SyncML

(**Sync**hronization **M**arkup **L**anguage)

and its Java Implementation sync4j

Diploma Thesis in Informatics
University of Fribourg, Switzerland

Author:
David Buchmann
david.buchmann@gmx.net

Supervisor:
Dr. Rudolf Scheurer
Telecom Research Group
Department of Informatics

September 2002

Abstract

This paper examines the data synchronization protocol SyncML. The **Sync**hronization **M**arkup **L**anguage is an initiative of major mobile companies and other information technology firms to achieve a general standard for synchronization of any data between mobile devices and computers.

Further, the Java implementation of SyncML sync4j is described. The main focus is on the parts implemented by the author within the writing of this diploma thesis. A prototype has been built and is used for synchronizing time reporting between a Palm handheld computer and the industry standard system SAP R/3.

Acknowledgments

The author wants to thank Dr. Rudolf Scheurer from the University of Fribourg, Lars Erdmann and the ESPRiT Consulting Team, the sync4j developers and Nicola Fankhauser.

The writing of this diploma thesis was kindly supported by

ESPRiT CONSULTING

Contents

Chapter 1

Introduction

Over the last years, small, mobile devices with data processing capabilities enjoy of increasing popularity. Modern cellular phones don't have just a memory for phone numbers, but have become a personal information manager (PIM) with address book, agenda, email client, browser and much more. Small computers in the size of calculators, so called handhelds, are getting commonly used.

Applications similar to those on such devices exist on personal computers and inside enterprise databases (address book, time planner, ...). Mobile devices are not permanently connected to a network and therefore need an own copy of those databases. Modifications will be made on mobile devices as well as on the network. Periodically, informations about these modifications should be exchanged. This operation is called synchronization.

Currently, manufacturers use their proprietary synchronization protocols, the devices lack interoperability. Flexibility and usability of the synchronization tools are often unsatisfactory and there is no standard way to synchronize devices from different manufacturers.

In 1999, a group of major companies founded the SyncML Initiative, with the aim of developing and establishing a general standard for synchronization issues. They are Ericsson, IBM, Lotus, Matsushita Communications Industrial Co., Motorola, Nokia, Palm Inc., Psion and Starfish Software. One year later, the first step had been accomplished: Version 1.0 of Synchronization Markup Language (SyncML) has been published. It is formulated in the e**X**tended **M**arkup **L**anguage (XML). If the Initiative succeeds, future devices will support SyncML as standard synchronization protocol.

To achieve this, there is still a long way to go. Only if a majority of mobile device manufacturers and also some backend system developers implement SyncML into their products, SyncML will become an attractive standard. Microsoft, for example, is not member of the Initiative. Success depends not only on the quality of the protocol and its specifications, but also on economical factors and strategic decisions by the companies managers.

ESPRiT Consulting AG supported this diploma thesis to examine interaction between a Palm handheld computer and the industry standard system SAP R/3. As a concrete application, the SAP R/3 application CATS for managing working times had been chosen. During a stage in summer 2001, the author developed a prototype for time reporting on the Palm and a simple synchronization with SAP, but without using SyncML.

The telecom group of the University of Fribourg is interested in the SyncML protocol and offered a diploma thesis in this area. ESPRiT Consulting is interested in this protocol too, for improving the time reporting prototype.

1.1 Overview

Further down this chapter, the objectives of the diploma thesis are listed and the proceeding is explained.

The second chapter explains what SyncML is. An overview of the SyncML protocol concept is given, followed by a summary of the enquiry about SyncML performed by the author. The complete enquiry is available on the Internet[1] or on the accompanying CD (see Appendix B). A short overview of the few existing SyncML implementations concludes this chapter.

In the third chapter, the open source project sync4j is described. Sync4j is an effort to implement the SyncML protocol in Java. The author has developed a running prototype on base of this incomplete project.

The fourth chapter is dedicated to the practical application built with this prototype. The SAP time reporting system was extended, using a Palm handheld to enter data and SyncML to synchronize between the Palm and a SAP R/3 server.

The last chapter contains the conclusions from the project. Criticisms are made on what has been developed and remarks on what could be improved. The experiences with SyncML are discussed.

1.2 Objectives

This thesis will examine Synchronization and especially SyncML. Theoretical analysis will be accompanied by the development of a concrete application. The program “TiME” will allow to report working time on a Palm Handheld computer and SyncML will be used to synchronize the time reports. With the practical example, we will find out whether SyncML is usable. SyncML could be too simple to allow flexible synchronization - or too complex to be implemented with reasonable effort.

The SyncML Initiative wants its protocol to become *the* universal synchronization standard, but alternatives shall be searched and examined if existing.

Goals for the practical work

- A running SyncML prototype.
- Data synchronization based on the SyncML prototype, between SAP R/3 and the Palm.
- An application for Palm handhelds allowing the user to report working time.
- Conduct a user test for the Palm application.

[1]`http://http://diuf.unifr.ch/telecom/projects/syncml/SyncML_Enquiry.pdf`

1.3 Procedure

Work on the diploma thesis began in November 2001 and had been finished by September 2002. Table 1 gives an idea of what was achieved when.

In a first phase, SyncML had been studied and other synchronization protocols had been searched for. Part of the SyncML research was an online enquiry among the companies that form the SyncML initiative and the companies from the regional contact organization IT Valley Fribourg. At the same time, the program for time reporting on Palm was improved and submitted to a user test.

Depending on the results of the studies on SyncML and other protocols, SyncML was selected as being suitable for implementing synchronization between a Palm and SAP. The only existing base found was sync4j, so it was chosen although it was not yet finished. A considerable part of work was done to get sycn4j running. The Palm application was improved by implementing requests of the user feedback.

01. 11. 2001	Proposal accepted, begin of work
31. 01. 2002	First version of the time reporting tool running
01. 04. 2002	Enquiry completed
20. 04. 2002	User tests with first version finished and evaluated
15. 05. 2002	Redesign of Palm application finished
10. 07. 2002	Sync4j prototype working, including Palm conduit
10. 09. 2002	Paper finished

Table 1: Time planning for the diploma thesis

Chapter 2

SyncML

This chapter is a introduction to synchronization concepts in general and SyncML specifically. Both parts of SyncML, data synchronization and device management, are explained. A look on alternatives to SyncML is taken. Another Section gives a short overview of the enquiry on SyncML held by the author in spring 2002. Finally, an overview of existing SyncML implementations is given.

2.1 Synchronization concepts

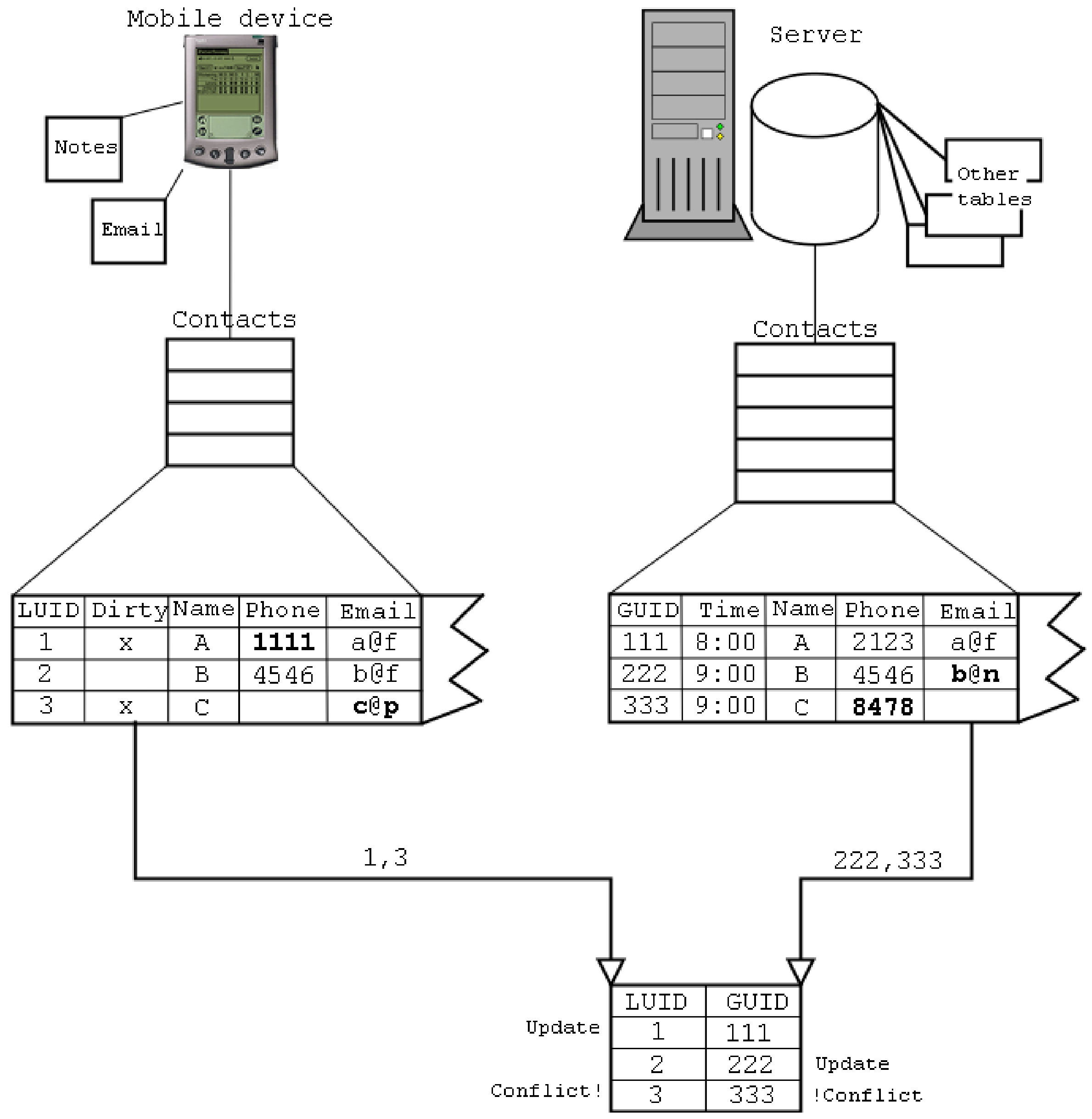
As soon as data is replicated over different locations, synchronization is needed to ensure all copies hold the same information (consistency). Different granularities exist: One can compare files in two file systems and check modification dates and sizes. CVS, the concurrent version system, synchronizes on lines of text inside the files.

What we will examine is not synchronization in a file system, but between databases. Mobile devices are used to store contact information, agenda entries, etc. Each address entry has the same structure, name, street, city, phone number etc. They can be thought of as being rows[2] in a table. After synchronization, both sides address books should have all entries with the same content. Figure 1 shows the big picture.

The simplest method for synchronization is to send all entries to the other side. There they are matched to the existing entries to avoid duplicates. This is called **full sync** or **slow sync**. This type is not very efficient, as unmodified data existing on both side is transmitted and compared. However, this long during process cannot be avoided if the devices lost track of what has already been synchronized and what not.

It is more efficient to only send modified items. For this purpose, changes must be detectable. On the server, this probably is a "last modified" timestamp. On the client, a simpler scheme often is used, the **dirty bit**. If a record is edited, it is set dirty. On synchronization, only dirty records are sent and after the termination of the communication, all dirty bits are removed. This type is called **fast sync**, as it is much more efficient. If both the server and the client record have been modified since last sync session, a **conflict** occurs. Some conflicts can be resolved automatically by merging the two items, others can't be solved and the user must be asked which version is the correct one.

[2]The words row, entry, item and record are used interchangeable in this thesis.



LUID	Dirty	Name	Phone	Email
1	x	A	**1111**	a@f
2		B	4546	b@f
3	x	C		**c@p**

GUID	Time	Name	Phone	Email
111	8:00	A	2123	a@f
222	9:00	B	4546	**b@n**
333	9:00	C	**8478**	

LUID	GUID
1	111
2	222
3	333

Figure 1: Concept of record based synchronization.

To reliably identify items, unique identifiers are used. As sometimes identifiers are generated automatically (think of `autoincrement` fields in databases) client and server may assign different ID's to items. The client ID's are called local unique identifier (**LUID**), as other clients are not known and could assign the same ID to an other item. The server has global unique identifiers (**GUID**). The server has to map between LUID and GUID for each device.

The example in Figure 1 shows the synchronization of an address book with three entries. On client side, they are numbered 1,2,3, on server side 111,222,333. The client items 1 and 3 have the dirty bit and thus are sent to the server. The server items 222 and 333 have a timestamp that dates after the last sync, so they have been modified since. The server maps 1 to 111 and updates the server item 111. 222 is mapped to 2 and sent to the client. Because 3 maps to 333 and both sides are modified, a conflict occurs. As the differing fields are empty on the other sides, they can be merged. The client gets the phone number, the server the email address.

For further reading on synchronization concepts, [4] is recommended.

2.2 Introduction to SyncML

Mobile devices like handheld computers, mobile phones and laptops are commonly used. They serve as personal information management (PIM) devices, are used for business critical data and also just for fun. One important issue related to mobile devices is synchronization. For private users, this could be just a backup, but business users need to maintain consistency with backend systems and central administration of devices.

To replace the chaos of existing proprietary protocols, major companies founded the SyncML Initiative in February 2000. The aim of this initiative is to establish SyncML as the general standard for synchronization issues.

The first specification of SyncML was released in December 2000 on the SyncML web site `http://www.syncml.org`. In February 2002, version 1.1 of the specification has been published.

The focus of this XML protocol is on mobile devices that need to synchronize with database servers, but other scenarios should be possible as well. SyncML can be used over different connections like the Internet, WAP, infrared and others. SyncML provides a framework adopted to the slow connections wireless devices offer nowadays. The hardware limits of mobile devices like maximal record size of free storage capacity are taken into account by the exchange of extra information inside `Meta` tags.

Version 1.1 added a new area to SyncML. A device management protocol allows central administration of large numbers of devices (see Section 2.2.4).

2.2.1 Alternatives to SyncML

The enquiry[3] revealed no real alternative to SyncML. There are standards for network communications, but SyncML is the only general standardized protocol with specific capabilities for synchronization of records. For special cases, proprietary protocols exist.

One example is the HotSync application[4] for record oriented communication between

[3]See Section 2.3 and Appendix B

[4]See Section 4.2 and [16] for details about HotSync.

Palm and Desktop. It is used to read records from Palm over a serial line or infrared. The protocol used by HotSync is not public and is not thought to be used over the Internet or other networks. The traffic generated would be very inconvenient, as a separate read command is issued for each record. Some concepts of HotSync can be seen in the SyncML specification.

For synchronizing iCalendar entries, the IETF[5] defines the **i**Calendar **T**ransport-Independent **I**nteroperability **P**rotocol (iTIP). iCalendar is a format for agenda entries. The iTIP protocol is specific for iCalendar and oriented towards different people using automated scheduling of meetings. Users publish their free and busy time and automated agenda planners use iTIP to find a time at which everybody can attend the meeting. It can not be used to synchronize two calendars of the same person.

2.2.2 SyncML concept

As mentioned before, SyncML was designed with mobile devices in mind. The following requirements where defined by the initiative in their whitepaper, page 5, available at [1]:

- Operate effectively over wireless and wireline 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 Internet and Web technologies
- The protocol's minimal function needs to deliver the most commonly required synchronization capability across the entire range of devices.

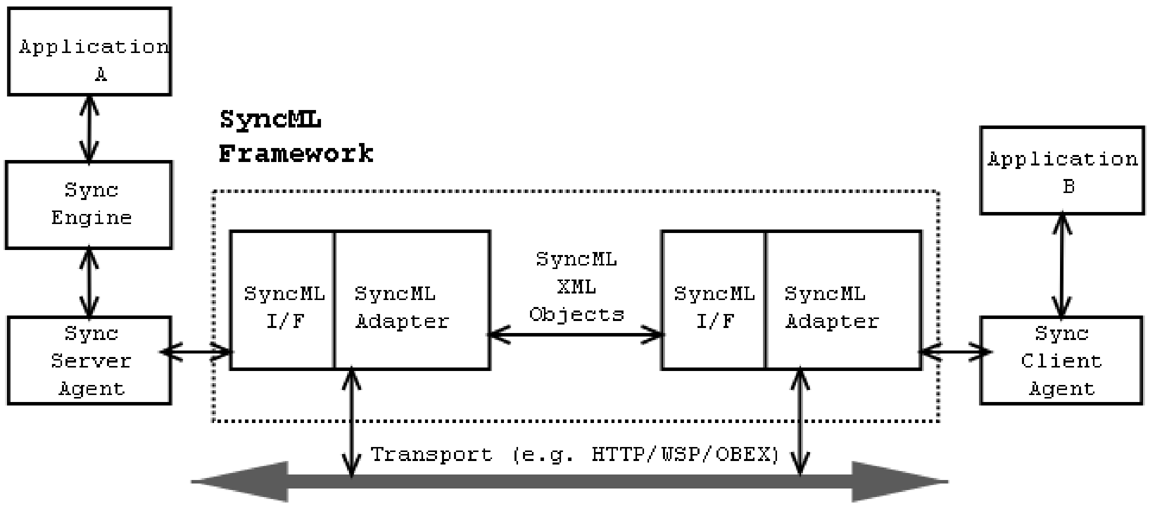


Figure 2: General concept of SyncML utilization. [1]

[5]Internet Engineering Task Force, a standardization organization

SyncML organizes the synchronization process using the client-server approach, with one exception: the server may also initiate communication. The typical architecture of a SyncML application is shown in Figure 2.

The client always sends his data first. Then the server compares the data for conflicts and sends back the result along with his own modifications to the client. Different sync types exist.

Fast sync is the normal type: The client only sends the items modified since the last synchronization session to the server. The server checks if items conflict with modifications on his data and tries to solve those conflicts. Then he sends his modification commands back to the client.

Slow sync is used when the devices lost track of what modified had already been synchronized, if the last synchronization was interrupted, or if they know by another way that their databases are inconsistent. The client has to send all his data, for the server to compare everything. The server tries to match the items with his items and see if they differ. Afterwards he sends the resulting modifications back to the client.

Other types are synchronization only from server to client or vice versa. They exist in fast and slow versions. The server could completely replace the inconsistent client data, for example.

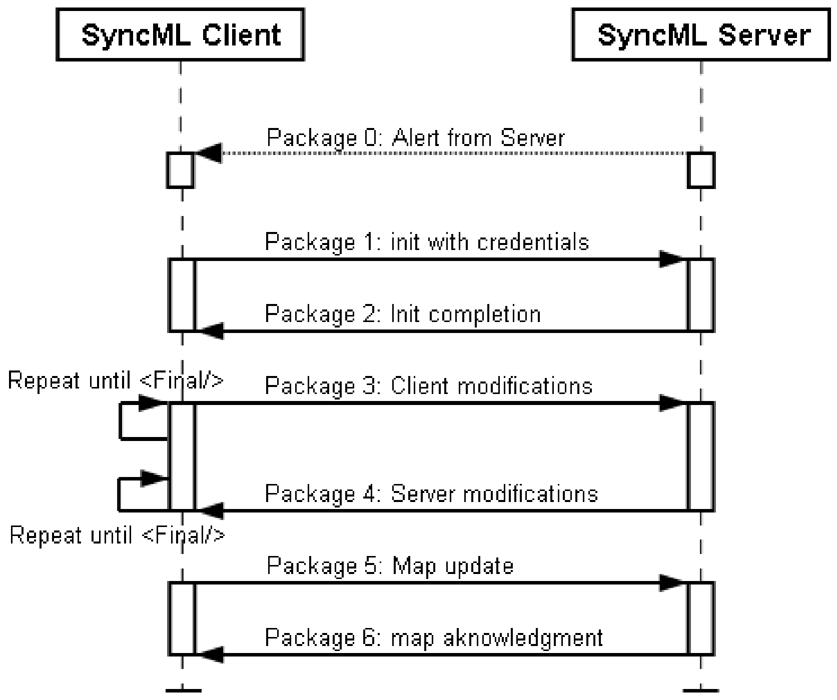

SyncML Client
SyncML Server
Package 0: Alert from Server
Package 1: init with credentials
Package 2: Init completion
Repeat until <Final/>
Package 3: Client modifications
Package 4: Server modifications
Repeat until <Final/>
Package 5: Map update
Package 6: map aknowledgment

Figure 3: Message flow of SyncML data synchronization. [1]

SyncML uses XML messages to exchange information. Figure 3 illustrates the message exchange of a synchronization: First, the client initializes synchronization by an

`<Alert>` command. This command tells the server with which database the client wants to synchronize and what sync type is wanted. The server replies by confirming or refusing the alert. In the next message, the client sends his modifications. After processing the received data, the server sends all resulting modifications to the client. The last message by the client is a *map update* (see Section 2.2.3). The server finishes synchronization (sync for short) with a *map acknowledgment*.

Because in some environments every message causes high costs, some steps can be merged into one. Status commands are included in the next message. The client may send his modifications along with initialization. If the credentials are accepted, the server already has the data and can send his modifications with the initialization completion. The mapping can be sent in the first message of the next session. This way, all sync can be done in one request-response cycle. To further improve the protocol for wireless devices, a definition for WAP Binary XML (WBXML) is given. WBXML is a compressed form of XML where the tags are no longer strings in angle brackets but a code. This reduces the size of an XML message considerably.

SyncML can be used over any transport layer. Good options include HTTP or better HTTPS, WSP (Wireless session protocol, part of WAP) or OBEX (IrDA, Bluetooth, . . .). But also SNMP / POP3 could be used, a good example for high packet costs.

A typical example of a SyncML message is depicted in Listing 1. For the exact document type definition (DTD) of SyncML, see [1].

Listing 1: Example of a SyncML message

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>SyncML/1.1</VerProto>
    <SessionID>1029143392160</SessionID>
    <MsgID>1</MsgID>
    <Target><LocURI>http://localhost/sync</LocURI></Target>
    <Source><LocURI>IMEI:123445</LocURI></Source>
    <Cred>
      <Meta>
        <Type xmlns='syncml:metinf'>syncml:auth-basic</Type>
        <Format>b64</Format>
      </Meta>
      <Data>dXNlcjpwYXNz</Data>
    </Cred>
  </SyncHdr>

  <SyncBody>
    <Alert>
      <CmdID>1</CmdID>
      <Data>200</Data>
      <Item>
        <Target><LocURI>server_db/</LocURI></Target>
        <Source><LocURI>dev_db/</LocURI></Source>
```

```
25        <Meta><Anchor xmlns='syncml:metinf'>
26          <Last>1004589132042</Last>
27          <Next>2029143393370</Next></Anchor>
28        </Meta>
29      </Item>
30    </Alert>
31
32    <Final/>
33  </SyncBody>
34 </SyncML>
```

This message is an initialization message. Lines 2 to 16 form the header and 18 to 33 the body. The header contains version information for the protocol, a unique session ID, the source and target URI and possibly a *Cred* element (lines 9-15). If present, the credential is used for the authentication. Three possible authentication schemes are defined: no authentication, basic and MD5 hash. Basic is the transmission of *username:password* (base 64 encoded). More secure is MD5, as it uses a nonce issued by the server and calculates a hash of *username:password:nonce*. This way, replay attacks can be prevented, as the nonce will change the authentication hash in each message.

The *Alert* (lines 19-30) tells the server that the client wants to sync his database `dev_db` with the server database `server_db`. If the client would like to sync different databases, more alert commands can be placed into the SyncBody.

The *Anchor* (line 25-27) is used to check if the last sync terminated correctly. After successful sync, the `<Next>` anchor is stored. On the next sync session, it is sent as `<Last>`. Fast sync is only possible if the anchor saved at the server is the same as the last anchor the client sent. If they differ, the last session did not terminate correctly and a slow sync is started.

`<Meta>` tags can be used at many places to specify additional information. In our example, line 10-13 specify the authentication type, the anchor is also a meta information. Other meta information are maximal message and record sizes, content data type and device information. The `DevInf` tag is used for device information. It contains details about device specific properties like which content types can be understand, memory limits or OS version.

2.2.3 Data synchronization

SyncML is oriented towards synchronization of small independent records, as the modified records are transmitted entirely. This is adequate for address entries, short messages and similar data. On the primary target of SyncML, mobile devices, most data is of this type.

The devices must be able to keep track which of their records have been changed. Each record is identified by a unique ID, so conflicts can be detected quite simple. As the record ID's may not be arbitrarily chosen but automatically created, mapping between server and client ID's is defined in the protocol. Mapping is always managed by the server. When the client receives a new item from the server, he can send a map update command to tell the server what ID he assigned to the item. Now the server uses the client ID in all his messages.

The specification does not restrict the format of payload data. A MIME type[6] can be provided to avoid confusion. The specification of SyncML declares that some types like vCard or vCalendar have to be supported mandatory. Hopefully, this will allow at least the synchronization of standard databases between all SyncML implementations.

The root of data synchronization is the `<Sync>` element. Inside the Sync, different commands to modify data can be used: `<Add>, <Delete>, <Replace>, <Copy>`. These commands contain a list of the items to be modified. Listing 2 shows a typical *Sync* command.

Listing 2: A Sync command issued by the server

```
<Sync>
  <CmdID>5</CmdID>
  <Target><LocURI>dev_db/</LocURI></Target>
  <Source><LocURI>server_db/</LocURI></Source>
  <Replace>
    <CmdID>6</CmdID>
    <Item>
      <Target><LocURI>c1</LocURI></Target>
      <Source><LocURI>1</LocURI></Source>
      <Meta><Type xmlns='syncml:devinf'>text/plain</Type></Meta>
      <Data>arbitrary data that replaces client item c1</Data>
    </Item>
  </Replace>
  <Delete>
    <CmdID>6</CmdID>
    <Item>
      <Target><LocURI>c3</LocURI></Target>
      <Source><LocURI>23ab3</LocURI></Source>
      <!-- data not needed, only id, as item is deleted -->
    </Item>
  </Delete>
</Sync>
```

Additionally, `<Atomic>` and `<Sequence>` can be used inside `<Sync>` to ensure all-or-nothing semantics respectively sequential processing of the contained commands.

Every sync command has to be confirmed with a `<Status>` command.

2.2.4 Device Management

With version 1.1 of SyncML, the device management (DM) protocol is introduced. In a large company, the administration of dozens or even hundreds of machines can be quite difficult. Automated administration tools exist for personal computers, but they are not adequate for devices not permanently connected to the network.

SyncML provides a general way to exchange configuration data. The approach seems heavily inspired by the Simple Network Management Protocol (SNMP), but in the official

[6] "Multipurpose Internet Mail Extensions" is a definition of how to declare the encoding and type of a document part. It was designed for e-mail but can be used for other protocols too.

specifications, there is no explicit reference to it. Table 2 shows the SNMP components and their corresponding SyncML names.

SNMP	**SyncML DM**	**Explanation**
Structure of Management Information (SMI)	Device Description Framework (DDF)[a]	Syntax rules for describing the MIB.
Managed Object (MO)	Management object	Manageable entity inside a device. May have children.
Management Information Base (MIB)	Management tree	Hierarchically organized values that can be stored and read.
SNMP agent	Management client	Software on device that interprets management commands and operates on the MO's.
SNMP manager	Management server	Controller for sending commands to the clients and interpreting their responses.
Message Authentication Code (MAC, SNMPv3)	Message Authentication Code (MAC)	A hash value of the message body and a shared secret to ensure data integrity.
User Based Security (SNMP version 3 only)	Access Control List and server identifier	Permission for specific operation can be restricted to some users.

[a]*Remark:* DDF documents should enable a management server to understand the meaning of device properties by himself. It is considered an interim solution until a more general standard arrives.

Table 2: Similarities between SNMP and SyncML DM

A DM session can be initiated either by client or by server. Because it's always the client who has to establish connection, the server can only send a notification to the client, triggering him to begin a management session. Therefore the server can only initiate DM if he is able to reach the client. Different protocols like WAP push, OBEX or TCP are possible. The client can also initiate management without prior notification from server. This can be caused by user interaction, when an internal timer is finished or if configuration errors are detected.

The message flow is depicted in Figure 4. Initialization is virtually the same as in a normal SyncML session, except that sending device information is not optional but required for the client. With the completion of initialization, the server sends his first management orders. The client processes the orders and sends status commands.

Packages 3 with management operations and 4 with confirmation are repeated until the server sends a message containing only commands requiring no response (e.g. Status commands).

An entire document of the specification tells how to bootstrap device management. Two scenarios handle the cases where a device gets SyncML DM bootstrap information installed by the manufacturer and where the server pushes the information onto the new device. This bootstrap is very important, because no help desk would like to explain hundreds of employees how to install and configure their device for SyncML usage. Its just extensive support needs like explaining everybody how to configure his device that

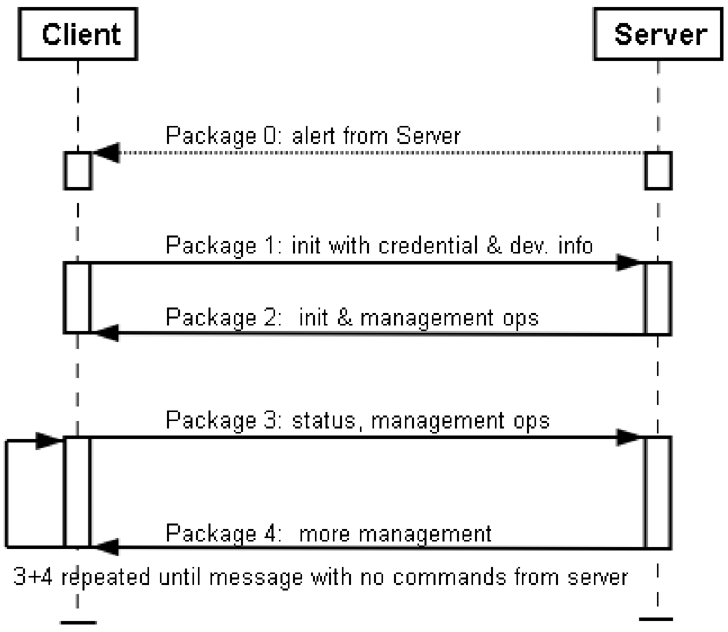

Client
Server
Package 0: alert from Server
Package 1: init with credential & dev. info
Package 2: init & management ops
Package 3: status, management ops
Package 4: more management
3+4 repeated until message with no commands from server

Figure 4: Message flow in device management. [1]

SyncML DM should replace.

Another document describes the security mechanisms. SyncML DM only offers authentication and integrity mechanisms. It can ensure no malicious configuration messages are accepted, but confidentiality (e.g. encryption) is not provided. The document recommends using encrypted protocols like HTTPS or TLS. Also, the configuration information itself can be in an encrypted format, if both client and server support such a format. This last option seems not very helpful as the whole structure of the management message would be readable by an attacker.

The author did not make any programming with the DM part of SyncML. But it seems astonishing that things like Access Control List (ACL) are defined in a non-XML format[7] and transmitted as text inside the configuration messages. There is no explanation why XML is not used in such cases.

2.2.5 Differences between version 1.0.1 and 1.1

Version 1.1 contains no fundamental changes to the data synchronization protocol, but it adds the new area of device management (see earlier in this chapter).

Most of the changes to the existing documents are error corrections, clarifications and adjustments for the device management protocol[8]. In version 1.0.1, some element descriptions differed between text and DTD and several typing errors were made in the examples.

An important addition to the data synchronization protocol is the introduction of large object handling. There is still no way of sending only parts of an item, but it can be split over several messages with help of the `<MoreData>` tag. This is especially needed if the

[7]See [1], `syncml_dm_tnd_v11_20020215.pdf` , page 17: ACL syntax

[8]For criticisms on those adjustments see Chapter 5.1

message size is limited, e.g. by the underlying transport protocol or by device restrictions.

Further, a `<NumberOfChanges>` tag can help to make an estimation for the sync duration or to show a progress bar. This is not really essential, but with lots of data to synchronize, users could loose patience if they see no progress.

2.3 SyncML market perspectives

To learn about the perspectives of SyncML, an enquiry has been held in early 2002 among companies and institutions concerned with informatics. At this time, the SyncML specification had been publicly available for one year. The goals were to see how much interest there is for SyncML and what products exist. This section gives a short overview of the enquiry results. The complete enquiry with in deep interpretation and diagrams of the answers is in a separate document (see Appendix B).

About half of the companies were selected to participate because they are known to do SyncML development. Their addresses were found on `http://www.syncml.org`. The others are members of the IT Valley Fribourg association (`http://www.itvalleyfribourg.ch`). A total of 274 companies had been invited by e-mail to participate in the enquiry by filling out a web form. About 50 of them gave a usable answer within the deadline. Unfortunately, the big companies who are part of the SyncML organization did not answer. This could mean low interest - or just bad communications.

2.3.1 Results of the enquiry

It appeared that although some technical and organizational problems exist, the majority of the companies gives SyncML a good chance - under the condition that it gets widely used.

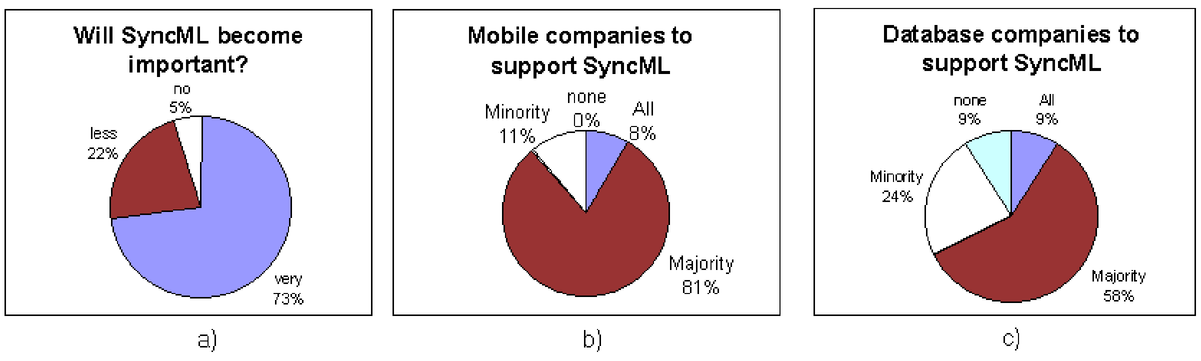


Figure 5: Assumed importance of SyncML

From the perspective of the answering companies, SyncML seems to be rather important[9]. Figure 5 shows the answers on the questions about the importance of SyncML and the expected support by mobile devices manufacturers and back end systems.

[9]The euphoric impression Figure 5 gives is not entirely representative, as companies not expecting SyncML to be important probably did not answer to the enquiry.

The question for the main benefits of SyncML revealed that one of the major advantages of SyncML is its aim to become an open de-facto standard. The companies see it as a good base for their sync software development.

The biggest problem of SyncML is to become widely accepted. The technology will only be really interesting if most of the mobile devices contain a SyncML client and back-end systems support it too. There have been some complaints about the implementation also, especially about usability and access control.

The questions about product development revealed a number of devices with SyncML support and some SyncML servers. Most of them were not tested for compliance, but the companies planned to let the SyncML committee test them. The servers are stand alone products. What seems to be missing are extensions to industry standard systems like Oracle or SAP. For private users, a small stand alone server is exactly what they need, but companies need to synchronize directly into their back end system.

The final part of the enquiry asked for the marketing interest in SyncML, revealing a rather small interest from private customers, but a considerable interest for business customers.

2.3.2 Conclusion

If everybody is using the same protocol, servers and clients can easily be replaced. This is good for customers, but a challenge for the manufacturers. They can not sell the simple fact that their system can synchronize, but the whole service must convince the customers. This should improve the competition among manufacturers.

As SyncML might not completely replace proprietary protocols, interoperability with such protocols could be an important field. But some enquiry answers indicate that SyncML seems not prepared for this challenge. Perhaps this could be an extension for a next version of SyncML.

There is no big hype about SyncML, not like, for example, WAP or SOAP. Most end users never heard about it. The primary interest lies in business customers with many employees using corporate date and address management.

With more and more gadgets and services being used, growing needs for a general standard are to expect for private users too. Syncing one mobile phone with Microsoft Outlook on a PC may work well. But with each family member having his personal phone and computer plus some PDA's plus a web-based PIM, there are too many combinations for proprietary protocols. There is no other common standard for synchronization in sight. If SyncML succeeds getting over a critical mass, it may become commonly used as *the* standard for mobile synchronization.

2.4 Implementations

On page 12 of their whitepaper[1], the SyncML initiative promises to deliver "An openly available prototype implementation of the protocol". For SyncML 1.0.1, the SyncML initiative provided a reference toolkit written in C on their web site. It implements something like the core layer of sync4j (see Section 3.1 and Figure 6). Programming with it requires a good knowledge of SyncML.

But with the publishing of version 1.1, no new toolkit appeared. Even the old 1.0.1 version was removed from the server and no longer available to the public.[10]

Until recently, the new toolkit was only available to paying members of SyncML. [4] says only promoter members (20,000 $ per year) had access to it. Shortly before finishing this thesis, on august, 20th 2002, the committee seems to have reconsidered its position and announced to release the toolkit as open source.

2.4.1 Commercial offers

Starfish, a founding member of SyncML, offers its system TrueSync. The solution based on a proprietary protocol and has been extended with SyncML support. But Starfish does not sell its SyncML implementation, nor do they sell any tools directly to end users.

2.4.2 Open source implementations

Some open source implementations exist. *libsyncml* [4] is a C++ library for the lower levels of the protocol, using a callback mechanism. It is available on [5]. The project is in alpha phase, last updates were in April. The enhydra project has a Java implementation of SyncML too. It is available as *kSync* at [6]. kSync doesn't support the full SyncML protocol and development seems to have stopped, last updates were in November 2001 and the CVS repository is empty.

Last but not least, the sync4j project exists. This is subject of the next chapter.

[10]The toolkit is on the CD in the `Documents/SyncML/1.0.1/` directory. The license for the source code states it may distributed freely. The copy information of the toolkit *manual*, however, says the document may not be distributed and to obtain it, you would have to sign a non disclosure agreement. (I downloaded it without any problems form www.syncml.org .)

Chapter 3

Java implementation of SyncML: sync4j

The open source project sync4j wants to build a SyncML application framework for client and server in Java. It is available at `http://sync4j.sourceforge.net`. Unfortunately it is not yet complete. As part of this diploma thesis, the author worked on sync4j, achieving a running prototype.

The first section of this chapter describes the concepts provided by Sean C. Sullivan and the existing framework written by the sync4j developer group. The second section describes the steps by the author to create a running prototype. How to run the prototype is described in the last section.

3.1 The existing framework

Figure 6 gives an overview of sync4j. It is divided into four layers with different roles. Details on the classes are found in the generated javadoc on CD (see Appendix A).

3.1.1 Transport

The whole message is encoded for network transmission in the transport layer. The only implemented transport is HTTP. The client uses a `java.net.HttpURLConnection` to issue a POST command and the server side is a servlet[11].

3.1.2 Core

The core contains classes for all XML elements in SyncML. Hence it serves as a full abstraction of the XML nature of the protocol. The classes can be constructed from XML fragments and can generate their XML representation. The classes for error messages are implemented as `Exception`, so they can be thrown.

Several corrections to the core were done by the author. Some classes did not generate the correct XML, resulting in invalid messages that could not be parsed. The helper class `sync4j.core.Util` contained errors regarding namespaces. Two exceptions had to be

[11]Servlets are small Java programs running in a "JavaServer" environment. See [7] for more information about servlets.

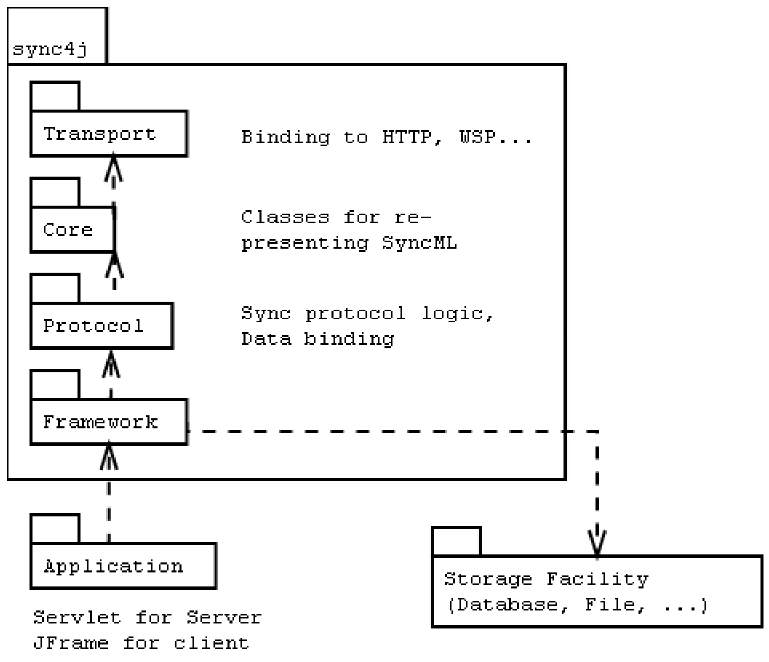


Figure 6: Sync4j architecture

removed as they were double, `UnauthorizedException` (= `ForbiddenException`) and `VersionNotSupportedException` (= `DTDVersionNotSupportedException`).

3.1.3 Protocol

On protocol layer, there were no classes existing at the time the author began his work on sync4j. This layer is concerned with the elements of the synchronization (i.e. initialization, modifications, ...) and the protocol work flow.

3.1.4 Framework

The framework consists of JavaServer classes for the server and a GUI for the client. The client side consists only of empty classes, on server side a EJB (Enterprise Java Bean) exists.

The author had to rewrite parts of the server because EJB are only supported by application servers and not by the Apache Tomcat used.

3.2 Concept of the prototype

To get a running prototype, the protocol layer had to be implemented by the author. Planning was done together with Nicola Fankhauser at the University of Fribourg, but most implementation was made by the author of this diploma thesis. For details on who wrote which class, see the *author* lines in the javadoc of sync4j.

One of the most important concerns for the prototype was the organization of data access. Figure 7 shows the big picture. Interfaces are used for retrieving and storing data and mapping as well for authentication and for the synchronization strategy.

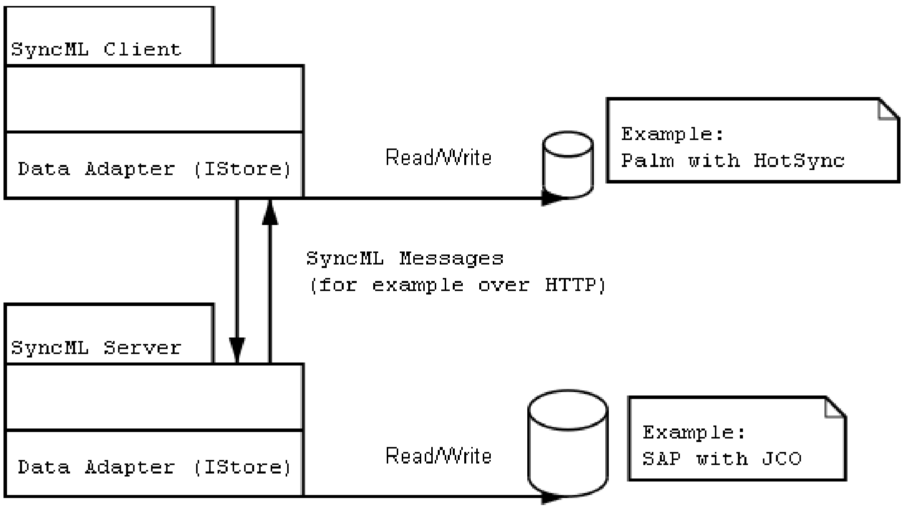


Figure 7: General concept of sync4j

The protocol layer is split up into two parts. The lower part is called synchronization engine, consisting mainly of different interfaces. The higher part is the handler, concerned with generating the protocol flow and is using the interfaces of the sync engine.

3.2.1 Synchronization Engine

The interfaces for accessing data and mapping are instantiated using the design pattern "Factory" [12]. This pattern allows data access to be implemented for whatever storage system needed. The interfaces architecture is shown in Figure 8.

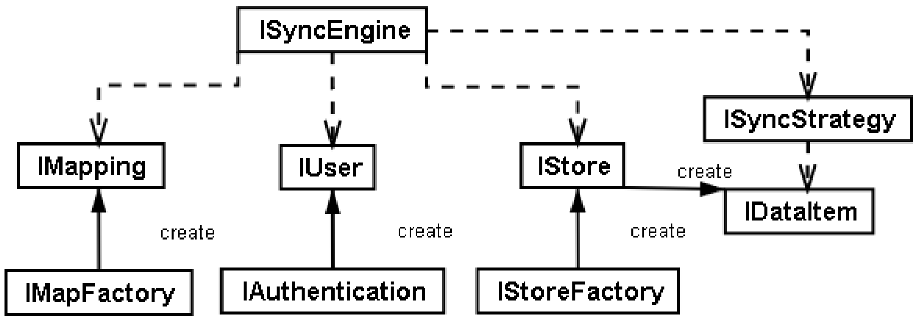


Figure 8: Architecture of the sync engine

Store factory Creates `IStore` objects for the database specified by an URL and the optional mime type. The store objects contain methods to add update or remove records from the database and to get all resp. all modified items. A store can be implemented using an XML file or JDBC or whatever the application needs.
In the concrete example, a Palm handheld computer and SAP R/3 are used as data source.

[12]See [9] for design patterns

Map factory Creates `IMapping` objects for a database. Mapping handles the conversion of item ID's between client and server.

Strategy Conflicting items are passed to the method `solve()` of the strategy. To solve the conflict, the strategy can test if it can merge the items, always let win the client item or using the input alerts of SyncML to ask the user for correct values.
There are two strategies implemented in the prototype: MergeOrFail and AskTheUser. MergeOrFail tries to merge the conflicting items. If this is not possible, it returns a conflict error status code. AskTheUser first tries to merge as well. If this is not possible, it issues an input alert command, asking the client which one is the correct data.

IDataItem Interface for easy interaction between sync engine, store and strategy. Contains payload data, record ID and methods to check if two items are different and if they can be merged. These last methods allow the sync strategy to be independent of the concrete data types.[13]

Authentication Creates objects representing the user for permission checking from the `<Cred>` element. Stores must always ask the user object whether the specific command is permitted. Different roles can be created by having the authentication class selecting different `IUser` implementations. The authentication can ask for a specific scheme (e.g. no auth, basic and md5 digest) and checks the supplied username/password. Thus, sync4j can work with any existing user authentication system that can be accessed from within Java.
The authentication interface even allows authentication per database, but this is not implemented in the rest of the prototype.

3.2.2 Handlers for client and server

The handler has to control communication flow. Figure 9 shows the message exchange. To reduce the number of messages, some can be combined. The thin arrows show which messages are combined and which states receive a combined message. It is possible that more than one message is sent in one state. In this case, state transition will wait and messages are not combined.

From the message flow in Figure 9, the state diagram is deduced. Figure 10 shows the states client and server have to pass.

It is interesting to notice that client and server have similar states. Sending and receiving modifications is about the same for client and server. Using a special mapping that creates map commands instead of storing the ID's and a dummy strategy that does nothing, the client uses the same code for sending and receiving modifications as the server.

[13]The store knows which implementation of IDataItem must be instantiated for this database. The item implementation knows the data type and thus if and how items of this type can be merged.

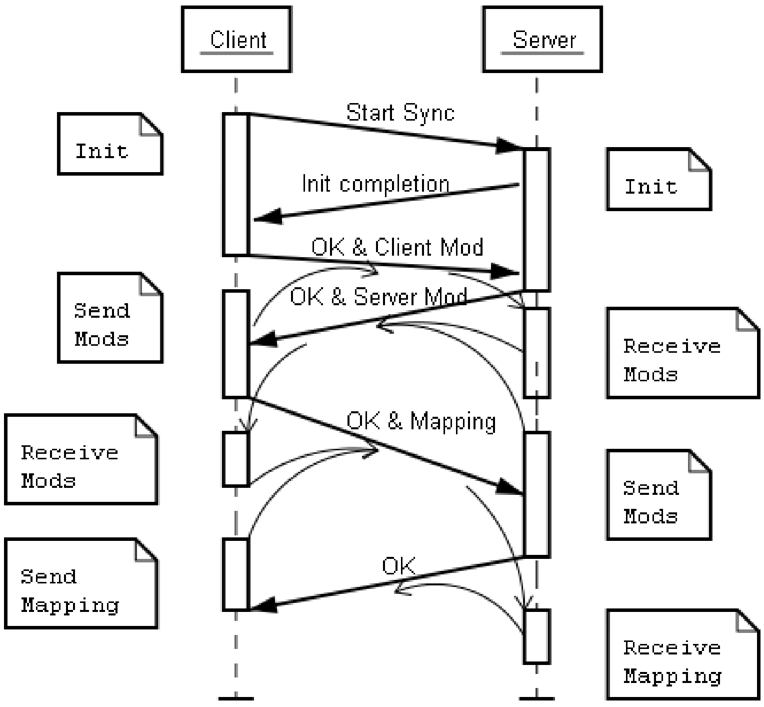


Figure 9: Communication flow in a synchronization

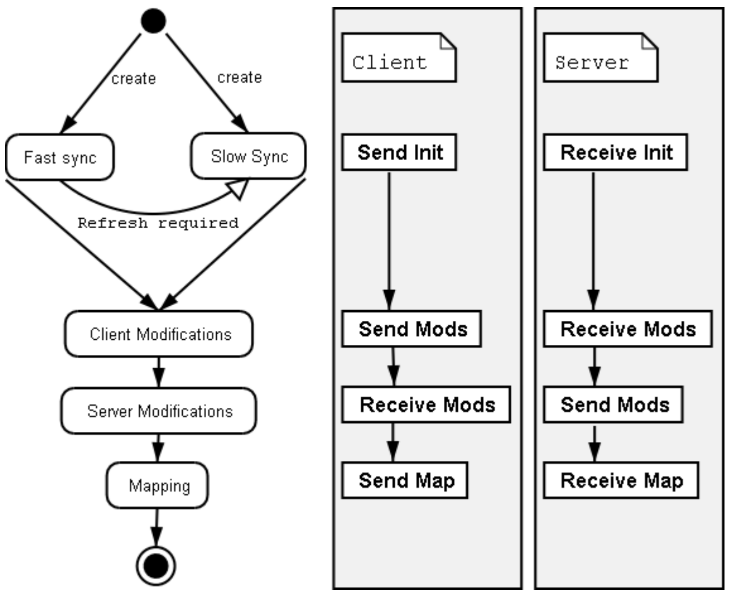


Figure 10: Handler states

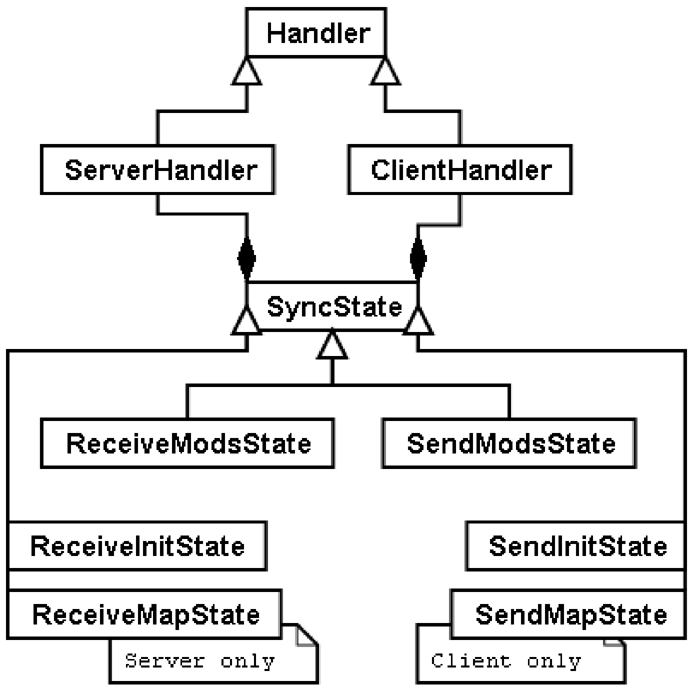


Figure 11: Handler architecture

Figure 11 shows the architecture of the handler. Server as well as client have a handler class to control state transition and status of synchronization. The server and client handlers have little differences, most code is in the common base class. The most important difference is that the order of state transition is not the same.

Each state is implemented as separate class. As noticed before, sending and receiving modifications is the same for both client and server. Both use `SendModsState` and `ReceiveModsState`. The other states are different, as the client sends init while the server has to receive init, and the client sends mapping and the server has to store it. The states have a common base class which does most of the work.

3.2.3 Exception handling

The exception concept is extended. To the SyncML exceptions in the core layer, additional exceptions for the sync engine and the handler have been added. Figure 12 shows the new hierarchy.

The meaning of all core exceptions is described in the SyncML reference, as each exception corresponds to an error code. But the sync engine and handler exceptions need some explanation.

Syncengine exceptions tell in which part the problem occurred. Each part (data, mapping, authentication and strategy) has his exception. Core exceptions can not be thrown by the classes implementing the interfaces. If core exceptions occur, they must be wrapped inside a `SyncengineException`.

The *handler exceptions* are not grouped by functional parts. Rather, they tell to which level the synchronization must fail because of the exception. They can contain any `Sync4jException`. If the contained exception is a `SyncMLException`, its error code is

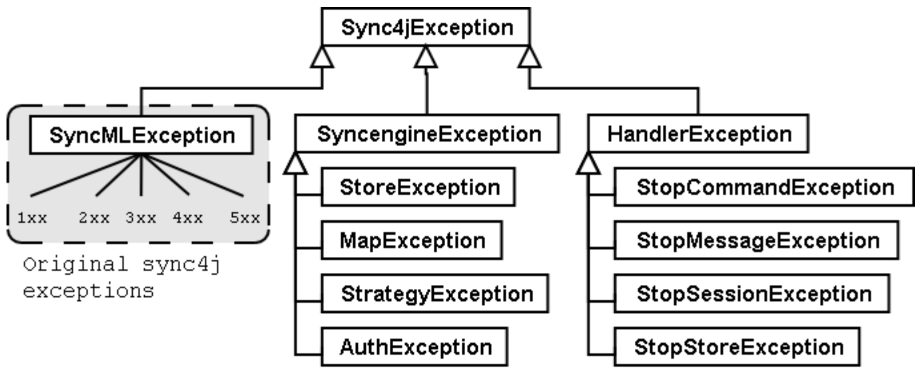


Figure 12: Exception hierarchy

used in the answer, otherwise an internal failure is reported. Some of the sync engine exceptions can be used to give precise internal failure codes (i.e. store failure).

3.3 How to run the simple test

The CD contains the complete sync4j code. A simple test synchronization can be run using the test client `sync4j.tests.protocol.handler.ClientHandlerTest` and the server running inside any servlet container.

For convenience, a ready to run Apache Tomcat is on the CD to. It is important to copy the `SyncML/Apache_Tomcat_4.0/` directory as well as the whole `SyncML/sync4j/` directory to a writable disk, as server and client need to write files.

3.3.1 Server

The server is started with the batch file `startup.bat` on Windows, `startup.sh` under Unix/Linux[14]. The scripts are located in Tomcat's bin directory.

The resulting files are stored in `webapps/syncml/WEB-INF/stores/` . To view the messages exchanged in this session, take a look at the log files in the subdirectory `logs` of the stores folder.

3.3.2 Client

When the server is running, the client can be run using `testClient.bat` respectively `testClient.sh` in the `src/tests/` directory. The store files are in the same directory.

The other batch files call JUnit test cases running without the server.

[14]If the shell script is not executable after copying from CD, use `chmod u+x startup.sh`.

Chapter 4

Practical example

As a concrete example, sync4j is used to synchronize time reports between a Palm handheld computer and a SAP R/3 backend. As mentioned in Section 1.2, the project will consist of two parts. The "TiME" application will run on the Palm computer, providing means reporting time. The sync4j client will be used to synchronize the reports from a PC connected to the Palm with a Java server running the sync4j server.

This system will be used by the company ESPRiT Consulting AG.

4.1 Concepts

The concept for the TiME application is deduced from the SAP time reporting system (CATS), as the information must be compatible with this system. Figure 13 shows the SAP dialog for entering working time. Basically, time is reported by working duration. Every entry must be associated with either a costcenter (KS, internal costs payed by the enterprise) or a project element (PSP, payed by the project's budget).

What is needed on the Palm are the lists of projects, costcenters and time reports.

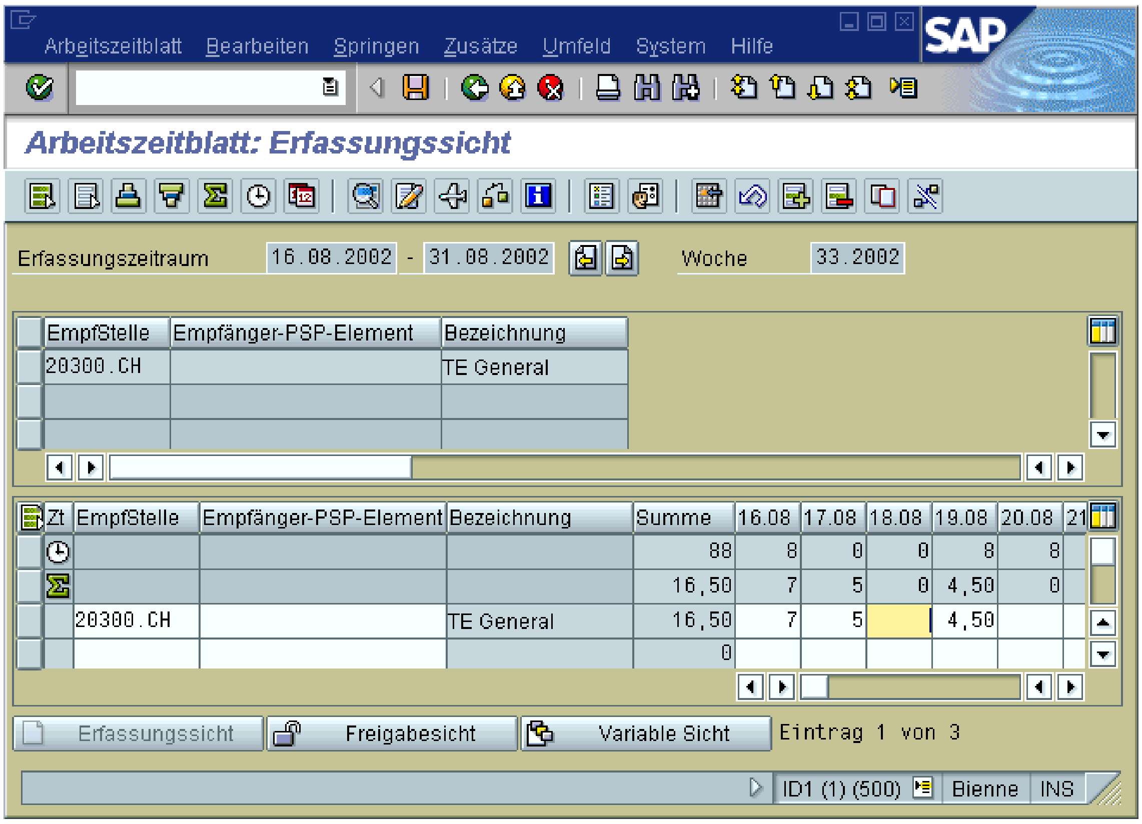
4.1.1 Data structures

For the interface, the notion of a period is used. Working time is entered, confirmed and controlled per period. The methods for manipulating time entries work on single days, not on periods. The periods are only a user interface construct with no impact on the underlying time reporting data and its synchronization. We will use one item per day and KS/PSP.

Time entries contain a reference to a KS or PSP as foreign key. All other fields than the name of KS / PSP are used only as information for the user to select the correct target to charge on.

SAP communication

To communicate with SAP from within an external application, the Business Application Programmers Interface (BAPI) is used. The BAPI methods are defined by SAP and guaranteed to stay unchanged in their functionality over several versions. Table 3 lists the BAPI methods used in the TiME application.



Arbeitszeitblatt: Erfassungssicht

Erfassungszeitraum 16.08.2002 - 31.08.2002 Woche 33.2002

EmpfStelle	Empfänger-PSP-Element	Bezeichnung
20300.CH		TE General

Zt	EmpfStelle	Empfänger-PSP-Element	Bezeichnung	Summe	16.08	17.08	18.08	19.08	20.08	21
				88	8	0	0	8	8	
				16,50	7	5	0	4,50	0	
	20300.CH		TE General	16,50	7	5		4,50		
				0						

Eintrag 1 von 3

ID1 (1) (500) Bienne INS

Figure 13: SAP dialog for time reporting (Application CAT2).

Reading costcenters	
BAPI_COSTCENTER_GETLIST1	Read the list of costcenters

Reading Projects	
BAPI_PROJECTDEF_GETLIST	Read the list of projects
BAPI_PROJECT_GETINFO	Get details of a project (i.e. project elements)

Reading and manipulating time entries	
BAPI_CATIMESHEETRECORD_GETLIST	Get list of entries
BAPI_CATIMESHEETMGR_INSERT	Insert a new entry
BAPI_CATIMESHEETMGR_CHANGE	Modify an existing entry
BAPI_CATIMESHEETMGR_DELETE	Remove an entry

Table 3: BAPI's used for synchronizing.

Projects and costcenters are only read and never written, because the Palm application is for time reporting only and not for managing projects. Time records are read, inserted, changed and deleted, each operation with a different BAPI.

The BAPI's listed in Table 3 have input and output parameters. Table 4 lists the important fields found in the time report manipulation system.

BAPI_CATIMESHEETRECORD_GETLIST allows to select any employee. In the TiME application, only the currently synchronizing employee will be selected in each request.

In BAPI_CATIMESHEETMGR_INSERT the employee number is specified, and here too, only the current employee is chosen.

Description	**Field name**	**Remark**
BAPI_CATIMESHEETRECORD_GETLIST		
From date	FROMDATE	
To date	TODATE	
Employee selection	SEL_EMPLOYEE	Table to define search criteria.
Results are in a CATSRECORDS table.		
BAPI_CATIMESHEETMGR_INSERT and - CHANGE		
Reporting profile	PROFILE	Always "Consult" for Esprit
Testrun	TESTRUN	Whether changes should be made
Release data	RELEASE_DATA	Whether to confirm time data
Time entries	CATSRECORDS_IN	Table of working times (see tab. 5)
BAPI_CATIMESHEETMGR_DELETE		
Reporting profile	PROFILE	Always "Consult" for Esprit
Testrun	TESTRUN	Whether changes should be made
Time entries	CATSRECORDS	Table of entry numbers to delete

Table 4: Fields of the BAPI's for time report manipulation.

Table 5 lists the fields of the Catsrecords table used wherever time reports are read or written.

Table 6 contains two BAPI's for reading costcenters and projects. Costcenters can be read directly, while reading projects requires two steps. Each project contains a list of project elements. GETLIST returns the list of projects. GETINFO is called for each project to get its elements. Work can not be charged on projects directly but on a project element.

Description	**Field name**	**Remark**
Unique ID	COUNTER	Not available in INSERT
Work date	WORKDATE	Date the work was done
Employee number	EMPLOYEE	The employee this entry is for
Receiving costcenter	REC_CCTR	Only if charging on costcenter
Project element	WBS_ELEMENT	Only if charging on project
Working time	CATSHOURS	Duration of work

Table 5: Fields of CATSRECORDS

Description	**Field name**	**Remark**
BAPI_COSTCENTER_GETLIST1		
Controlling area	CONTROLLINGAREA	From value
Returns a list of costcenters with name, description and CO_AREA		
BAPI_PROJECTDEF_GETLIST		
Maximal rows to be returned	MAX_ROWS	Not used
BAPI_PROJECT_GETINFO		
Project key	PROJECT_DEFINITION	Returns all elements of specified project
Returned is a list of project elements with their description, cost area, business area, profit center etc		

Table 6: Fields of BAPI's for getting KS and PSP.

DTD for content data

SyncML defines everything except the payload data. It seems a good choice to define a XML representation of this data as the rest of the protocol is XML too. (But any custom format would do.) The document type definitions (DTD) of the Listings 3-5 describe the items. These DTD's are not used inside the code, as the elements never occur in an own document, but are always included inside a SyncML message.

Listing 3: DTD for time entries

```
<!ELEMENT Entry (DateInfo, refType, refID, minutes)>
<!ELEMENT DateInfo EMPTY>
<!ATTLIST DateInfo year CDATA #REQUIRED
                   month CDATA #REQUIRED
                   day CDATA #REQUIRED>
<!ELEMENT refType #PCDATA> <!-- only 1 (KS) or 2 (PSP) -->
<!ELEMENT refID #PCDATA>   <!-- integer -->
<!ELEMENT minutes #PCDATA> <!-- integer between 0 and 1440 -->
```

Listing 4: DTD for costcenter

```
<!ELEMENT KS (ID, COSTCENTER, NAME, DESCRIPT,
              CO_AREA, BOOKABLE?)>
<!ELEMENT ID #PCDATA> <!-- integer, unique for all KS -->
<!ELEMENT COSTCENTER #PCDATA>
<!ELEMENT NAME #PCDATA>
<!ELEMENT DESCRIPT #PCDATA>
<!ELEMENT CO_AREA #PCDATA>
<!ELEMENT BOOKABLE EMPTY>
```

Listing 5: DTD for projects

```
<!ELEMENT PSP (ID, WBS_ELEMENT, DESCRIPTION, PROJECT_DEFINITION,
          CO_AREA, COMP_CODE, BUS_AREA?, PROFIT_CTR?, BOOKABLE?)>
<!ELEMENT ID #PCDATA> <!-- integer, unique for all PSP -->
<!ELEMENT WBS_ELEMENT #PCDATA>
<!ELEMENT DESCRIPTION #PCDATA>
<!ELEMENT PROJECT_DEFINITION #PCDATA>
<!ELEMENT CO_AREA #PCDATA>
<!ELEMENT COMP_CODE #PCDATA>
<!ELEMENT BUS_AREA #PCDATA>
<!ELEMENT PROFIT_CTR #PCDATA>
<!ELEMENT BOOKABLE #PCDATA>
```

KS stands for "Kostenstelle", the German translation of costcenter. The key of a project element is called WBS_ELEMENT in the important BAPI, so this name is used here too.

As can be seen in Listing 3, costcenter and projects are not referenced the same way SAP does. SAP has fields for both, selecting is done by filling in the right field (see Table 5). TiME uses a different scheme, as the string references would not work well on Palm. A reference ID and a type telling if it references a KS or a PSP are used. Figure 14 shows an entity relationship view of this data. The field names are taken from the BAPI's.

The ID's are not the SyncML ID's, as the later may be changed by the client. It would be very complicated and error-prone if the references in a time entry would have to be changed on a map update for a KS/PSP. The SyncML server must create unique reference codes for all KS and PSP. He actually uses a hash code on the PSP/KS name, thus the references can be recreated even if the server lost his data.

Palm structures

The records on Palm are created with fields in the same order as shown in the entity-relationship diagram of Figure 14. All fields of the working time entries are of fixed length, making it simple to access them with a C struct. DateInfo is written as integer with 32 bits. The first 16 bit are used for the year, then eight bit for month and eight for day. This way records can be sorted on the resulting 32 bit - integer.

The fields of KS are semantically the same as the first six of PSP, allowing a common base class. All fields except for ID and BOOKABLE are character strings of variable length. This means the fields can not be found at a fixed position but the record must be read sequentially. As the fields must have the possibility to be long but often are short or even empty, lots of memory is saved.

At the time of this writing, ESPRiT has about 400 costcenters and 100 projects. As they can not be deleted in SAP, their number will increase. The synchronization should take care only to transmit costcenters and projects that make sense for the user.

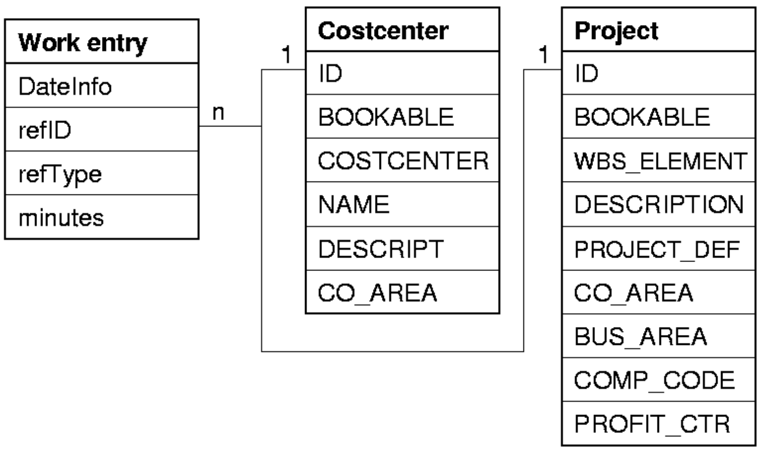
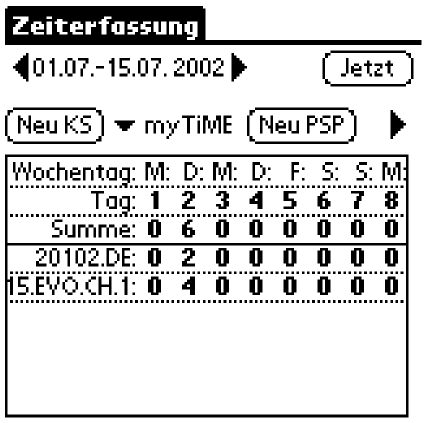


Figure 14: Entity-relationship diagram of time data.

Connection information for synchronization is configured on Palm. This way, it will be easier to implement the SyncML client directly on the Palm one day. Each line is a record by itself. The conduit reads it and stores it as a properties file. No XML format is needed as the connection information is not transmitted in the sync but used to establish sync.



Wochentag:	M:	D:	M:	D:	F:	S:	S:	M:
Tag:	1	2	3	4	5	6	7	8
Summe:	0	6	0	0	0	0	0	0
20102.DE:	0	2	0	0	0	0	0	0
15.EVO.CH.1:	0	4	0	0	0	0	0	0

Figure 15: Main screen of the Palm application

The Palm application main screen (Figure 15) has similarities with the SAP dialog, but also some extra features like the favorites.

Figure 16 shows the communication network for synchronization of a Palm with SAP R/3 using the SyncML conduit and the SyncML servlet. If we write a SyncML application for the Palm, the HotSync and conduit can be removed without any problems, as the protocol to the server is standardized. We could even synchronize the Palm to other systems than SAP without modifying the client, or write a Java application for time reporting that runs on a computer without touching the server.

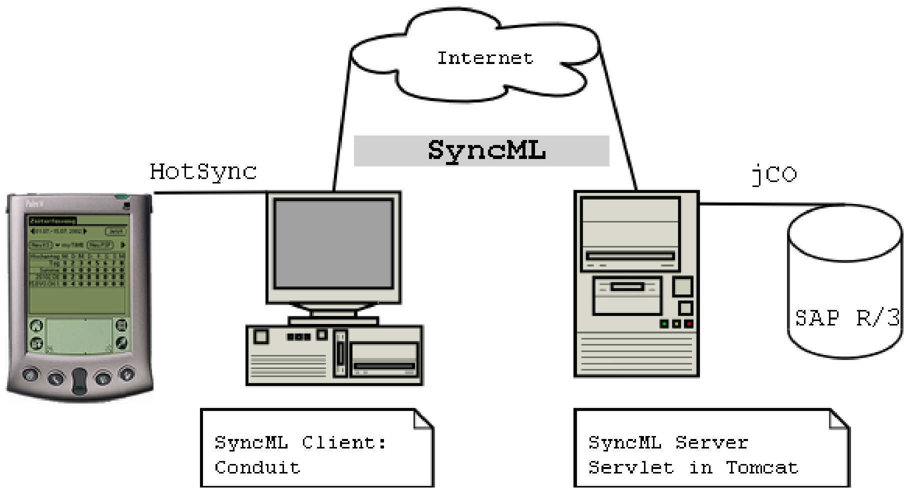


Figure 16: Synchronization flow of ESPRiT TiME.

4.2 Client application with Palm

Palm computers were developed for being as small and inexpensive as possible. Compared to the Windows CE devices and, of course, desktop PC's, Palm computers have a slow processor and little memory. They are suited as mobile agenda and notepad, and from the first devices on, interaction with personal computers was possible. Everything needed for data exchange is included in the price of a Palm. The 'Cradle' is recharging station and synchronization adapter. Synchronization is also possible via infrared port (IrDA), which is present in newer devices.

The program HotSync is used on the PC side. It waits on the serial, USB or IrDA port for the Palm to start synchronization. Each Palm program has its corresponding *Conduit*. All synchronization logic lies within the conduit, the Palm is only used to read data. Those small software parts registered in the HotSync Manager get called one after another during synchronization. HotSync gives the conduits access to the Palm data.

Custom conduits can be added to change the synchronization for an existing application or to add synchronization for a new application. The standard language for HotSync conduits is C++, but they can be written in Java too. In this case, the JSync extension to HotSync must be installed.

4.2.1 Getting sync4j to use a Palm

It was rather simple to enable sync4j using a Palm. The only thing to be done was writing a store that reads from and writes to the Palm. The main client application is a conduit. It acts as a SyncML client, using the sync4j library with a `PalmStoreFactory` that creates stores which can access the Palm.

To install the conduit and the Palm application, Conduit Installer 1.6.1 by handX[18] is used .

For information on the time reporting application, see the documents mentioned in the Appendix A. Resources for Palm programming from the manufacturer himself are available at [16], a good book about Palm and Conduit programming is [17].

4.3 Server application with SAP R/3

The server is running as a web application inside a java webserver. To understand the following chapter, you need basic knowledge of java servlets [10].

The sync4j server framework is slightly extended. The most important features are:

- The factory for Mapping/Stores is a servlet instantiated at webserver startup.
- Costcenters and Projects are not directly read from SAP. A cache instantiated at startup and controlled by the factory servlet holds the SAP data. The KS/PSP Stores query the cache.
- Time reports are directly read from and written to SAP, without caching.
- Dirty items are identified using timestamps. Every item has a last modified timestamp. The stores have a timestamp of the last successful synchronization. Everything modified since the last sync is dirty.

4.3.1 Server architecture

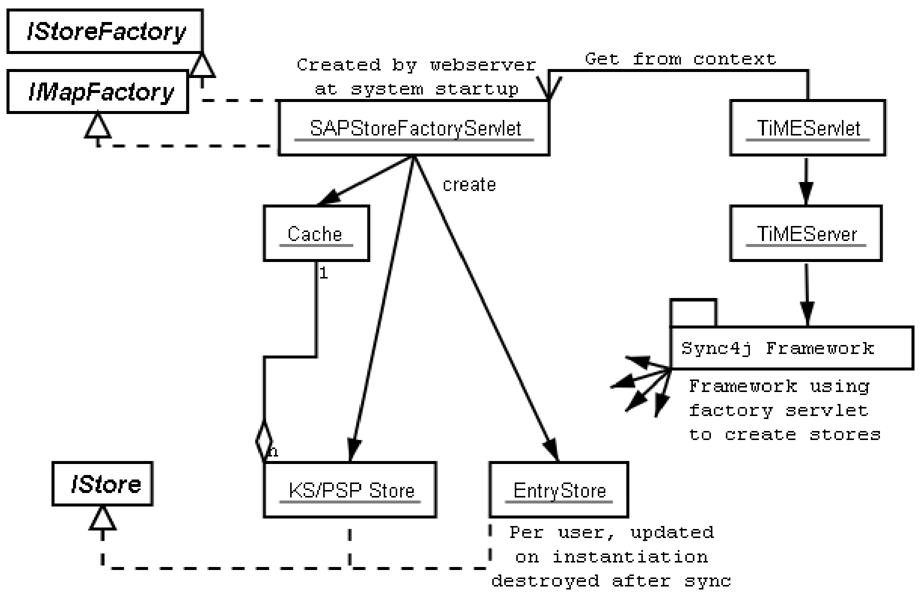

IStoreFactory
IMapFactory
Created by webserver at system startup
Get from context
SAPStoreFactoryServlet
TiMEServlet
create
Cache
1
TiMEServer
Sync4j Framework
Framework using factory servlet to create stores
n
IStore
KS/PSP Store
EntryStore
Per user, updated on instantiation destroyed after sync

Figure 17: Architecture of the server.

Figure 17 shows the TiME SyncML server. In the web.xml configuration file, the servlet container is told to instantiate the `SAPStoreFactoryServlet` on startup. This servlet is factory for both stores and mappings. On instantiation, it creates the caches and updates them for the first time. Caching is explained more detailed further down this chapter. The factory servlet finally places itself into the servlet context.

The servlet `TiMEServlet` receives the synchronization messages. It does session management and retrieves the factory from its context. To continue a session, it looks up the `TiMEServer` instance, if none is found, a TiMEServer is instantiated and a new session created. The TiMEServer passes the factory servlet and an authentication implementation for SAP (described below) to the framework.

4.3.2 The caching mechanism

Reading costcenters and projects from SAP takes a long time[15]. But it is an other reason that makes it really mandatory to cache the data. We only want to synchronize the modified items with the client. The author found no way to get the time of last modification from SAP. As a work around, a copy of the data is stored locally on the server in XML files, containing a last modification timestamp. Updating the cache is done by reading everything from SAP and comparing it to the local data. If an item changed, it is replaced with the SAP data and the timestamp set to the current time.[16]

Figure 18 shows the caching concept. The caches for costcenters and projects have only one instance, as their data is the same for all clients. To avoid unintended multiple

[15]For ESPRiT Consulting, there are about 100 projects and 400 costcenters, taking about 30 seconds. Other enterprises may have many more.

[16]For getting the current time, the java.util.Date class is used. This class depends on the system clock. If the system clock runs false more than a day, synchronization will work on past or future periods instead of the current.

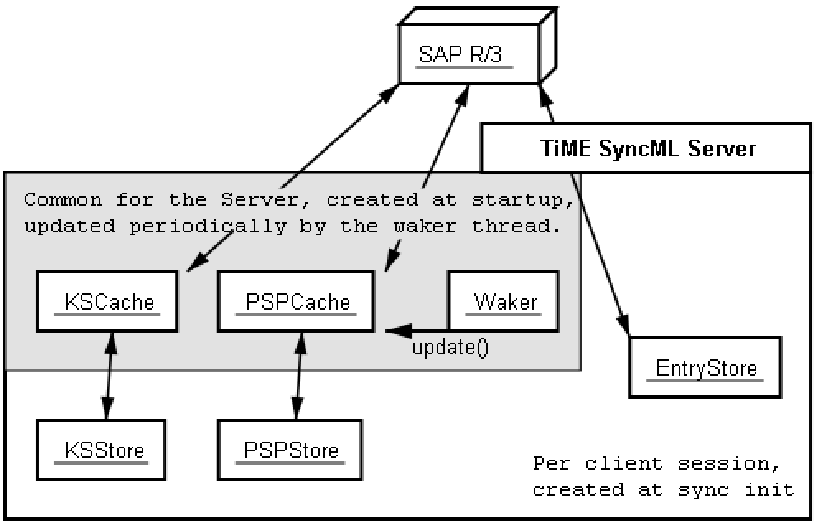

SAP R/3
TiME SyncML Server
Common for the Server, created at startup, updated periodically by the waker thread.
KSCache
PSPCache
Waker
update()
EntryStore
KSStore
PSPStore
Per client session, created at sync init

Figure 18: Cache concept of the server.

instantiation of them, they are implemented as *Singleton*[17]. To learn about new or modified projects and costcenters, a separate thread is started. It sleeps for a configurable amount of time, typically a day. Then it wakes and calls update on the caches and sleeps again. This is repeated as long as the web server is running.

The actual stores get a reference to the caches. The stores only save the last synchronization time for the actual client, so they can ask the cache what has been modified since. Writing of projects and costcenters to SAP is forbidden, as it should never be needed.

No cache for time reports

For time reports, no caches are used. They are client specific, only a small number exists per user and they change often. Upon connection with the client, the `EntryStore` is created and reads all time reports for the client from SAP. A local copy of the data is used nevertheless, in order to find what was modified on SAP since the last synchronization. Modifications received from the client are passed directly to SAP, to be sure nothing gets lost in case of a server crash.

Updating the caches only once every couple of hours sounds a little dangerous. If a new project is defined, a user immediately charges work on it inside SAP and then synchronizes, the new time report is found, but the new project not, as the cache has not been updated since. The Palm application can not work with reports for inexistent projects and would crash.

To avoid this problem, the EntryStore checks for every new report read from SAP if the referenced project or costcenter exists in the cache. If it is not found, cache update is triggered. Because reports are read at sync initialization, the caches are updated before anything is sent to the client. Either the new costcenter / project will be sent, or the connection times out from client side and synchronization has to be restarted. But sending inconsistent data would be worse than risking a timeout.

[17]See [9] for more information about the design pattern "Singleton".

4.3.3 Authentication

For authentication, `SAPAuthentication` is used. This extension of the basic authentication checks the correctness of the credentials by trying to connect to SAP with the specified username and password. Authentication is delegated to SAP by only accepting credentials that result in successfully connecting.

The connection is stored in a special IUser type called `SAPUser`. The entry store requires the IUser to be of type SAPUser, in order to connect with SAP.

For updating the caches, some username and password are needed. They must be in the configuration at startup, as no client specified username and password are available at this moment.

4.3.4 jCO: Java Connector

Using SAP from external programs is usually done calling BAPI's (see Section 4.1). SAP provides a library to their customers called Java Connector (jCO, [19]) for calling BAPI's from within a Java program. The BAPI parameters can be filled in by method calls or with XML strings and results can be retrieved as XML too. `EntryStore` and the caches use the BAPI's listed in Section 4.1.

To connect, lots of parameters are needed. They are passed as a `Properties` object to jCO. Table 7 lists the parameters with old examples from ESPRiT. `SAPAuthentication` uses those parameters to create connections. Everything except username and password is specified at startup.

The user ID for the connection must not necessarily be the ID of the employee reporting time. A user with rights to modify time reports of any employee could be used for this application, but then, user authentication had to be done entirely by the SyncML server.

Parameter name	**Explanation**	**Example value**
client	SAP client	200
user	User ID	1125
passwd	User password	test123
lang	Language for messages	DE
ashost	Host name	/H/195.179.4.138/H/genf
sysnr	System number	00
gwhost	Gateway Host	195.179.4.138
gwserv	Gateway service	sapgw00

Table 7: Connection parameters for jCO.

Chapter 5

Conclusion

5.1 Criticisms of SyncML

The implementation of sync4j revealed smaller problems in the SyncML specification. The politics of the SyncML committee are diverging. Having the reputation of being closed and not very flexible, it announced recently to move towards Open Source.

5.1.1 The SyncML committee

The aim of the committee is to make SyncML an industry wide standard, but their structure is rather closed. Even some of their (smaller) members complained about it in the enquiry held for this diploma thesis. Only one of the 10 main sponsors with direct contact addresses on the syncml.org web site answered to the enquiry email at all.

To have a device tested for SyncML compliance, companies must attend a meeting of SyncML called SyncFest. With the membership fees, SyncML development gets rather expensive. Until recently, the committee began to sell the SyncML Conformance Test Suite used at the SyncFest. The parts synchronization and device management cost each \$ 5000. For promoter members, it is free, normal members get a \$ 1000 discount on the first part they purchase.

As discussed in Section 2.4, the reference implementation of SyncML was not made available for some time. But a press release dated from 8-20-2002 announces that the SyncML committee released the toolkit under Open Source License. [18]

A publicly available reference implementation had been the key for the well known Internet protocols to succeed and become a wide used standard, for example HTTP, SMTP or FTP.

SyncML has it's potential. Perhaps this more "Open Source oriented" attitude will help it to conquer the synchronization world.

5.1.2 Flaws in the specification

The SyncML specification tries to be as clear as possible, using the keywords "MUST", "MUST NOT", "MAY", etc. as defined by the IETF for RFC's. But the specification is sometimes unclear and some operations can be done in different ways.

[18]The press release is at `http://www.syncml.org/press_release.asp?id=36`, the toolkit is available at `http://sourceforge.net/projects/syncml-ctoolkit`.

- REPLACE must be interpreted as ADD if the item does not exist. This is not very intuitive, I would prefer to use PUT if the application does not know if the item already exists. The ADD operation is rather useless if REPLACE adds items too.
- It is not clear if GET and PUT can only be used in device information or if they are used to transmit data also.
- In the status codes for operations, two status exist to tell something is forbidden. 403 "forbidden" and 405 "not allowed". The descriptions are worded different, but it is not explained why two codes exist and when to use which.
- Addressing using URIs is very flexible, but also rather complicated. Because each item inside a command can be addressed with an absolute URI[19], the database in the Sync command could be overwritten. I think it would be sufficient if the item just would have an identifier consisting of `#PCDATA`.

Thus, SyncML is a rather complex protocol, mostly because there are too many places to specify meta information, targets etc. This makes it difficult to implement. The author had to spend more time coding than planned.

Confusion caused by device management requirements

Since version 1.1, the commands Add, Replace, Delete etc. may be placed directly in the SyncBody instead of the Sync element. This changes were made for the device management part of the protocol, but the specification does not restrict it to this part. It would be better to have this restriction, as with the current situation, both versions must be supported. The tag <`Sync`> looses its sense and the code gets more complex.

The same problem is with commands allowed inside Atomic and Sequence. Because of the device management protocol, it is allowed to put Get, Exec and Alert commands into them. But this is not restricted to device management, confusing the structure proposed in 1.0.1 .

5.2 Criticisms of the project

This section leads along the analyzing loop of "achieved results" - "known problems" - "possible extensions". At the end, the tools used for the diploma thesis are listed.

5.2.1 Achieved results

The sync4j prototype is runnable and synchronization between Palm and server working. A fast and easy to use Palm application for time reporting has been developed. The user test revealed lots of possible improvements, of which a majority has been implemented.

The author learned a lot for different areas: SAP, Palm C++ programming, Java programming using existing libraries and holding an online enquiry. Because the work for completing the Palm application and also the difficulties for getting the sync4j prototype running were underestimated, the project duration was longer than planned.

[19]The spec indicates, "The Target and Source specified within the Item element type SHOULD be a relative URI, as relative to the corresponding Target and Source specified in the parent Atomic, Sequence or Sync command". I would prefer a MUST.

The design is not always compatible with the envisioned sync4j design by S. Sullivan. But the writing of the Palm store proved that this design is very attractive for quickly adding a resource to the sync4j system. Using the knowledge on HotSync conduits gathered with the first version, the client had been written within 3 working days.

As no real alternatives to SyncML were found, the comparison between different synchronization standards could not be done.

It would have been interesting to test interoperability with the SyncML toolkit, or even try to use it on the Palm. As it was released only a few days before the end of the diploma work, this was no more possible.

5.2.2 Known problems

Sync4j is not complete. The whole part of device management is not implemented in the prototype. Also, no meta information about maximal record size, maximal record number etc. is processed.

The code is not very stable and error case behavior sometimes confusing. For a really reliable application, restructuring of the code and much more debugging will be needed.

The compatibility with other SyncML applications has not been tested. As the protocol is not always easy to understand, some wrong interpretations of the specification will have found their way into the code.

The Palm program has been thoroughly tested. The only major bug is that the application crashes with an error message if a KS / PSP referenced by a time report does not exist. The other bug is that it is possible to add the same item more than one times in the favorites. But this is less severe, as the program is able to run correctly anyway.

5.2.3 Possible extensions for sync4j

- Add support for the mandatory data types vCard, vCalendar etc .
- Implement MD5 support. Possible source for the MD5 code could be [11].
- Implement CGI scripting. Possible source for CGI parser could be [11].
- If the payload is no XML, it could contain the reserved characters "`< > & ''`". If for example "<" is not matched by ">", the whole message is invalid and can't be parsed.
- Client mapping should be able to save mapping commands for using them on next sync. Handler should be able to send the update at the beginning of a new sync session if it has not been sent at the end of the last session.
- Improve the structure of the handler. Perhaps it would be cleaner to use one instance per data store.
- SyncEngine classes should be integrated into the handler, as the division is arbitrary. The handler should be restructured to be more modular, possibly with interfaces.
- Improve the server. Currently, sessions are never removed from the environment, even if they terminate correctly.[20]

[20]The TiME server however, does remove them, but not through the standard interfaces. He knows the specific implementations.

5.2.4 Development directions for the TiME application

The ESPRiT TiME synchronization should be improved by enabling the server to detect if a costcenter is still bookable (for projects, it is already implemented). Further, only projects and costcenters valid for the current user should be transmitted. The problem is that SAP seems not to provide this information. For some cases, CGI filters could do the job. ESPRiT for example has all names twice, one ending with .CH for Swiss employees and one with .DE for Germans.

Old time reports should be soft deleted for properly restricting what is synchronized and what not.

Extension possibilities for the Palm application

To enable the Palm to synchronize directly with the server, for example via a cellular phone with infrared port, the SyncML toolkit could be integrated into the application.

It is currently not possible to confirm the time report from Palm, the user has to log into the SAP GUI for this task. The confirmation is needed at the end of a booking period for the bookkeepers. Implementation would consist of setting the RELEASE_DATA flag in `BAPI_CATIMESHEETMGR_CHANGE`.

Additional functions for the application are possible. Very important would be reporting of expenses. They are also charged on costcenters and projects. Another task could be annotations to the work done, in case a project needs more detailed proof for how the time was used. Also very nice would be a recorder function to just click when work starts and again when it ends and automatically create the report entry.

5.2.5 Used tools

Java programming was first done using UltraEdit32 on Windows and "make" in cygwin environment, a shell with lots of GNU tools for Windows. For SyncML, Together Control Center 6.0 was used.

Primary developing system for the Palm application was Linux, with XEmacs as editor. For compiling, the PRC toolchain 2.0, including a GCC modified for PalmOS, was used. Resources where designed with pilrcedit, edited in XEmacs and compiled with pilrc version 2.8 beta 7. Debugging was done running m68k-palmos-gdb on Linux, connecting to Pose on Windows.

Earlier documentations where written with Microsoft Word, but the thesis is written in latex, the PDF created with pdflatex. For the diagrams, dia 0.86 was used[21].

The installation program for the Conduit and Palm application is the Conduit Installer 1.6.1 by handX software, distributed under the Lesser Gnu Public License (LGPL).

[21] For Windows users, a Windows port of dia is included on CD.

Bibliography

[1] http://www.syncml.org[22], official SyncML web site. (Specifications and toolkit also on accompanying CD)

[2] http://sync4j.sourceforge.net, project home of sync4j.

[3] Sullivan S., Presentation of sync4j at JavaOne, 2002. http://sync4j.sourceforge.net/presentations/sync4j-JavaOne2002.ppt (Also on accompanying CD, sync4j-JavaOne2002.ppt)

[4] Berger M., Integrated PIM data management with SyncML, Technische Universität München, 2002. http://wwwbrauer.informatik.tu-muenchen.de/~bergerm/syncml/ (Also on accompanying CD, bergerm.pdf)

[5] libsyncml, Open Source SyncML implementation in C++, http://libsyncml.sourceforge.net

[6] kSync, Open Source SyncML implementation in Java, http://ksync.enhydra.org

[7] Bergsten H., JavaServer Pages, O'Reilly, Sebastopol, 2001.

[8] Flanagan D., Java in a Nutshell, O'Reilly, Sebastopol, 1999.

[9] Gamma, Erich (et al.), Design patterns: Elements of reusable object-oriented software, Addison-Wesley, 1995

[10] Java 2 Enterprise Edition, http://java.sun.com/j2ee/

[11] Java utilities, Ostermiller S., http://ostermiller.org/utils/
http://ostermiller.org/utils/MD5.html
http://ostermiller.org/utils/CGIParser.html

[12] http://www.zvon.org/HTMLonly/XSLTutorial/Books/Book1, Zvon XSLT Tutorial, Nic Miloslav
http://www.zvon.org/xxl/XSLTreference/Output/, Zvon XSLT Reference

[13] http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html, Kapitel 17 of XML Bible, Second Edition, Harold, Elliotte R., June 2001

[14] http://www.jdom.org JDom, simplyfied XML wrapper for Java.

[15] http://xml.apache.org/xerces-j/ Xerces XML processor for Java.

[22] All web links where last checked 13.9.2002 if not mentioned otherwise.

[16] Official Palm OS web page, `http://www.palmsource.com/developers/`
Palm: Palm OS Companion.pdf, Palm OS Reference.pdf (On CD: palmos40-docs.zip)
Conduits: JSyncCompanion.pdf, JSyncTutorial.pdf (On CD: cdk402-docs-win-pdf.zip)

[17] Palm Programming, The Developers Guide, Rhodes, Neil und McKeehan, Julie, O'Reilly, Sebastopol 1999

[18] Conduit Installer, `http://www.handx.net` (Also on accompanying CD, ConduitInstaller_v1_6_1.zip)

[19] `http://service.sap.com/connector/` Libraries, reference and tutorial for Java Connector jCO. (Password protected page for SAP customers)

Appendix A

Directory organization on the CD

Contents of readme.txt in CD root directory:

```
/
|
|
+-readme.txt : Informations about the data on the CD
|
|
+-Documents/ : Diploma thesis and additional documents
|  |
|  +-paper.pdf : Main document of the diploma thesis
|  |
|  +-summary.pdf : One page summary of the thesis
|  |
|  +-plakat.pdf : Eight pages short overview of the thesis
|  |
|  +-Extra/
|  |  Additional documents written by the author.
|  |  (Folders for enquiry, Palm program documents, and beta test)
|  |
|  +-Sources/
|  |  | Source of the diploma thesis paper and summary (latex files)
|  |  |
|  |  +-img/
|  |    Image files for the thesis and installer to edit dia diagrams.
|  |
|  +-SyncML/
|     Specifications from syncml.org for versions 1.0.1 and 1.1,
|     along with the 1.0.1 toolkit and the most recent toolkit from
|     sourceforge.
|     Presentation of sync4j by Sean C. Sullivan (MS powerpoint file).
|     Thesis about SyncML by Maximillian Berger.
|
|
|
+-ESPRiT_TiME/ : Installer for Palm application & source.
|  |
|  |
|  +-Build2.0/
|  |  | Compiled Palm program TiME, version 2.0.
|  |  |
|  |  +-DemoVersion/
|  |     A version of the application which does not require to be
|  |     synchronized before running, for the Palm simulator.
|  |
|  +-Evaluation_Environments/
|  |  Documentation of environment evaluation for Palm programming.
|  |  Installation packages for the evaluated environments.
|  |  Test applications for the environments.
|  |
|  +-Installs/
|  |  | Installation packages for editors, compilers and documentation
|  |  | used. (Open Source and public domain programs)
|  |  | => To install the application, use the installer in Setup2.0
|  |  |
|  |  +Pose/
|  |     Palm simulator (OS may not be distributed, must be downloaded
|  |     from a Palm computer or from palmos.com web page after signing
|  |     a non disclosure agreement.)
|  |
|  +-Old/
|  |  Old versions of the time reporting application and a test
|  |  application to crash the Palm with heap overflow.
|  |
|  +-Setup2.0/
|  |  The setup for installing Palm program and conduit.
|  |  Run ConduitInstaller.exe to start installation.
|  |  After the installation, start TiME on Palm once.
|  |  Then start HotSync to load projects and costcenters onto the Palm.
|  |  For synchronization, the tomcat server provided on this CD must be
|  |  running. (Copy it on a writable drive!)
|  |  The results of the synchronization are in the file
|  |  [Tomcat]\webapps\syncml\WEB-INF\stores\Entry.xml
|  |
|  +-Source2.0/
|     | The source code of version 2.0 with makefile for m68k-palmos-gcc
|     | To compile the code, use the compilers from the Installs dir.
|     |
|     +resources/
|       Resource files for pilrc.
|
|
+-SyncML/
   |
   |
   +-Apache_Tomcat_4.0/
   |  A JavaServer configured to run the sync4j servlet. Before running
   |  it, copy the folder to a writeable drive. On the CD, tomcat can
   |  not create files and will not work correctly!
   |  It can be started and stopped with help of the batch files in bin\
   |  The application data is in [Tomcat]\webapps\syncml\WEB-INF\stores\
   |
   +-Installs/
   |  | Installation packages for developping conduits.
   |  |
   |  +-Library_Distributions/
   |      Original library distributions used by sync4j.
   |      (The necessary jar files are already in the right directories)
   |
   +-Old_Conduit/
   |  Old versions of the conduit.
   |  (Without SyncML and only poor conflict resolution.)
   |
   +-sync4j/
      Source code, compiled java classes and documentation.
      For running the tests, use the batch files in src\tests\
      Copy the whole sync4j directory to a writable disk as the tests
      need to write files.
      prepareStore.bat can be used to reset the stores.
      For sucessfully running testClient.bat, Tomcat must be running.
```

Appendix B

Guide to the additional documents

Additional documents written by the author are found on CD in subfolders of folder `Documents/Extra`. Below is a description of those files. Some of the earlier documents are written in German.

B.1 Enquiry about SyncML

The file `SyncML_Enquiry.pdf` contains the evaluation of the enquiry on 25 pages. The additional document `Enquiry_Questions.pdf` lists the exact questions.

The enquiry is also available through the Internet under the address `http://diuf.unifr.ch/telecom/projects/syncml/`.

B.2 ESPRiT TiME: Time reporting on Palm

The file TiME_Palm.pdf contains information about the Palm application for time reporting and some comments on Palm programming in general.

The installation guide and user manual is in the file Anleitung.pdf (also in German).

B.3 Beta Test

To test the Palm application, a user test was conducted with three employees of ESPRiT. The evaluation of their feedback on version 1.0 of the Palm application and synchronization without SyncML are in the folder `Documents/Extra/BetaTest/`

B.4 Development environments on Palm

To choose the right environment for programming the Palm handheld, an evaluation of different developing environments was done. Tests were made with the open source C++ compiler GCC for Palm, IBM's Java environment Visual Age Micro Edition (VAME), a small virtual machine implementing a subset of Java called Waba, and the high level design tool satellite forms. GCC was finally selected for programming the application.

The file `palmos_developtools_evaluation.pdf` holds the results of the evaluation (in German). Installs for the evaluated environments along with the evaluation programs are on CD too. All files are in the directory `ESPRiT_TiME/Evaluation_Environments/` .