>STATE_ERROR</i>.
     *
     * @param statusCode the error code
     *
     * @see sync4j.framework.core.StatusCode for valid status codes
     *
     */
    public void abort(int statusCode) {
        moveTo(STATE_ERROR);
    }

    /**
     * Called to permanently commit the synchronization. It does the following:
     * <ul>
     *  <li>persists the <i>last</i> timestamp to the database for the sources
     *      successfully synchronized
     * </ul>
     */
    public void commit() {
        assert (loggedPrincipal != null);

        LastTimestamp last = null;

        PersistentStore ps = syncEngine.getStore();

        for (int i = 0;  (dbs != null)
                    && (i < dbs.length)
                    && (dbs[i].getStatusCode() == StatusCode.OK); ++i) {
            last = new LastTimestamp(
                    loggedPrincipal.getId(),
                    dbs[i].getName(),
                    dbs[i].getAnchor().getNext(),
                    startTimestamp,
                    endTimestamp
            );

            if (log.isLoggable(Level.FINE)) {
                log.fine("Commiting databse "
                        + dbs[i].getName()
                        + " ( "
                        + last
                        + " )"
                );
            }

            try {
                boolean stored = false;
```

```
                setClientMappings();

                stored = ps.store(last);
                log.fine("LastTimeStamp stored: " + stored);
            } catch (Sync4jException e) {
                log.severe("Error in saving persistent data");
                log.throwing(getClass().getName(), "commit", e);
            }
        }

        //
        // And at last log out
        //
        logout();
    }

    // ------------------------------------------------------- Private methods

    /**
     * Processes the given initialization message.
     *
     * @param message the message to be processed
     *
     * @return the response message
     *
     * @throws ProtocolException
     */
    private Message processInitMessage(Message message)
    throws ProtocolException {
        log.setLevel(Level.ALL);

        syncInit = new SyncInitialization(message.getHeader(),
                                          message.getBody()  );

        syncInit.setIdGenerator(cmdIdGenerator);
        syncInit.setFlag(Flags.FLAG_FINAL_MESSAGE);

        syncInit.setServerCapabilities(syncEngine.getServerCapabilities());

        if (isAuthenticated()) {
            syncInit.setAuthorizedStatusCode(StatusCode.AUTHENTICATION_ACCEPTED);
            nextState = STATE_INITIALIZATION_PROCESSED;
        } else {
            if (isAccountExpired()) {
                syncInit.setAuthorizedStatusCode(StatusCode.PAYMENT_REQUIRED);
            } else {
                syncInit.setAuthorizedStatusCode(StatusCode.FORBIDDEN);
            }
            nextState = STATE_START;
        }
        syncInit.setClientCapabilitiesRequired(false);

        //
        // Gets the databases requested for synchronization and for each of
        // them checks if the database exists on the server and if the
        // credential is allowed to synchronize it
        //
        dbs = syncInit.getDatabasesToBeSynchronized();

        //
        // This will change the status code of the elements of clientDBs to
        // reflect the availability and accessibility of the given databases
        // on the server
        //
        syncEngine.checkDatabases(dbs);

        //
        //      Set the GetSyncMode properties
```

```
        //
        syncMode.setPersistentStore(syncEngine.getStore());
        syncMode.setDatabases(dbs);
        syncMode.setTimeStamp(startTimestamp, endTimestamp);
        syncMode.setPrincipal(loggedPrincipal.getId());

        if (log.isLoggable(Level.FINEST)) {
            log.finest("Requested databases: " + Arrays.asList(dbs));
        }

        for (int i = 0; ((dbs != null) && (i < dbs.length)); ++i) {
            syncInit.setStatusCodeForCommand(
                    dbs[i].getAlertCommand(),
                    dbs[i].getStatusCode()
            );

            if (dbs[i].getStatusCode() == StatusCode.OK) {
                //
                // If the client requested a FAST sync, set the sync type based on
                // Server & Client Anchors. Otherwise, just set a SLOW sync
                //
                if (dbs[i].getMethod() != AlertCode.SLOW) {
                    dbs[i].setMethod(syncMode.getSyncModeUsingAnchor(dbs[i]));
                }

                syncEngine.addClientSource(
                        new MemorySyncSource(dbs[i].getName(),
                                null,
                                dbs[i].getTarget().getURI())
                );
            }
        }

        //
        // Setting the synchronized databases. This will force the relative
        // <Alert>s to be inserted in the response.
        //
        syncInit.setDatabases(dbs);

        return syncInit.getResponse();
    }

    /**
     * Processes the given synchronization message.
     *
     * @param syncRequest the message to be processed
     *
     * @return the response message
     *
     * @throws ProtocolException
     */
    private Message processSyncMessage(Message syncRequest)
            throws ProtocolException {
        log.finest("client sources: " + syncEngine.getClientSources());

        try {
            modifications =
                    new ClientModifications(syncRequest.getHeader(),
                                            syncRequest.getBody()  ,
                                            dbs                    );

            List responseCommands = processModifications(modifications);

            if (log.isLoggable(Level.FINEST)) {
                log.finest("responseCommands: " + responseCommands);
            }

            modifications.setIdGenerator(cmdIdGenerator);
```

```
            modifications.setFlag(Flags.FLAG_ALL_RESPONSES_REQUIRED);

            modifications.setClientModificationsStatus(
                    (StatusCommand[]) filterCommands(
                            responseCommands,
                            new String[]{StatusCommand.COMMAND_NAME}
                    ).toArray(new StatusCommand[0])
            );

                // .setServerModifications sets all the modification commands
                // required to be send to client from Server


                // cannot block this whole code,as this code may be used
                // by other DBS within same sync but differnt alert code.

            modifications.setServerModifications(
                    (AbstractCommand[]) filterCommands(
                            responseCommands,
                            new String[]{
                                SyncCommand.COMMAND_NAME
                            }
                    ).toArray(new AbstractCommand[0])
            );

            modifications.setFlag(Flags.FLAG_FINAL_MESSAGE);

            Message response = modifications.getResponse();

            response.getBody().setFinal();

            return response;
        } catch (Sync4jException e) {
            throw new ProtocolException(e);
        }
    }

    /**
     * Executes the given modifications and for each of returns a status
     * command. It forwards the execution to the synchronization engine.
     *
     * @param modifications synchronization commands containing the modifications
     *                 that the client requires
     *
     * @return an array of command objects, each containig the result of
     *         a modification or a modification itself
     * @throws Sync4jException
     */
    private List processModifications(ClientModifications modifications)
            throws Sync4jException {
        SyncHeader header           = modifications.getSyncHeader();
        String     sourceURI        = header.getTarget().getURI()  ;
        String     msgId            = header.getMessageID()        ;
        boolean    headerNoResponse = header.getNoResponse()       ;

        ArrayList responseCommands = new ArrayList();

        syncEngine.setCommandIdGenerator(cmdIdGenerator);

        SyncCommand[] syncCommands =
                (SyncCommand[]) modifications.getClientSyncCommands();

        //
        // Retrieves existing LUID-GUID mapping
        //
        getClientMappings();

        //
```

```
        // First of all prepare the memory sources with the modification commands
        // receveid by the client
        //
        prepareMemorySources(syncCommands);

        syncEngine.setDbs(dbs);

        try {
                syncEngine.sync(loggedPrincipal);
        } catch (Sync4jException e) {
            log.throwing(getClass().getName(), "processModifications", e);
        }

        //
        // Status code for sync commands
        //
        String uri = null;
        Target target = null;
        int statusCode = StatusCode.OK;
        for (int i = 0; (  (headerNoResponse == false)
                        && (syncCommands     != null )
                        && (i < syncCommands.length)); ++i) {

            target = syncCommands[i].getTarget();

            //
            // A Sync command can be empty...
            //
            if (target == null) {
                continue;
            }
            uri = target.getURI();
            if (syncEngine.getClientSource(uri) != null) {
                statusCode = StatusCode.OK;
            } else {
                statusCode = StatusCode.NOT_FOUND;
            }

            TargetRef targetRef = new TargetRef(uri);
            SourceRef sourceRef = new
SourceRef(syncCommands[i].getSource().getURI());
            responseCommands.add(
                    new StatusCommand(
                            cmdIdGenerator.next(),
                            lastMsgIdFromClient,
                            syncCommands[i].getCommandIdentifier().getValue(),
                            syncCommands[i].COMMAND_NAME,
                            new TargetRef[]{targetRef},
                            new SourceRef[]{sourceRef},
                            null,
                            null,
                            new Data(statusCode),
                            new Item[0]
                    )
            );
        }

        //
        // Status for server-side executed modification commands
        //
        if (headerNoResponse == false) {
            StatusCommand[] operationStatus =
                syncEngine.getModificationsStatusCommands(msgId);

            log.finest("operationStatus.length: " + operationStatus.length);

            for (int i=0; i<operationStatus.length; ++i) {
                responseCommands.add(operationStatus[i]);
```

```
            }
        }

        //
        // SyncCommands send back to the client
        // No sync send to client if ONE_WAY_FROM_CLIENT


        SyncOperation[] operations = syncEngine.getSyncOperations();

        ItemizedCommand[] commands = null;

        for (int i = 0; ((syncCommands != null) && (i < syncCommands.length));
++i) {
            target = syncCommands[i].getTarget();

            //
            // A Sync command can be empty...
            //
            if (target == null) {
                continue;
            }
            uri = target.getURI();

            for (int j = 0; ((dbs != null) && (j < dbs.length)); ++j) {

                if (dbs[j].getName().equals(uri) &&
                       dbs[j].getMethod() != AlertCode.ONE_WAY_FROM_CLIENT ){

                    commands = syncEngine.operationsToCommands(
                                    (ClientMapping)clientMappings.get(uri),
                                    operations,
                                    uri
                                );

                    if ((commands != null) && (commands.length > 0)) {
                        responseCommands.add(
                                new SyncCommand(
                                        cmdIdGenerator.next(),
                                        false,
                                        null,

ProtocolUtil.source2Target(syncCommands[i].getSource()),

ProtocolUtil.target2Source(syncCommands[i].getTarget()),
                                        null,
                                        commands
                                )
                            );
                    }
                }
            } // next j
        } // next i

        return responseCommands;
    }

    /**
     * Process the completion message from the client check status for sync
     * command and update client mapping
     * @param message
     * @return the response message
     * @throws ProtocolException
     */
    private Message processCompletionMessage(Message message)
            throws ProtocolException {
        log.finest("processCompletionMessage");
        Message ret = null;
```

```
        try {
            ClientCompletion clientCompletion = new ClientCompletion(
                    message.getHeader(),
                    message.getBody()
            );

            clientCompletion.setIdGenerator(cmdIdGenerator);
            clientCompletion.setFlag(Flags.FLAG_FINAL_MESSAGE);

            //
            // IF the client completion request contain MapCommand
            //
            if (clientCompletion.isMapCommandFind()) {
                String uri =
clientCompletion.getMapCommand().getTarget().getURI();
                MapItem[] mapItems = clientCompletion.getMapItems();
                for (int i = 0; i < mapItems.length; i++) {
                    MapItem mapItem = mapItems[i];

                    //Adding item properties to the persistent mapping
                    String GUID = mapItem.getTarget().getURI();
                    String LUID = mapItem.getSource().getURI();
                    ((ClientMapping)clientMappings.get(uri)).addMapping(LUID,
GUID);
                }
            }
            ret = clientCompletion.getResponse();

        } catch (Sync4jException e) {
            log.severe("Error in process completion");
            throw new ProtocolException(e);
        }
        return ret;
    }

    /**
     * Makes a state transition. Very simple implementation at the moment: it
     * changes the value of <i>currentState</i> to the given value.
     *
     * @param state the new state
     */
    private void moveTo(int state) {
        log.info("moving to state " + state);
        currentState = state;
    }

    /**
     * Prepares the memory sources with the modification commands receveid
     * by the client.
     * <p>
     * Note that if the requested synchronization is a slow sync, the items are
     * inserted as "existing" items, regardless the command they belong to.
     * (maybe thuis will change as soon as the specification becomes clearer)
     *
     * @param syncCommands the commands used to prepare the source
     *
     */
    private void prepareMemorySources(SyncCommand[] syncCommands) {

        List sources = syncEngine.getClientSources();

        //
        // For efficiency: put the databases in a HashMap
        //
        HashMap dbMap = new HashMap();
        for (int i=0; ((dbs != null) && (i<dbs.length)); ++i) {
            dbMap.put(dbs[i].getName(), dbs[i]);
```

```
        }

        //
        // First of all prepare the memory sources with the modification commands
        // receveid by the client
        //
        MemorySyncSource mss = null;
        for (int i = sources.size(); i > 0; --i) {
            if (log.isLoggable(Level.FINE)) {
                log.fine("Preparing "
                        + syncEngine.getClientSources().get(i - 1)
                        + " with "
                        + Arrays.asList(syncCommands)
                );
            }

            mss = (MemorySyncSource) sources.get(i - 1);

            boolean existing = true;
            String  name     = null;
            for (int j = 0; ((syncCommands != null) && (j < syncCommands.length));
++j) {
                name = mss.getName();
                if ((name.equals(syncCommands[j].getTarget().getURI()))
                    &&
(mss.getSourceURI().equals(syncCommands[j].getSource().getURI()))) {
                    existing = (((Database)dbMap.get(name)).getStatusCode() ==
AlertCode.SLOW);
                    prepareMemorySource(mss, syncCommands[j].getCommands(),
existing);
                }
            }
        }
    }

    /**
     * Prepares a source with the items contained in the given client
     * modification commands.
     *
     * @param source the e source to prepare
     * @param commands the client modifications
     * @param addAsExisting if true, the commmands items are stored as existing
     *                      items; if false, the commmands items are stored
     *                      according to the command they belong to
     *
     */
    private void prepareMemorySource(MemorySyncSource  source       ,
                                     AbstractCommand[] commands     ,
                                     boolean           addAsExisting) {
        ArrayList existing = new ArrayList();
        ArrayList deleted  = new ArrayList();
        ArrayList created  = new ArrayList();
        ArrayList updated  = new ArrayList();

        for (int i = 0; ((commands != null) && (i < commands.length)); ++i) {
            if (addAsExisting) {
                existing.addAll(
                         Arrays.asList(
                                 syncEngine.itemsToSyncItems(

(ClientMapping)clientMappings.get(source.getSourceURI()),
                                    source,
                                    (ModificationCommand)commands[i],
                                    SyncItemState.SYNCHRONIZED
                                )
                        )
                );
                continue;
```

```
            }
            if (AddCommand.COMMAND_NAME.equals(commands[i].getName())) {
                created.addAll(
                        Arrays.asList(
                                syncEngine.itemsToSyncItems(

(ClientMapping)clientMappings.get(source.getSourceURI()),
                                    source,
                                    (ModificationCommand)commands[i],
                                    SyncItemState.NEW
                                )
                        )
                );
                continue;
            }

            if (DeleteCommand.COMMAND_NAME.equals(commands[i].getName())) {
                deleted.addAll(
                        Arrays.asList(
                                syncEngine.itemsToSyncItems(

(ClientMapping)clientMappings.get(source.getSourceURI()),
                                    source,
                                    (ModificationCommand)commands[i],
                                    SyncItemState.DELETED
                                )
                        )
                );
                continue;
            }

            if (ReplaceCommand.COMMAND_NAME.equals(commands[i].getName())) {
                updated.addAll(
                        Arrays.asList(
                                syncEngine.itemsToSyncItems(

(ClientMapping)clientMappings.get(source.getSourceURI()),
                                    source,
                                    (ModificationCommand)commands[i],
                                    SyncItemState.UPDATED
                                )
                        )
                );
                continue;
            }
        }

        source.initialize(existing, deleted, created, updated);
    }


    /**
     * Filters a list of commands extracting the ones of the given types.
     *
     * @param commands the list of command to be filtered
     * @param types the command types to extract
     *
     * @return an array of the selected commmands
     */
    private List filterCommands(List commands, String[] types) {
        StringBuffer sb = new StringBuffer(",");

        for (int i = 0; ((types != null) && (i < types.length)); ++i) {
            sb.append(types[i]).append(',');
        }

        ArrayList selectedCommands = new ArrayList();
        AbstractCommand command = null;
```

```
        Iterator i = commands.iterator();
        while (i.hasNext()) {
            command = (AbstractCommand) i.next();

            if (sb.indexOf(',' + command.getName() + ',') >= 0) {
                selectedCommands.add(command);
            }
        }

        return selectedCommands;
    }


    /**
     * Checks that the credentials of the given message are allowed to start a
     * session.
     *
     * @param credential the message
     * @param deviceId the deviceId
     */
    private boolean login(Credential credential, String deviceId) {
        //
        // May be the credential is already logged in...
        //
        logout();

        //
        // If the credential is not specified, create a new "guest" credential
        //
        if (credential == null) {
            credential = Credential.getGuestCredential();
        }

        Sync4jPrincipal p = Sync4jPrincipal.fromCredential(credential.getData(),
                                                           credential.getType(),
                                                           deviceId            );
        if (syncEngine.login(credential)
                && syncEngine.authorize(p, SecurityConstants.RESOURCE_SESSION)) {
            loggedCredential = credential;
            loggedPrincipal  = p         ;

            return true;
        }

        return false;
    }

    /**
     * Logs out the logged in credential
     */
    private void logout() {
        if (isAuthenticated()) {
            syncEngine.logout(loggedCredential);
        }
        loggedCredential = null;
        loggedPrincipal  = null;
    }

    /**
     * Called by the <i>SyncBean</i> when the container release the session.
     * It commit the change to the DB, logs out the credential and
     * release aquired resources.
     */
    public void endSession() {
        commit();
        logout();
    }
```

```
    /**
     *   Checks if the message uses a separate initialization or not.
     *  If it contains &gt;Alert&lt; and &gt;Sync&lt; tag, it implies Sync with
     *  Initialization..
     *
     *   @param message Message to be checked for Sync with Initialization.
     *   @return TRUE (if syncWithInitialization) / FALSE (if not Sync with Init)
     *   @throws ProtocolException
    */
    private boolean checkSyncInit(Message message)
    throws ProtocolException{

          // check for the message of the SyncPackage
          // if it contains <alert> & <sync> tag, it implies Sync with
Initialization..
          // As the Initialization process is completed, only <sync> package is
needed
          // to be checked, but <alert> tag is also checked for Cross Verification.

          AbstractCommand[] clientCommands = message.getBody().getCommands();

         List syncs  = ProtocolUtil.filterCommands(clientCommands    ,
                                                   SyncCommand.class);
          List alerts = ProtocolUtil.filterCommands(clientCommands    ,
                                                   AlertCommand.class);

         return ((syncs.size() != 0) && (alerts.size() != 0));
   }

    /**
     * Checks if a message require a response from the client.<p>
     * A message requires a response if its body contains no commands other than
     * status commands.
     *
     * @param msg the message to check - NOT NULL and properly constructed
     *
     * @return true if the message requires a response, false otherwise
     *
     */
    private boolean noMoreResponse(Message msg) {
         AbstractCommand[] commands = msg.getBody().getCommands();

         for(int i=0; ((commands != null) && (i<commands.length)); ++i) {
             if (!StatusCommand.COMMAND_NAME.equals(commands[i].getName())) {
          return false;
         }
   }

         return true;
     }


   /**
      * Merges Initalization response & Synchronization response messages
      * prepared by Server. Function used during Syncrhonization with
      * Initalization Scheme.
      *
    * @param init Initialization Response Message
    * @param sync Synchronization Response Message
      *
      * @throws sync4.framework.core.RepresentationException in case of a
      *         representation error
      *
    *     @return Message
    */
   private Message mergeInitSyncResponse(Message init, Message sync)
     throws RepresentationException {
```

```
         //Message mergeResponse = null;
         AbstractCommand[] mergeCommand1 = init.getBody().getCommands() ;
         AbstractCommand[] mergeCommand2 = sync.getBody().getCommands() ;
         int i = mergeCommand1.length ;
         int j = mergeCommand2.length ;

         log.finest("Init length:" + i + "Sync length" + j);

         AbstractCommand[] mergeCommand = new AbstractCommand[i+j];

         for(;i>0;i--) {
                mergeCommand[i-1] = mergeCommand1[i-1];
                log.finest("Init ["+i+"]:"+mergeCommand[i-1].toString());
        }

         i = mergeCommand1.length;

         for(;j > 0;j--){
                mergeCommand[i+j-1] = mergeCommand2[j-1];
                log.finest("Sync["+j+"]:"+mergeCommand[i+j-1].toString());
        }

         SyncBody mergedBody = new SyncBody(mergeCommand,
sync.getBody().isFinal());

        return (new Message(init.getHeader(), mergedBody));
  }

    /**
     * Retrieves the existing LUID-GUID mappings for the logged in principal.
     * The mappings for all the databases involved in the synchronization are
     * loaded.
     *
     * @return a new <i>ClientMapping</i> object containing the mapping for the
     *         given client id
     *
     * @throws Sync4jException in case of error reading the mapping from the
     *         persistent store.
     */
    private void getClientMappings()
    throws Sync4jException {

        clientMappings = new HashMap();

        ClientMapping clientMapping = null;
        String uri = null;
        try {
            PersistentStore ps = syncEngine.getStore();

            for (int i = 0; ((dbs != null) && (i<dbs.length)); ++i) {
                uri = dbs[i].getName();

                clientMapping = new ClientMapping(loggedPrincipal, uri);

                ps.read(clientMapping);

                clientMappings.put(uri, clientMapping);
            }
        } catch (PersistentStoreException e) {
            log.severe("Unable to read clientMappings from the persistent store");
            throw new Sync4jException(e);
        }
    }

    private void setClientMappings() throws Sync4jException {
        if (clientMappings == null) {
            return;
        }
```

```
        try {
            PersistentStore ps = syncEngine.getStore();

            Iterator i = clientMappings.keySet().iterator();
            while (i.hasNext()) {
                ps.store(clientMappings.get(i.next()));
            }
        } catch (PersistentStoreException e) {
            log.severe("Unable to save clientMappings to the persistent store");
            throw new Sync4jException(e);
        }
    }

    /**
     * Read the given principal from the store. The principal must be
     * already configured with username and device. The current implementation
     * reads the following additional informatioin:
     * <ul>
     *  <li>principal id
     * </ul>
     *
     * @param principal the principal to be read
     *
     * @throws sync4j.framework.server.store.PersistentStoreException
     */
    private void readPrincipal(Sync4jPrincipal principal)
    throws PersistentStoreException {
        assert(principal               != null);
        assert(principal.getUsername() != null);
        assert(principal.getDeviceId() != null);

        PersistentStore ps = syncEngine.getStore();

        ps.read(principal);
    }

    private void readObject(java.io.ObjectInputStream in)
            throws java.io.IOException, ClassNotFoundException {
        in.defaultReadObject();
        log = Sync4jLogger.getLogger();
    }
}
```

2. File: sync4j.framework.protocol.ClientModifications.java

```
/*
 Copyright (c) 2001 sync4j project
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions, and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions, and the disclaimer that follows
    these conditions in the documentation and/or other materials
    provided with the distribution.

 3. The name "sync4j" must not be used to endorse or promote products
    derived from this software without prior written permission.
```

```
 4. Products derived from this software may not be called "sync4j", nor
    may "sync4j" appear in their name, without prior written permission.

 In addition, we request (but do not require) that you include in the
 end-user documentation provided with the redistribution and/or in the
 software itself an acknowledgement equivalent to the following:

     "This product includes software developed by the
      sync4j project."

 THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 DISCLAIMED.  IN NO EVENT SHALL THE SYNC4J AUTHORS OR THE PROJECT
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 */

package sync4j.framework.protocol;

import java.util.List;
import java.util.Arrays;
import java.util.ArrayList;

import sync4j.framework.core.*;
import sync4j.framework.database.Database;

import sync4j.framework.protocol.ProtocolUtil;
import sync4j.framework.protocol.v11.ClientModificationsRequirements;

/**
 * Represents a Client Modification package of the SyncML protocol.
 *
 *  The class is designed to be used in two times. First a
<i>ClientModification</i>
 *  is created and checked for validity and compliancy with the protocol. Than
 *  <i>getResponse()</i> can be used to get a response message for the given
 *  request. During the request validation process some information about the
 *  request message are cached into instance variables and used in
<i>getResponse()</i>.<br>
 *  Example:
 *    TO DO
 *
 * @author  Stefano Fornari @ Funambol
 *
 * @version $Id: ClientModifications.java,v 1.11 2003/01/29 23:04:40
stefano_fornari Exp $
 */
public class ClientModifications
extends SyncPackage
implements Flags {

    // ------------------------------------------------------------ Constructors

    /**
      *
      *  @param syncHeader the header of the syncronization packet
      *  @param syncBody   the body of the syncronization packet
      *
      *  @throws Sync4jException
      */
```

```
    public ClientModifications(final SyncHeader syncHeader,
                               final SyncBody   syncBody  )
    throws Sync4jException {
        super(syncHeader, syncBody);
        checkRequirements();
    }

   /**
    *
    *     @param syncHeader the header of the synchronization packet
    *     @param syncBody   the body   of the synchronization packet
    *  @param syncdb     the array of databases to be synchronized
    *
    *  @throws Sync4jException
    *  @author Chintan
    */

    public ClientModifications(final      SyncHeader syncHeader,
                                          final      SyncBody   syncBody  ,
                                          Database[] syncDb               )
    throws Sync4jException{
         super(syncHeader, syncBody);
         databases = syncDb;
         checkRequirements();
   }

    // ------------------------------------------------------------ Properties

    /**
     * Has the server sent its capabilities and is expecting a response?
     * If yes, <i>serverCapabilitiesCmdId</i> is set to the id of the Put command
     * sent by the server. If not, <i>serverCapabilitiesCmdId</i> is empty.
     */
    private CommandIdentifier serverCapabilitiesCmdId = null;

    public CommandIdentifier getServerCapabilitiesCmdId() {
        return this.serverCapabilitiesCmdId;
    }

    public void setServerCapabilitiesCmdId(CommandIdentifier
serverCapabilitiesCmdId) {
        this.serverCapabilitiesCmdId = serverCapabilitiesCmdId;
    }

    /**
     * Has the server requested client capabilities?
     * If yes, <i>clientCapabilitiesCmdId</i> is set to the id of the Get command
     * sent by the server. If not, <i>clientCapabilitiesCmdId</i> is empty.
     */
    private CommandIdentifier clientCapabilitiesCmdId = null;

    public CommandIdentifier getClientCapabilitiesCmdId() {
        return this.clientCapabilitiesCmdId;
    }

    public void setClientCapabilitiesCmdId(CommandIdentifier
clientCapabilitiesCmdId) {
        this.clientCapabilitiesCmdId = clientCapabilitiesCmdId;
    }

    /**
     * The results command in response to the request of client capabilities
     */
    private ResultsCommand clientCapabilitiesResults = null;

    public ResultsCommand getClientCapabilitiesResults() {
        return this.clientCapabilitiesResults;
    }
```

```
    /**
     * The status command in response to the sending of server capabilities
     */
    private StatusCommand serverCapabilitiesStatus = null;

    public StatusCommand getServerCapabilitiesStatus() {
        return this.serverCapabilitiesStatus;
    }

    /**
     * The client Sync command identifier. It is used when a response is required.
     */
    private CommandIdentifier clientSyncCmdId = null;

    public CommandIdentifier getClientSyncCmdId() {
        return this.clientSyncCmdId;
    }

    public void setClientSyncCmdId(CommandIdentifier clientSyncCmdId) {
        this.clientSyncCmdId = clientSyncCmdId;
    }

    /**
     * The modification commands the server wants to sent to the client.
     */
    private AbstractCommand[] serverModifications = null;

    public AbstractCommand[] getServerModifications() {
        return this.serverModifications;
    }

    public void setServerModifications(AbstractCommand[] serverModifications) {
        this.serverModifications = serverModifications;
    }

    /**
     * The status to be returned for the client sync command.
     */
    private StatusCommand[] clientModificationsStatus = null;

    public StatusCommand[] getClientModificationsStatus() {
        return this.clientModificationsStatus;
    }

    public void setClientModificationsStatus(StatusCommand[]
clientModificationsStatus) {
        this.clientModificationsStatus = clientModificationsStatus;
    }

    /**
     * Caches the commands sent by the client. It is set during the
     * checking of the requirements.
     */
    private AbstractCommand[] clientCommands = null;

    public AbstractCommand[] getClientCommands() {
        return clientCommands;
    }

   private ArrayList clientPermittedCmds = new ArrayList();

   public SyncCommand[] getClientPermittedCmds(){

         SyncCommand[] tempCmds = new
            SyncCommand[clientPermittedCmds.size()];
        clientPermittedCmds.toArray(tempCmds);
         return(tempCmds);
```

```
   }

    /**
     * Caches the SyncCommand sent by the client. It is set during the checking
     * of requirements.
     */
    private SyncCommand[] clientSyncCommands = null;

    public SyncCommand[] getClientSyncCommands() {
        return this.clientSyncCommands;
    }

    /**
     * Databases that the server wants to synchronize.
     */
    private Database[] databases = null;

    public void setDatabases(Database[] databases) {
        this.databases = databases;
    }

    public Database[] getDatabases() {
        return this.databases;
    }

    // ---------------------------------------------------------- Public methods

    /**
     * Checks that all requirements regarding the header of the initialization
     * packet are respected.
     *
     * @throws ProtocolException
     */
    public void checkHeaderRequirements() throws ProtocolException {
        ClientModificationsRequirements.checkDTDVersion
(syncHeader.getDTDVersion()           );

ClientModificationsRequirements.checkProtocolVersion(syncHeader.getProtocolVersion
()      );
        ClientModificationsRequirements.checkSessionId
(syncHeader.getSyncSessionIdentifier());
        ClientModificationsRequirements.checkMessageId
(syncHeader.getMessageID()            );
        ClientModificationsRequirements.checkTarget
(syncHeader.getTarget()               );
        ClientModificationsRequirements.checkSource
(syncHeader.getSource()               );
    }

    /**
     * Checks that all requirements regarding the body of the initialization
     * packet are respected.
     *
     * NOTE: bullet 2 pag 34 is not clear. Ignored for now.
     *
     * @throws Sync4jException
     */
    public void checkBodyRequirements() throws ProtocolException {
        clientCommands = syncBody.getCommands(); // NOTE: initializes the
clientCommands property

        //
        // If the server sent the device information to the client and requested
        // a response, serverCapabilitiesCmdId contains the command id of the
        // request command. A Status command with the same cmd id reference
        // must exist.
        //
        checkServerCapabilitiesStatus();
```

```
        //
        // If the server requested the device capabilities of the client,
        // clientCapabilitiesCmdId contains the command id of the Get command.
        // A Results command with the same cmd id reference must exist.
        //
        checkClientCapabilitiesResult();

        //
        // The Sync command must exists
        //
        checkSyncCommand();
    }

    // ---------------------------------------------------------- getResponse()

    /**
     * Constructs a proper response message.<p>
     * The sync package to the client has the following purposes:
     * <ul>
     *  <li>To inform the client about the results of sync analysis.
     *  <li>To inform about all data modifications, which have happened in the
     *      server since the previous time when the server has sent the
     *      modifications to the client.
     * </ul>
     *
     * @return the response message
     *
     * @throws ProtocolException in case of error or inconsistency
     */
    public Message getResponse() throws ProtocolException {
        ArrayList commandList = new ArrayList();

        if (idGenerator == null) {
            throw new NullPointerException("The id generator is null. Please set a
value for idGenerator");
        }

        //
        // Constructs all required response commands.
        //
        // NOTE: if NoResp is specified in the header element, than no
        //       response commands must be returned regardless NoResp is
        //       specified or not in subsequent commands
        //
        if (syncHeader.getNoResponse() == false) {

            TargetRef[] targetRefs = new TargetRef[] { new
TargetRef(syncHeader.getTarget().getURI()) };
            SourceRef[] sourceRefs = new SourceRef[] { new
SourceRef(syncHeader.getSource().getURI()) };

            StatusCommand statusCommand = new StatusCommand(
                                              idGenerator.next()              ,
                                              syncHeader.getMessageID()       ,
                                              "0" /* command ref */           ,
                                              "SyncHdr" /* see SyncML specs */ ,
                                              targetRefs                      ,
                                              sourceRefs                      ,
                                              null /* credential */           ,
                                              null /* challenge */           ,
                                              new Data(StatusCode.OK)         ,
                                              new Item[0]
                                          );

            commandList.add(statusCommand);

            //
```

```
            // 2. The Status element MUST be included in SyncBody if requested by
            //    the client. It is now used to indicate the general status of
            //    the sync analysis and the status information related to data
            //    items sent by the client (e.g., a conflict has happened.).
            //
            for (int i=0; (  isFlag(FLAG_SYNC_STATUS_REQUIRED)
                          && (clientModificationsStatus != null)
                          && (i<clientModificationsStatus.length) ); ++i) {
                commandList.add(clientModificationsStatus[i]);
            }
        }

        //
        // 3. The Sync element MUST be included in SyncBody, if earlier there
        // were no occurred errors, which could prevent the server to process
        // the sync analysis and to send its modifications back to the client.
        //
        for (int i=0; ((serverModifications != null) &&
(i<serverModifications.length)); ++i) {
            commandList.add(serverModifications[i]);
        }

        //
        // Constructs return message
        //
        Target target = new Target(syncHeader.getSource().getURI(),
                                   syncHeader.getSource().getLocationName());
        Source source = new Source(syncHeader.getTarget().getURI(),
                                   syncHeader.getTarget().getLocationName());
        SyncHeader responseHeader = new SyncHeader (

ClientModificationsRequirements.SUPPORTED_DTD_VERSION     ,

ClientModificationsRequirements.SUPPORTED_PROTOCOL_VERSION,
                                       syncHeader.getSyncSessionIdentifier()
,
                                       syncHeader.getMessageID()
,
                                       target
,
                                       source
,
                                       null  /* response URI */
,
                                       false
,
                                       null /* credentials */
,
                                       null /* meta data */
                                   );


         AbstractCommand[] commands       =      null;
        int size = commandList.size();
        if (size == 0) {
                 commands = new AbstractCommand[1];
         } else {
                commands = new AbstractCommand[size];
         }

        for (int i=0; i < size; i++) {
            commands[i] = (AbstractCommand)commandList.get(i);
        }
        SyncBody responseBody = new SyncBody(
                                    commands,
                                    isFlag(FLAG_FINAL_MESSAGE) /* final */
                                );
```

```
        try {
            return new Message(responseHeader, responseBody);
        } catch (RepresentationException e) {
            //
            // It should never happen !!!!
            //
            throw new ProtocolException("Unexpected error", e);
        }
    }

    /**
     * Create a Status command for the Sync sent by the client.
     *
     * <b>NOTE</b>: the protocol does not specify any information about the format
     * and the content of this message. By now a dummy status command is created
     * and returned.
     *
     * @return a StatusCommand object
     */
    public StatusCommand createSyncStatusCommand() {
        return new StatusCommand( idGenerator.next()                    ,
                                  "0" /* message id; TO DO */            ,
                                  clientSyncCmdId.getValue()            ,
                                  SyncCommand.COMMAND_NAME              ,
                                  (TargetRef[])null /* target refs */    ,
                                  (SourceRef[])null /* source refs */    ,
                                  null /* credential */                 ,
                                  null /* chal */                       ,
                                  null /* Data */                       ,
                                  null /* items */                      );

    }

    /**
     * For the Sync element, there are the following requirements.
     * <ul>
     *   <li> CmdID is required.
     *   <li> The response can be required for the Sync command. (See the Caching
of Map Item,
     *        Chapter 2.3.1)
     *   <li> Target is used to specify the target database.
     *   <li> Source is used to specify the source database.
     * </ul>
     *
     * 5. If there is any modification in the server after the previous sync,
     * there are following requirements for the operational elements (e.g.,
     * Replace, Delete, and Add 4 ) within the Sync element.
     * <ul>
     *   <li> CmdID is required.
     *   <li> The response can be required for these operations.
     *   <li> Source MUST be used to define the temporary GUID (See Definitions)
     *        of the data item in the server if the operation is an addition.
     *        If the operation is not an addition, Source MUST NOT be included.
     *   <li> Target MUST be used to define the LUID (See Definitions) of the
     *        data item if the operation is not an addition. If the operation is
     *        an addition, Target MUST NOT be included.
     *   <li> The Data element inside Item is used to include the data itself if
     *        the operation is not a seletion.
     *   <li> The Type element of the MetaInf DTD MUST be included in the Meta
     *        element to indicate the type of the data item (E.g., MIME type).
     *        The Meta element inside an operation or inside an item can be used.
     * </ul>
     *
     * @param db the database to be synchronized
     *
     * @return a Sync command
     *
     * @throws ProtocolException
```

```
     */
    public SyncCommand createSyncCommand(Database db)
    throws ProtocolException {
        CommandIdentifier syncId = idGenerator.next();

        AbstractCommand[] commands = null;

         // if db.getMethod is One_Way_Sync_CLIENT
         // no synccommand from Server to Client

         if(db.getMethod() != AlertCode.ONE_WAY_FROM_CLIENT){
               commands = prepareCommands(db);
         }

        return new SyncCommand(
            syncId                              ,
            isFlag(FLAG_SYNC_RESPONSE_REQUIRED),
            null                                , /* credentials */
            db.getTarget()                      ,
            db.getSource()                      ,
            null                                , /* Meta        */
            commands
        );

    }

    /**
     * Returns an array of synchronization commands (Add, Copy, Delete, Exec,
     * Replace) based on the content of the given database.
     *
     * @param db the database to be synchronized
     *
     * @return an array of AbstractCommand
     */
    public AbstractCommand[] prepareCommands(Database db) {
        ArrayList commands = new ArrayList();

        Meta meta = new Meta(new StringMetaContent(db.getType(), "Type"));

        Item[] items = null;  // reused many times

        //
        // Add
        //
        items = db.getAddItems();
        if (items != null) {
            commands.add(
                new AddCommand(
                    idGenerator.next()                          ,
                    isFlag(FLAG_MODIFICATIONS_RESPONSE_REQUIRED),
                    null /* credentials */                     ,
                    meta                                        ,
                    items                                       )
            );
        }

        //
        // Copy
        //
        items = db.getCopyItems();
        if (items != null) {
            commands.add(
                new CopyCommand(
                    idGenerator.next()                          ,
                    isFlag(FLAG_MODIFICATIONS_RESPONSE_REQUIRED),
                    null /* credentials */                     ,
                    meta                                        ,
                    items                                       )
```

```
            );
        }

        //
        // Delete
        //
        items = db.getDeleteItems();
        if (items != null) {
            commands.add(
                new DeleteCommand(
                    idGenerator.next()                          ,
                    isFlag(FLAG_MODIFICATIONS_RESPONSE_REQUIRED),
                    isFlag(FLAG_ARCHIVE_DATA)                   ,
                    isFlag(FLAG_SOFT_DELETE)                    ,
                    null /* credentials */                      ,
                    meta                                        ,
                    items                                       )
            );
        }

        //
        // Exec
        //
        items = db.getExecItems();

        for (int i=0; ((items != null) && (i<items.length)); ++i) {
            commands.add(
                new ExecCommand(
                    idGenerator.next()                          ,
                    isFlag(FLAG_MODIFICATIONS_RESPONSE_REQUIRED),
                    null /* credentials */                      ,
                    items[i]                                    )
            );
        }

        //
        // Replace
        //
        items = db.getReplaceItems();
        if (items != null) {
            commands.add(
                new ReplaceCommand(
                    idGenerator.next()                          ,
                    isFlag(FLAG_MODIFICATIONS_RESPONSE_REQUIRED),
                    null /* credentials */                      ,
                    meta                                        ,
                    items                                       )
            );
        }

        int size = commands.size();
        AbstractCommand [] aCommands = new AbstractCommand[size];
        for (int i=0; i < size; i++) {
            aCommands[i] = (AbstractCommand)commands.get(i);
        }
        return aCommands;
    }

    // ---------------------------------------------------------- Private methods

    /**
     * Checks if the requested status for server capabilities has been specified.
     * <p>
     *
     * @throws ProtocolException
     */
    private void checkServerCapabilitiesStatus()
    throws ProtocolException {
```

```
        //
        // If serverCapabilitiesCmdId is null no serverCapabilities status is
required
        //
        if (serverCapabilitiesCmdId == null) return;

        List list = ProtocolUtil.filterCommands(clientCommands         ,
                                                StatusCommand.class    ,
                                                serverCapabilitiesCmdId);

        if (list.size() == 0) {
            Object[] args = new Object[] { serverCapabilitiesCmdId.getValue() };
            throw new
ProtocolException(ClientModificationsRequirements.ERRMSG_MISSING_STATUS_COMMAND,
args);
        }

         for (int i= 0 ; list.size()!= 0 ; ++i ){
                clientPermittedCmds.add((SyncCommand)list.get(i));
         }

        serverCapabilitiesStatus = (StatusCommand)list.get(0);
    }

    /**
     * Checks if the result for client capabilities has been given.
     * <p>
     *
     * @throws ProtocolException
     */
    private void checkClientCapabilitiesResult()
    throws ProtocolException {
        //
        // If clientCapabilitiesCmdId is null no client capabilities were required
        //
        if (clientCapabilitiesCmdId == null) return;

        List list = ProtocolUtil.filterCommands(clientCommands         ,
                                                ResultsCommand.class   ,
                                                clientCapabilitiesCmdId);

        if (list.size() == 0) {
            Object[] args = new Object[] { clientCapabilitiesCmdId.getValue() };
            throw new
ProtocolException(ClientModificationsRequirements.ERRMSG_MISSING_RESULTS_COMMAND,
args);
        }

         for (int i= 0 ; list.size()!= 0 ; ++i ){
                clientPermittedCmds.add((SyncCommand)list.get(i));
         }

        ResultsCommand results = (ResultsCommand)list.get(0);

        ClientModificationsRequirements.checkCapabilities(
            results,
            ClientModificationsRequirements.CLIENT_CAPABILITIES
        );

        clientCapabilitiesResults = results;
    }

    /**
     * Checks the Sync command.
     * <p>Filters out the Sync Messages from client to Server with
     * ONE_WAY_SYNC_SERVER
     *
     * @throws ProtocolException
```

```
     */
    private void checkSyncCommand()
    throws ProtocolException {
        List list = ProtocolUtil.filterCommands(clientCommands   ,
                                                SyncCommand.class);

        if (list.size() == 0) {
            throw new
ProtocolException(ClientModificationsRequirements.ERRMSG_MISSING_SYNC_COMMAND);
        }

         // build up a list of commands to be executed on Server
         // Exclude the ONE_WAY_SYNC_SERVER (sync cmds from clnt to Server)

         GetSyncMode dbSyncMode = new GetSyncMode();
         dbSyncMode.setDatabases(databases);

         SyncCommand[] tempCommands = new SyncCommand[list.size()];

        //clientSyncCommands = new SyncCommand[list.size()];
        for (int i=0, j=0; i<list.size(); i++) {
                if(dbSyncMode.getSyncModeUsingDb(
                             ((SyncCommand)list.get(i)).getTarget(),
                             ((SyncCommand)list.get(i)).getSource()
                     ) != AlertCode.ONE_WAY_FROM_SERVER){

                     clientPermittedCmds.add((SyncCommand)(list.get(i)));
                     tempCommands[j++] = ((SyncCommand)(list.get(i)));
         }
        }


         clientSyncCommands = new SyncCommand[tempCommands.length];
         clientSyncCommands = tempCommands;

    }
}
```

3. File: sync4j.framework.protocol.SyncInitialization.java

```
/*
 Copyright (c) 2001 sync4j project
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions, and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions, and the disclaimer that follows
    these conditions in the documentation and/or other materials
    provided with the distribution.

 3. The name "sync4j" must not be used to endorse or promote products
    derived from this software without prior written permission.

 4. Products derived from this software may not be called "sync4j", nor
    may "sync4j" appear in their name, without prior written permission.

 In addition, we request (but do not require) that you include in the
 end-user documentation provided with the redistribution and/or in the
 software itself an acknowledgement equivalent to the following:
```

```
     "This product includes software developed by the
      sync4j project."

 THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 DISCLAIMED.  IN NO EVENT SHALL THE SYNC4J AUTHORS OR THE PROJECT
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 */


package sync4j.framework.protocol;

import java.util.ArrayList;
import java.util.Hashtable;
import java.security.Principal;

import sync4j.framework.core.xml.Element;

import sync4j.framework.core.*;
import sync4j.framework.database.Database;

import sync4j.framework.protocol.SyncPackage;
import sync4j.framework.protocol.ProtocolUtil;
import sync4j.framework.protocol.v11.InitializationRequirements;

import sync4j.framework.security.Sync4jPrincipal;

/**
 * Represents the Initialization package of the SyncML protocol.
 * @see SyncPackage
 *
 * Example:
 * <pre>
 *   SyncInitialization syncInit = new SyncInitialization(header, body);
 *   ... do something ...
 *   syncInit.setServerCapabilities(serverCapabilities);
 *   syncInit.setAuthorizedStatusCode(StatusCode.AUTHENTICATION_ACCEPTED);
 *   syncInit.setClientCapabilitiesStatusCode(StatusCode.OK);
 *   ... other initializations ...
 *   Message response = syncInit.getResponse();
 * </pre>
 *
 * @author Stefano Fornari @ Funambol
 *
 * @version $Id: SyncInitialization.java,v 1.16 2003/01/29 23:04:41
stefano_fornari Exp $
 *
 */
public class SyncInitialization
extends SyncPackage
{
    /**
     * Contains the request for server capabilities sent by the client
     * (null means capabilities not required)
     */
    private GetCommand serverCapabilitiesRequest = null;

    /**
```

```
     * The device capabilities sent by the client.
     */
    private PutCommand clientCapabilities = null;

    public DeviceInfo getClientDeviceInfo() throws ProtocolException {
        return ProtocolUtil.getDeviceInfo(this.clientCapabilities);
    }

    /**
     * Caches the commands sent by the client.It is set during the
     * checking of the requirements.
     */
    private AbstractCommand[] clientCommands = null;

    public AbstractCommand[] getClientCommands() {
        return clientCommands;
    }

    /**
     * Caches the alert command sent by the client. It is set during the
     * checking of the requirements.
     */
    private AlertCommand[] clientAlerts = null;

    public AlertCommand[] getClientAlerts() {
        return clientAlerts;
    }

    /**
     * Client Capabilities status code
     */
    private int clientCapabilitiesStatusCode = -1;

    public int getClientCapabilitiesStatusCode() {
        return this.clientCapabilitiesStatusCode;
    }

    public void setClientCapabilitiesStatusCode(int clientCapabilitiesStatusCode)
{
        this.clientCapabilitiesStatusCode = clientCapabilitiesStatusCode;
    }

    /**
     * Server capabilities
     */
    private DeviceInfo serverCapabilities = null;

    public void setServerCapabilities(DeviceInfo capabilities) {
        this.serverCapabilities = capabilities;
    }

    public DeviceInfo getServerCapabilities() {
        return this.serverCapabilities;
    }

    /**
     * Databases that the server wants to synchronized. They can be differnt
     * from the databases the client has requested to be synchronized.
     */
    private Database[] databases = null;

    public void setDatabases(Database[] databases) {
        this.databases = databases;
    }

    public Database[] getDatabases() {
        return this.databases;
    }
```

```
    /**
     * Does the server require client capabilities?
     */
    private boolean clientCapabilitiesRequired = false;

    public void setClientCapabilitiesRequired(boolean clientCapabilitiesRequired)
{
        this.clientCapabilitiesRequired = clientCapabilitiesRequired;
    }

    public boolean isClientCapabilitiesRequired() {
        return this.clientCapabilitiesRequired;
    }

    /**
     * Authorized status code - used in building response message
     */
    private int authorizedStatusCode = -1;

    public void setAuthorizedStatusCode(int authorizedStatusCode) {
        this.authorizedStatusCode = authorizedStatusCode;
    }

    // ------------------------------------------------------------ Command status

    /**
     * The map containing the status of the commands
     */
    private Hashtable commandStatus = new Hashtable();

    /**
     * Sets the status code for the given command.
     *
     * @param cmd the command
     * @param statusCode the status code
     */
    public void setStatusCodeForCommand(AbstractCommand cmd, int statusCode) {
        setStatusCodeForCommand(cmd.getCommandIdentifier().getValue(),
statusCode);
    }

    /**
     * Sets the status code for the command identified by the given id.
     *
     * @param cmdId the command id
     * @param statusCode the status code
     */
    public void setStatusCodeForCommand(String cmdId, int statusCode) {
        commandStatus.put(cmdId, new Integer(statusCode));
    }

    /**
     * Returns the status code for the given command. The status code must be
     * previously set with <i>setStatusCodeForCommand()</i>. If no status code
     * is associated to the given command, the default status code is returned.
     *
     * @param cmd the command
     * @param defualtStatus the default status code
     *
     * @return the status code for the command if previously set, the default
     *         status code otherwise
     */
    public int getStatusCodeForCommand(AbstractCommand cmd, int defaultCode) {
        String cmdId = cmd.getCommandIdentifier().getValue();

        return getStatusCodeForCommand(cmdId, defaultCode);
```

```
    }

    /**
     * The same as <i>getStatusCodeForCommand(AbstractCommand, int) but passing
     * in the command id instead of the command.
     *
     * @param cmdId the command id
     * @param defaultStatus
     *
     * @return the status code for the command if previously set, the default
     *         status code otherwise
     */
    public int getStatusCodeForCommand(String cmdId, int defaultCode) {
        Integer statusCode = (Integer)commandStatus.get(cmdId);

        return (statusCode == null) ? defaultCode : statusCode.intValue();
    }


    // ------------------------------------------------------------ Constructors

    /**
      *
      *  @param syncHeader the header of the syncronization packet
      *  @param syncBody   the body of the syncronization packet
      *
      *  @throws Sync4jException
      */
    public SyncInitialization(final SyncHeader syncHeader,
                              final SyncBody   syncBody  )
    throws ProtocolException
    {
        super(syncHeader, syncBody);
        checkRequirements();
    }

    // ---------------------------------------------------------- Public methods

    /**
     * Alerts specifying the database to be synchronized could be more
     * then one. Each can contains more than one item, which specifies
     * a single database. This method selects the items containing the
     * databases regardless in what alert command they where included.
     *
     * @return an array of Database objects
     */
    public Database[] getDatabasesToBeSynchronized() {
        ArrayList dbList = new ArrayList();

        Credential c = null;

        Database    db          = null;
        Item[]      items       = null;
        MetaContent metaContent  = null;
        for (int i=0; ((clientAlerts != null) && (i < clientAlerts.length)); ++i)
{
            //
            // Only database synchronization alerts are selected
            //
            if (!AlertCode.isInitializationCode(clientAlerts[i].getAlertCode())) {
                continue;
            }

            items = clientAlerts[i].getItems();
            for (int j=0; ((items != null) && (j<items.length)); ++j) {
                metaContent = items[j].getMeta().getValue();

                //
```

```
                // If the anchor does not exists, the alert does not represent
                // a database to be synchronized.
                //
                if (!(metaContent instanceof AnchorMetaContent)) {
                    continue;
                }

                c = syncHeader.getCredential();
                if (c == null) {
                    c = Credential.getGuestCredential();
                }
                Principal p = Sync4jPrincipal.fromCredential (
                                c.getData(),
                                c.getType(),
                                syncHeader.getSource().getURI());

                db = new Database(
                          items[j].getTarget().getURI()                    ,
                          null /* type */                                  ,
                          ProtocolUtil.source2Target(items[j].getSource()),
                          ProtocolUtil.target2Source(items[j].getTarget()),
                          (SyncAnchor)metaContent.getContentObject()       ,
                          p
                      );
                db.setMethod(clientAlerts[i].getAlertCode());
                db.setAlertCommand(clientAlerts[i]);

                dbList.add(db);
            }  // next j
        }  // next i

        int dbSize = dbList.size();
        Database[] dbArray = new Database[dbSize];
        for (int i=0; i<dbSize; i++) {
            dbArray[i] = (Database)dbList.get(i);
        }
        return dbArray;
    }

    // ------------------------------------------------------------------------

    /**
     * Checks that all requirements regarding the header of the initialization
     * packet are respected.
     *
     * @throws Sync4jException
     */
    public void checkHeaderRequirements()
    throws ProtocolException {
        InitializationRequirements.checkDTDVersion     (syncHeader.getDTDVersion()
);

InitializationRequirements.checkProtocolVersion(syncHeader.getProtocolVersion()
);
        InitializationRequirements.checkSessionId
(syncHeader.getSyncSessionIdentifier());
        InitializationRequirements.checkMessageId      (syncHeader.getMessageID()
);
        InitializationRequirements.checkTarget         (syncHeader.getTarget()
);
        InitializationRequirements.checkSource         (syncHeader.getSource()
);
    }

    /**
     * Checks that all requirements regarding the body of the initialization
     * packet are respected.
     *
```

```
     * @throws Sync4jException
     */
    public void checkBodyRequirements()
    throws ProtocolException {
        if (!syncBody.isFinal()) {
            throw new
ProtocolException(InitializationRequirements.ERRMSG_BODY_NOT_FINAL);
        }

        clientCommands = syncBody.getCommands();

        //
        // Extracts and checks alert commands
        //
        ArrayList alertList = ProtocolUtil.filterCommands(clientCommands    ,
                                                          AlertCommand.class);
        int size = alertList.size();
        AlertCommand[] alerts = new AlertCommand[size];
        for (int i=0; i < size; i++) {
            alerts[i] = (AlertCommand)alertList.get(i);
        }

        for (int i=0; (alerts != null) && (i < alerts.length); ++i) {
             InitializationRequirements.checkAlertCommand(alerts[i]);
        }  // next i

        //
        // All alerts are OK => they can be cached
        //
        clientAlerts = alerts;

        ArrayList clientCapabilitiesList =
            ProtocolUtil.filterCommands(clientCommands, PutCommand.class);

        if ((clientCapabilities != null) && (clientCapabilitiesList.size()>0)) {

InitializationRequirements.checkCapabilities((PutCommand)clientCapabilitiesList.ge
t(0)     ,

InitializationRequirements.CLIENT_CAPABILITIES);
            clientCapabilities = (PutCommand)clientCapabilitiesList.get(0);
        }

        ArrayList capabilitiesRequest =
            ProtocolUtil.filterCommands(clientCommands, GetCommand.class);

        if ((capabilitiesRequest != null) && (capabilitiesRequest.size()>0)) {

InitializationRequirements.checkCapabilitiesRequest((GetCommand)capabilitiesReques
t.get(0));
            serverCapabilitiesRequest = (GetCommand)capabilitiesRequest.get(0);
        }
    }


    /**
     * Constructs a proper response message.<br>
     * NOTES
     * <ul>
     *  <li> If server capabilities are not required, they are not sent (in
     *       the SyncML protocol the server MAY send not required capabilities)
     * </ul>
     *
     * @return the response message
     *
     * @throws ProtocolException in case of error or inconsistency
     */
    public Message getResponse() throws ProtocolException {
```

```
        ArrayList commandList = new ArrayList();

        //
        // Constructs all required response commands.
        //
        // NOTE: if NoResp is specified in the header element, than no
        //       response commands must be returned regardless NoResp is
        //       specified or not in subsequent commands
        //
        if (syncHeader.getNoResponse() == false) {
            //
            // Session authorization
            //
            TargetRef[] targetRefs = new TargetRef[] { new
TargetRef(syncHeader.getTarget().getURI()) };
            SourceRef[] sourceRefs = new SourceRef[] { new
SourceRef(syncHeader.getSource().getURI()) };

            StatusCommand statusCommand = new StatusCommand(
                                                idGenerator.next()
,
                                                syncHeader.getMessageID()
,
                                                "0" /* command ref */
,
                                                "SyncHdr" /* see SyncML specs */
,
                                                targetRefs
,
                                                sourceRefs
,
                                                null /* credential */
,
                                                null /* challenge */
,
                                                new
Data(String.valueOf(authorizedStatusCode)),
                                                new Item[0]
                                            );

            commandList.add(statusCommand);

            //
            // Status for each command that requested it (it is supposed each
            // command has bean already checked).
            //
            for (int i=0; ((clientCommands != null) && (i <
clientCommands.length)); ++i) {
                if (clientCommands[i].getNoResponse()) continue;

                targetRefs = new TargetRef[0];
                sourceRefs = new SourceRef[0];
                if (clientCommands[i] instanceof ItemizedCommand) {
                    Item[] items =
((ItemizedCommand)clientCommands[i]).getItems();

                    targetRefs = new TargetRef[items.length];
                    sourceRefs = new SourceRef[items.length];
                    ProtocolUtil.extractRefs(items     ,
                                             targetRefs,
                                             sourceRefs);
                }

                String commandReference =
clientCommands[i].getCommandIdentifier().getValue();
                int status = getStatusCodeForCommand(clientCommands[i],
StatusCode.OK);
```

```
                Item[] items = new Item[0];

                      ///
                      //     Within Response of Alert, Item must contain the NEXT
Anchor.
                      //
                if (clientCommands[i] instanceof AlertCommand) {
                    for(int j=0; (databases != null) && (j<databases.length) ;
++j)
                    {

if((databases[j].getTarget().getURI()).equals(targetRefs[0].getValue())){
                            items = new Item[1];

                            AnchorMetaContent alertAnchor = new AnchorMetaContent(
                                null,                   // last
                                databases[j].getNext() // next
                            );
                            items[0] = new Item(
                               null,  // target
                               null,  // source
                               null , // meta
                               new Data(alertAnchor.toXML())
                            );

                            break;
                        }
                    }
                }

                statusCommand = new StatusCommand(
                                    idGenerator.next()          ,
                                    syncHeader.getMessageID()   ,
                                    commandReference            ,
                                    clientCommands[i].getName() ,
                                    targetRefs                  ,
                                    sourceRefs                  ,
                                    null /* credential */       ,
                                    null /* challenge */        ,
                                    new Data(status)            ,
                                    items
                                );

                commandList.add(statusCommand);
            }  // next i

            //
            // Status for client capabilities
            //
            if (clientCapabilities != null) {
                String commandReference =
clientCapabilities.getCommandIdentifier().getValue();
                targetRefs = new TargetRef[] {};
                sourceRefs = new SourceRef[] {
                    new SourceRef(InitializationRequirements.CAPABILITIES_SOURCE)
                };

                Data data = new
Data(String.valueOf(clientCapabilitiesStatusCode));
                statusCommand = new StatusCommand(
                                    idGenerator.next()          ,
                                    syncHeader.getMessageID()   ,
                                    commandReference            ,
                                    clientCapabilities.getName(),
                                    targetRefs                  ,
                                    sourceRefs                  ,
                                    null /* credential */       ,
                                    null /* challenge */        ,
```

```
                                    data                            ,
                                    null /* items */
                                );

                commandList.add(statusCommand);
            }
        }  // end if syncHeader.getNoResponse() == false

        //
        // Server capabilities
        //
        if (serverCapabilitiesRequest != null) {
            if (serverCapabilities == null) {
                throw new ProtocolException("Error in creating a response: server
capabilities not set (use setServerCapabilities())");
            }

            String commandReference =
                serverCapabilitiesRequest.getCommandIdentifier().getValue();
            Meta meta = new Meta(
                            new StringMetaContent(
                                ProtocolUtil.getTypeElement(
                                    Constants.NAMESPACE_METINF              ,
                                    Constants.MIMETYPE_SYNCML_DEVICEINFO_XML
                                ),
                                "Type"
                            )
                        );
            Data   data     = new Data(serverCapabilities.toXML());
            Source source   = new Source(
InitializationRequirements.CAPABILITIES_SOURCE,

InitializationRequirements.CAPABILITIES_SOURCE);
            Item[] capabilities = new Item[] { new Item(null, source, null, data)
};

            ResultsCommand resultsCommand = new ResultsCommand(
                                                idGenerator.next()       ,
                                                syncHeader.getMessageID(),
                                                commandReference         ,
                                                meta /* meta */          ,
                                                null /* target ref */    ,
                                                null /* source ref */    ,
                                                capabilities
                                            );
            commandList.add(resultsCommand);
        }

        //
        // Alerts for each database to be synchronized
        //
        for (int i=0; (databases != null) &&
                      (i<databases.length) &&
                      (databases[i].getStatusCode() == StatusCode.OK); ++i ) {
            AlertCommand alertCommand =
                ProtocolUtil.createAlertCommand(idGenerator.next(),
                                                false             ,
                                                null              ,
                                                databases[i]      );
            commandList.add(alertCommand);
        }

        //
        // If client capabilities are required but not provided, a get command
        // must be added.
        //
        if (clientCapabilitiesRequired && (clientCapabilities == null)) {
            Meta meta = new Meta(
```

```
                            new StringMetaContent(
                                ProtocolUtil.getTypeElement(
                                    Constants.NAMESPACE_METINF              ,
                                    Constants.MIMETYPE_SYNCML_DEVICEINFO_XML
                                )
                                ,
                                "Type"
                            )
                        );
            Target target = new Target(
InitializationRequirements.CAPABILITIES_TARGET,

InitializationRequirements.CAPABILITIES_TARGET);
            Item[] items = new Item[1];

            items[0] = new Item(
                           target,
                           null  , /* source */
                           null  , /* meta   */
                           null    /* data   */
                       );
            GetCommand getCommand = new GetCommand(
                                        idGenerator.next()      ,
                                        false /* no response */ ,
                                        null  /* language    */ ,
                                        null  /* credentials */ ,
                                        meta                    ,
                                        items
                                    );
            commandList.add(getCommand);
        }

        //
        // Constructs return message
        //
        Target target = new Target(syncHeader.getSource().getURI(),
                                   syncHeader.getSource().getLocationName());
        Source source = new Source(syncHeader.getTarget().getURI(),
                                   syncHeader.getTarget().getLocationName());
        SyncHeader responseHeader = new SyncHeader (

InitializationRequirements.SUPPORTED_DTD_VERSION     ,

InitializationRequirements.SUPPORTED_PROTOCOL_VERSION,
                                        syncHeader.getSyncSessionIdentifier()
,
                                        syncHeader.getMessageID()

,
                                        target

,
                                        source

,
                                        null  /* response URI */

,
                                        false

,
                                        null /* credentials */

,
                                        null /* meta date */
                                    );

        int size = commandList.size();
        AbstractCommand [] aCommands = new AbstractCommand[size];
        for (int i=0; i < size; i++) {
            aCommands[i] = (AbstractCommand)commandList.get(i);
        }
        SyncBody responseBody = new SyncBody(
                                    aCommands,
```

```
                                    isFlag(Flags.FLAG_FINAL_MESSAGE)
                                );

        try {
            return new Message(responseHeader, responseBody);
        } catch (RepresentationException e) {
            //
            // It should never happen !!!!
            //
            throw new ProtocolException("Unexpected error", e);
        }
    }

    // ------------------------------------------------------- Private methods


}
```

4. File: sync4j.framework.protocol.GetSyncMode.java

```
/*

 Copyright (c) 2001 sync4j project
 All rights reserved.

 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions
 are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions, and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions, and the disclaimer that follows
    these conditions in the documentation and/or other materials
    provided with the distribution.

 3. The name "sync4j" must not be used to endorse or promote products
    derived from this software without prior written permission.

 4. Products derived from this software may not be called "sync4j", nor
    may "sync4j" appear in their name, without prior written permission.

 In addition, we request (but do not require) that you include in the
 end-user documentation provided with the redistribution and/or in the
 software itself an acknowledgement equivalent to the following:

     "This product includes software developed by the
      sync4j project."

 THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 DISCLAIMED.  IN NO EVENT SHALL THE SYNC4J AUTHORS OR THE PROJECT
 CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 SUCH DAMAGE.

 */
```

```
package sync4j.framework.protocol;


import java.util.logging.Logger;
import java.util.logging.Level;

import sync4j.framework.core.*;
import sync4j.framework.database.Database;
import sync4j.server.engine.Sync4jEngine;
import sync4j.framework.logging.Sync4jLogger;
import sync4j.framework.server.LastTimestamp;

import sync4j.framework.server.store.PersistentStore;
import sync4j.framework.server.store.PersistentStoreException;

public final class GetSyncMode{

    // ------------------------------------------------------------ Private data
    private transient Logger log = Sync4jLogger.getLogger();

    /**
     * Timestamp (in milliseconds) of when the synchronization started
     */
    private long startTimestamp = -1;

    /**
     * Timestamp (in milliseconds) of when the synchronization ended
     */
    private long endTimestamp = -1;


    /**
     * The <i>Persistent Store</i>
     */
    private PersistentStore ps = null;


    // ---------------------------------------------------------- Properties

    /**
     * Databases that the server wants to synchronized. They can be differnt
     * from the databases the client has requested to be synchronized.
     */
    private Database[] dbs = null;

    public void setDatabases(Database[] syncdatabases) {
        this.dbs = syncdatabases;
    }

    private String principal = null;

    public void setPrincipal(String principal){
        this.principal = principal;
    }

    public void setTimeStamp(long start, long end){
        this.startTimestamp = start;

        // When this object is called, the endTimeStamp is not set
        // @todo check if endTimeStamp is really required in ps.read(obj)
        if(end == -1){
            this.endTimestamp = System.currentTimeMillis();
        }
        else{
            this.endTimestamp   = end;
        }
    }
```

```
public void setPersistentStore(PersistentStore pstore){
    this.ps = pstore;
}
// ------------------------------------------------------- Constructor
public GetSyncMode(){
}

// ------------------------------------------------------- Public methods

/** Check the Anchors of the Client with the Server Anchor for
 *   the database to synchronized.
 *
 *   @param Database dbSync <i>database</i> to be syncronized
 *  @return syncType Aert Code indicating type of sync to be
 *   carried out on this database.
 */
public int getSyncModeUsingAnchor(Database dbSync) {

    LastTimestamp clientLast = new LastTimestamp(
    principal,
    dbSync.getName(),
    dbSync.getAnchor().getLast(),
    startTimestamp,
    endTimestamp
    );

    LastTimestamp serverLast = new LastTimestamp(
    principal,
    dbSync.getName(),
    "", // Last anchor
    startTimestamp,
    endTimestamp
    );

    //      Retrieve the values of Last Sync from the server
    //      based on the database, username through Persistent Store

    try {
        ps.read(serverLast);
    } catch(PersistentStoreException pse) {
        log.severe("Unable to retrieve timestamp from store");
        log.throwing(getClass().getName(), "getSyncModeUsingAnchor", pse);
    }

    // Check Client & server anchors to determine the type of Sync
    int syncType = AlertCode.SLOW; // set default to SLOW

    if((serverLast.last).equals(clientLast.last)){
        // store clients original Alert Code.
        syncType = dbSync.getMethod();
        log.fine("Using Client requested Alert Code ->"+ syncType);
    }
    else{
        log.info("Server forcing for SLOW Sync");
    }

    return (syncType);
}


/**
 *   Function to get the Synchronization mode of the database
 *  @param TargetRef     Targeted database   - generally Server
 *   @param SourceRef    Source database     - genearlly Client
 *   @return AlertCode of the Sync carried out on this database pair.
 *   @author Chintan
 */
public int getSyncModeUsingDb(Target tRef, Source sRef) {
```

```
        for(int i=0; (dbs!= null) && (i < dbs.length) ; i++) {
            if ( (dbs[i].getTarget().getURI()).equals(tRef.getURI()) &&
                 (dbs[i].getSource().getURI()).equals(sRef.getURI())) {
                return(dbs[i].getMethod());
            }
        }

        //
        // Forcing Sync Mode to SLOW - 201
        //

        return(AlertCode.SLOW);
    }

    // ------------------------------------------------------- Private methods


}
```

Appendix C: Problem definition

Introduction

Growth of electronic form of data and accessing them with mobility has lead to the development of mobile device applications capable of processing this data. Many times it becomes mandatory to maintain this data within certain consistent state between devices storing them. If these devices need to stay in the same identical state, synchronization is required to be carried in between them.

Synchronization

"*Data Synchronization is the process of making two data sets identical*." (Rajkiran, 2001) Synchronizations like e-mail, calendar entries and contact list in Pocket PC takes place with that of a stand alone PC or network computer. It consists of identifying difference in data between those data sets, to resolve the possible conflict and propagate relevant updates to them in order to bring both data sets into identical states. Some of the data synchronization protocols are Palm's HotSync Protocol, Pumatech's Intellisync, Microsoft's ActiveSync and SyncML initiative. SyncML is an open industry initiative supported by hundreds of companies including Nokia, OpenWave, Starfish, Ericsson, IBM, Lotus, Panasonic and Motorola. *"SyncML is the open standard that drives data mobility by establishing a common language for communications between devices, applications and networks*" (SyncML [a], 2002). Today approximately 36 SyncML servers are available in the market. Sync4J is one of those, but it is the first server being produced as an Open Source Code.

Sync4j

Sync4J is an open source Java implementation of SyncML protocol sponsored by Funambol; targeting audience with knowledge of (a) SyncML but not Java, (b) Java but not SyncML, (c) Commercial application developers and (d) Open source application developers. The uniqueness of this project is to build an application based on SyncML to integrate itself into the user's requirement for further development or synchronization. Sync4J's design goals involve core protocol support, transport protocol libraries like HTTP, SMTP, OBEX; extensible client & server framework, client & server applications.

Problem Definition

It is important to see that within Synchronization, devices already undergoing synchronization get update only of those data's that has been altered from its previous known consistent state in order to prevent from unnecessary flow of data between two synchronizing devices. SyncML Anchors can be used to retrieve the last time stamp when successful sync occurred between those devices. Modifications occurred after this anchor is exchanged. This also leads to requirements of either sending or receiving modifications or both of them from the device i.e. One-way or Two-Way Synchronization. Users may also send this synchronization data along with its initialization data within the same package to save additional data transfer. Current state of Sync4J involves the designed and developed core modules like Sync engine, user connectivity through HTTP, authentication, implementation of various SyncML specifications. To benefit users with above helpful features, some more specifications must be added within Sync4J.

Analysis and Development

SyncML specifications are implemented within Sync4J using J2EE, XML Technology. Hence, for the above problem SyncML specifications are required to be studied along with the coding technologies like J2EE and XML. As Sync4J is being developed, its architecture must be analysed along with its current code. Analysis must be carried out for integrating the above required specifications within the existing code. Analysis must be followed by development to fulfil the stated problem. A development environment should be built or configured in order to start with the coding for the analysis carried out.

Testing and Evaluation

Testing of the developed code must be carried out in order to verify its compliance with the SyncML & Sync4J Specifications. Code must be developed by practicing the Sync4J coding standards. Few test messages (request & response) must be compiled based on the specifications. These messages are then compared with the request and responses of the application to test the developed code.

Based on the differences between the compared messages, the developed modules can be evaluated to measure its success of synchronization using Sync Anchors, one-way

synchronization from server and synchronization of data with initialization. The outcome product can also be evaluated based upon its usefulness, reliability and accuracy.

Outcome

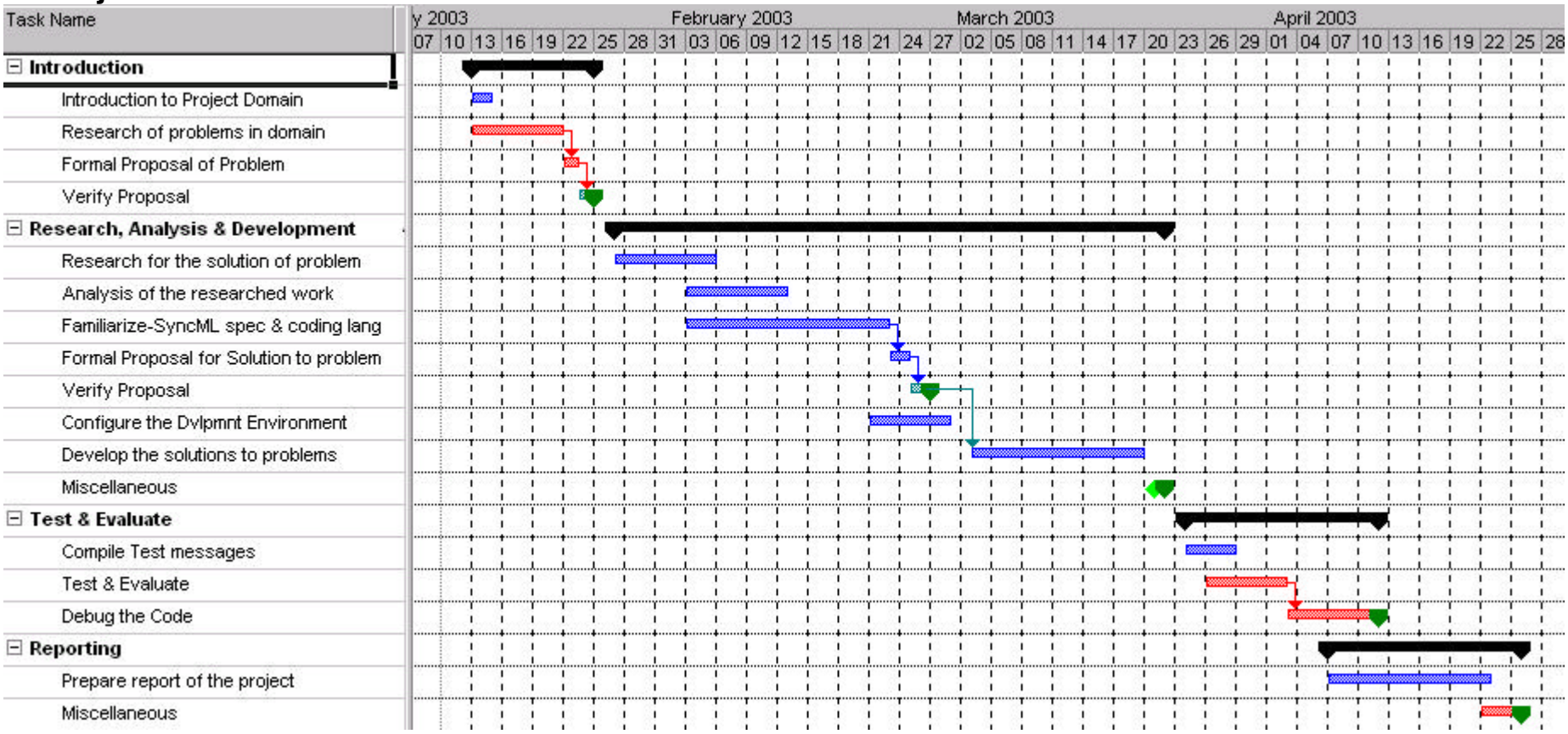
At the end of analysis & development the product would be an enhanced SyncML server with additional SyncML specifications like Sync without separate initialization, One-Way Sync and Full support for Sync Anchors.

Benefits may include

- Undergoing synchronization as required by client (user) i.e. Slow Sync, One Way sync from client / server.
- Current synchronization based upon last successful synchronization i.e. Support for SyncML Anchor

Reduce flow of data packages by clubbing Initialization with Sync data i.e. Sync without separate initialization.

Project Timeline



Legends:

	Major Task			External entity Task			Task completion Time check
	Sub Tasks			Task must be completed for its successor			Dependency between tasks

Chapter 8

Bibliography and References

8.

Bibliography and References

- **Agarwal, S. Starobinski, D. & Trachetnberg, A.** (2002) On the scalability of data synchronization protocols for PDAs and mobile devices. *IEEE Network* Volume 16 Issue 4, July-August 2002, pp. 22-8.
- **Ant** (2002) *Home page of Ant* [Internet] Apache Jakarta. Available online from <http://jakarta.apache.org/ant> [Accessed date: 08 November 2002]
- **Blaxter, L. Christina, H. & Malcolm, T.** (2001) *How to research*. 2nd Edition. Buckingham, Open Univeristy Press.
- **Blodgett, M.** (1997) *Mobile users stays up-to-the-minute.* [Internet] Computer World June 23. Available from <http://www.computerworld.com/news/1997/story/0,11280,6192,00.html> [Accessed date: 20 October 2002]
- **Buchmann, D.** (2002) *SyncML and its java implementation Sync4j* [Internet] University of Fribourg, Switzerland. Available from <http://sync4j.sourceforge.net/web/RandD/David%20Buchmann.pdf> [Accessed date: 11 November 2002]
- **Cactus** (2002) *Home page of Cactus* [Internet] Apache Jakarta. Available online form <http://jakarta.apache.org/cactus> [Accessed date: 08 November 2002]
- **Concurrent Versions System – CVS** (2002) *Introduction to CVS* [Internet] CVS Official Home page. Available from <http://www.cvshome.org> [Accessed date: 13 November 2002]
- **Coulouris, G. Dollimore, J. & Kindberg, T.** (2001) *Distributed Systems Concepts and Designs. Chapter 10 – Time and Global States*. pp. 385-416. 3rd Edition. Harlow, Addison Wesley Publishers.
- **Crouch, C**. (2001) *Device synchronization gets simpler.* [Internet] Computer World April 17. Available from <http://www.computerworld.com/mobiletopics/mobile/story/0,10801,59770,00.html> [Accessed date: 15 November 2002]
- **Cygwin** (2002) *Home page of Cygwin* [Internet] Cygwin Available from <http://www.cygwin.com> [Accessed date: 08 November 2002]

- **Encarta** (2003) *MSN learning & research dictionary* [Internet] Microsoft. Available from <http://encarta.msn.com/encnet/features/dictionary/DictionaryHome.aspx> [Accessed date: 28 January 2003]
- **Hansmann, U. Mettala, R. Purakavastha, A. & Thompson, P.** (2002) *SyncML: Synchronizing and managing your mobile data.* New Jersey, Prentice Hall.
- **IEEE** (1990) *Standard glossary of software engineering terminology, IEEE standard 710.121990*. New York, Institute of Electrical and Electronics Engineers.
- **Jdom** (2002) *Home page of Jdom* [Internet] Jdom. Available from <http://www.jdom.org> [Accessed date: 08 November 2002]
- **Jonsson, A. & Lars, N.** (2001). SyncML – Getting the mobile internet in Sync. *Ericsson Review*. Volume 78, Issue no. 3, pp. 110-115.
- **Junit** (2002) *Home page of Junit* [Internet] Junit. Available from <http://www.junit.org> [Accessed date: 08 November 2002]
- **Kenton, O'Hara. Peny, M. Sellen, A. & Brown, B**. (2001) *Wireless World – Social and inter-actional aspects of the mobile age. Chapter 12 – Exploring the relationships between mobile phone and document activity during business travel.* pp. 180-194. Eds Brown, B. Green, N & Harper, R. London, Springer
- **Keri, S.** (1998) *Short introduction to the bus priority functions used in Helsinki* [Internet] City of Helsinki, Traffic planning division. Available from <http://www.hel.fi/ksv/entire/repPriorityFunctions.htm> [Accessed date: 24 January 2003]
- **Maximilian, B.** (2002) *Integrated PIM data management with SyncML*. [Internet] Available from <http://wwwbrauer.informatik.tu-muenchen.de/~bergerm/syncml/> [Accessed date: 12 November 2002]
- **McGregor, J.** (2001) Testing a software product line [Internet] Carnegie Mellon software engineering institute, Pittsburgh, USA. Available from <http://www.sei.cmu.edu/pub/documents/01.reports/pdf/01tr022.pdf> [Accessed date: 11 January 2003]
- **Microsoft** (2003) *Active Sync* [Internet] Microsoft Available from <http://www.microsoft.com/mobile/pocketpc/downloads/activesync.asp> [Accessed date: 15 January 2003]
- **Mobile Computing and Networking.** Dellas, Texas (1998) *What is a file synchronizer? : Proceedings of the fourth Annual ACM/IEEE International conference on Mobile computing and networking*. By Balasubramaniam, S & Pierce B.C., October 1998.

- **Post Gre SQL** (2002) Home page of Post Gre Sql [Internet] PostGre Sql. Available online from <http://www.postgresql.com> [Accessed date: 08 November 2002]
- **Pots, C**. (1998) Software process models. [Internet] Georgia Tech, School of Computing, Atlanta, USA. Available from <http://www.cc.gatech.edu/computing/SW_Eng/people/Faculty/Colin.Potts/Courses/3302/1-08-mgt/index.htm> [Accessed date: 15 January 2003]
- **Pressman, R**. (2001) *Software Engineering: A practitioner's Approach – European adaptation. Chapter 2 – The process.* pp. 18-53. 5th Edition, Berkshire, McGrawHill Publishers.
- **Rajkiran** (2001) Standard sync with SyncML *Developer 2.0 December 2001*
- **Stemberger, S** (2001) *Syncing data* [Internet] IBM Available from <http://www-106.ibm.com/developerworks/wireless/library/wi-syncml> [Accessed date: 18 December 2002]
- **Sullivan, S.** (2001) *Data synchronization with SyncML and Sync4j* [Internet] Sync4j Available from <http://sync4j.sourceforge.net/web/presentations/sync4j-2001-08-28-pjug.ppt> [Accessed date: 10 November 2002]
- **Swetnam, D.** (2000) *Writing your dissertation: How to plan, prepare and present successful work.* Oxford, How to Books.
- **Sync4j** (2002) *Sync4j Architecture* [Internet] Sync4j. Available from <http://sync4j.sourceforge.net/web/project/architecture/sync4j-architecture.html> [Accessed date: 01 November 2002]
- **Synchrologic** (2002) *The hand held applications guidebook. Getting started deploying and supporting enterprise applications on hand held PDAs.* [Internet]. Synchrologic, Alpharetta, GA 30022. Available from <http://www.synchrologic.com/images/whitepapers/handheld_applications_guidebook.html> [Accessed date: 02 November 2002]
- **SyncML [a].** (2001) *Building an industry-wide mobile synchronization protocol* [Internet] SyncML white paper. Available from <http://www.syncml.org/download/whitepaper.pdf> [Accessed date: 05 October 2002]
- **SyncML [b].** (2002) SyncML representation protocol, data synchronization usage [Internet] SyncML. Available from

`<http://www.syncml.org/docs/syncml_sync_represent_v11_20020215.pdf>` [Accessed date: 05 November 2002]

- **SyncML [c].** (2002) SyncML sync protocol [Internet] SyncML. Available from `<http://www.syncml.org/docs/syncml_sync_protocol_v11_20020215.pdf>` [Accessed date: 05 November 2002]
- **SyncML [d].** (2002) SyncML representation protocol [Internet] SyncML. Available from `<http://www.syncml.org/docs/syncml_represent_v11_20020215.pdf>` [Accessed date: 05 November 2002]
- **Veitch, R**. (2003) *The truth about Mobile and Wireless Computing*. [Internet] Computer World January 3. Available from `<http://www.computerworld.com/mobiletopics/mobile/story/0,10801,77230,00.html>` [Accessed: 22 January 2003]
- **Wireless and Mobile Communications.** 4586. Beijing, China (2001). *Mobile synchronization solutions in bluetooth wireless technology: Proceedings of the SPIE – The international society for optical engineering*, by Hongbo, L. Qiang, G. pp. 236-44
- **World Wide Retail Exchange** (2002) *What is data Synchronization?* [Internet] Available from `<http://www.worldwideretailexchange.org/cs/loadmydata/lmd003.htm>` [Accessed date: 20 October 2002]