Integrated PIM data management with SyncML

Maximilian Berger
Technische Universität München

Integrated PIM data management with SyncML

by Maximilian Berger

Published 2002

Table of Contents

List of Tables

List of Figures

List of Examples

I. Introduction

Chapter 1. Motivation

> *When an actor comes to me and wants to discuss his character, I say, "It's in the script." If he says, "But what's my motivation?", I say, "Your salary."*
>
> *Alfred Hitchcock*

For a long time, communication devices were dumb. When I wanted to call a friend, I would take out my paper organizer and look up the phone number. Then I would punch it into the phone, getting just the desired results. And when my friend suggested a date, I would take a pen and note it in the same organizer.

But the times have changed rapidly. Today, I do not have a paper organizer anymore, I have a Palm Pilot™. I do not type in numbers into my phone anymore, it has an address book. So does my phone at work, and of course my mobile phone. And this is where the problems begin:

Whenever I get a new contact, I have to enter his phone number three times. After all, I want to be able to reach him from home and work, and of course from my mobile. Ok, you might say, most of my work contacts I would not call from home and the other way round, but this is just evading the problem instead of solving it. A real solution would be to let those Personal Information Manager (*PIM*) devices talk to each other.

And such software really exists. I can synchronize my Palm address book with MS Outlook, or with Gnomecard. I have even found software for synchronizing my mobile phone with MS Outlook or with the Palm address book. So the current solution is to adopt every single *PIM* device to work with every other single *PIM* device.

Obviously that this is not a very good solution. One good solution would be to standardize the synchronization protocol and to standardize the data being synchronized. The goal of this thesis is to analyze one of theses approaches and to try to implement a fully working version.

This solution comes mostly from the vendors of mobile phones and PIMs. They were tired of supporting every single product. So most of the larger ones (Ericsson, IBM, Lotus, Matsushita, Motorola, Nokia, Openwave, Starfish Software and Symbian are mentioned on the SyncML web site http://www.syncml.org) started and sponsored the SyncML project.

According to its web site, "SyncML is the leading open industry standard for universal synchronization of remote data and personal information across multiple net-

works, platforms and devices." In reality, SyncML is the only industry standard for synchronization of *PIM* data.

SyncML defines a basic client - server interface for exchanging personal data. Basically, any devices can act as a server or a client. The requirements for a server are much higher, though. And to avoid complete data confusion, this approach leads to one central server.

The SyncML standard describes only the way data is exchanged, but not the format of the data itself. Is does, however suggest some formats, such as vCalender and vCard and even requires them for certain applications. SyncML itself is described in an *XML* syntax.

Since SyncML is supposed to be an "open" standard, the suggestion may occur, that there are already some open tools for it. There is a reference implementation and some java projects that will be discussed later.

And this is where the brave new world already ends. There is currently no working, fully implemented C or C++ library. And when it comes to common open source projects such as Gnomecard or Kab, none of them implement SyncML, not even as client. And when it comes to a server back-end with a real scalable database, there is yet an open-source implementation to be made.

To build a prototype this project has therefore to design and to create a SyncML application. In particular, the goals are to:

- create a full SyncML featured C++ library, that implements all requirements for SyncML 1.01
- adapt an existing open-source *PIM* client to use this library for acting as a SyncML client.
- write a full featured SyncML capable server which is able to store calender data.

SyncML is an event based protocol. The library must have a way of calling back into the main program and of selecting explicit data record without having to know their content. The basic idea here comes from another *XML* programming implementation: *SAX*. In SAX, the parser itself is an object. Whenever an event gets fired, the appropriate method gets called. A programmer has to extend the basic parser class and overwrite the methods that need to be customized.

The same idea will be used in the SyncML library. Ideally, all configurable parameters and functions have reasonable defaults, so that the library can be effectively used out-of-the-box.

With this library, the client part is actually pretty easy: Both, Gnomecal and KOrganizer, store their data in the standard vCalender format. So an application, which

reads a vCalender file and synchronizes it, will meet the requirements, and would be even more extensible.

Unfortunately, the same is not completely true for the address book data. Gnomecard uses the standard vCard format, and will therefore be supported. Kab on the other hand uses a proprietary format, which would need more customizing.

The server application has other issues to solve. To be able to sync data, it must keep complete logs which data has changed on which client. This data is needed to determine which entries must be synchronized. Also, the server must remember which clients it has previously synced to. There is a lot of data to store about the client: type of client, its capabilities, local *UID*s, etc. Whenever the same data has changed on more than one client, the server must have a way of finding out which new data to keep and where to merge.

Keeping all this information in the server leads to another important issue: The data structures must be capable of holding all this information. The question is, how detailed the change logs have to be, which information is kept about the clients, etc.

To conclude, the necessary steps are:

- Design a concept for a complete client - server architecture for distributed management of PIM data
- Design data structures, especially on the server for keeping the information
- Define, how the clients and server actually connect
- Implement the SyncML protocol
- Implement a client and server prototype

Chapter 2. Infrastructural overview

There is no reason for any individual to have a computer in his home.

Ken Olson, President, Digital Equipment, 1977

Let us take a step back and look at the problem again: Electronic PIMs are currently not very usefull, because each has its own database and it is very hard if not impossible to keep all of them in sync. The goal is to find some way to keep the data consitent on as many devices as possible.

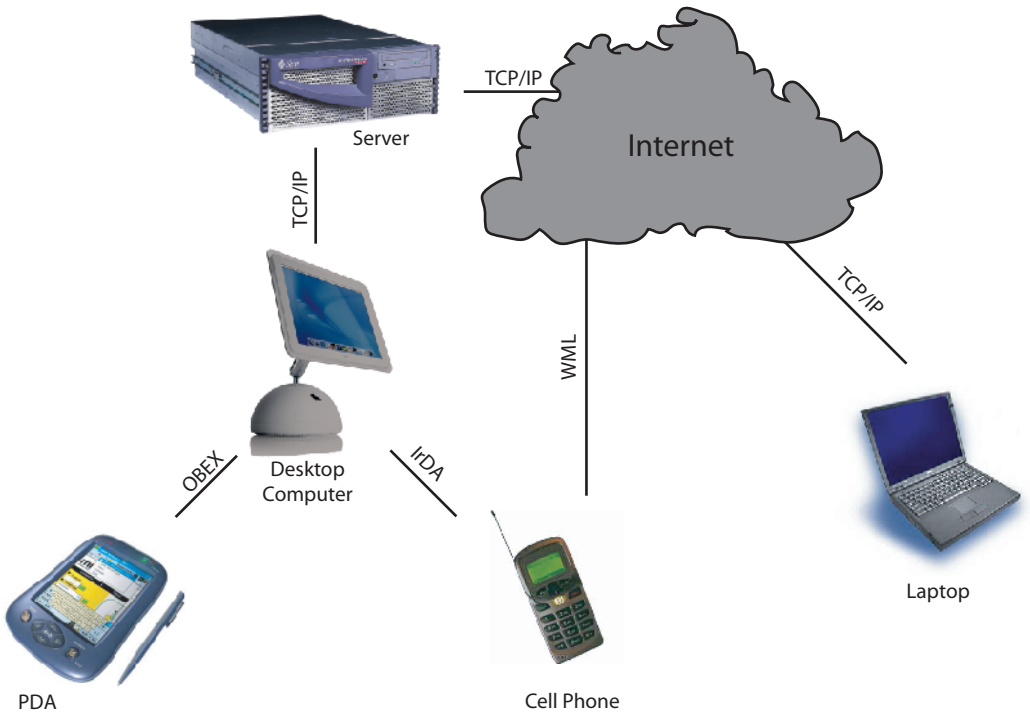
To design a solution for all devices, we have to find out which devices exist and what their capabilities are. Table 2-1 gives an overview of some PIM databases:

Table 2-1. Overview of some PIM databases

Device	Examples	Type of Data	Capabilities
Cell phone	Nokia Communicator	Phone Numbers	very small in memory
PDA	Palm Pilot, Compaq iPAQ	Addresses, Schedule, Notes	medium in memory and computing power
Desktop Organizer	MS Outlook, Gnomecard		large memory, fast computing
Relational Database	MySQL	Any	very large memory, not only for PIM data

The next step is evaluation how those devices connect. As said earlier, most of them already have some kind of network connection. Figure 2-1 gives us a a little view on how those devices typically connect.

Figure 2-1. Different clients and how they connect



As you can see, the whole situation is pretty complicated. Different architectures have to be considered, and different transports. But the most difficult to consider are the different types of clients: server class, workstation class and thin clients.

As suggested earlier, we will reduce the problem to a much simpler client - server problem. When looking at Figure 2-1 the only problem left is the desktop computer. The application here would have to act as a client to the central database sever and as a server for the mobile devices.

Even simpler, we could write a proxy application for the desktop computer. The thin clients would communicate with an adapter program on the desktop computer. Then the adapter program would handle all communications with the actual server.

Having solved the “desktop computer” issue enables us to build an approach on the base of TCP/IP. No other protocols need to be supported directly. It also enables us to reduce this problem to what we wanted: a simple client - server problem. Now all that is left is implementing a server, a client, and also a common protocol...

Chapter 3. Evaluating complete solutions for data synchronization

Well done is better than well said.

Benjamin Franklin

Before deciding which method to use for data synchronization, several solutions have to be evaluated. Based on this we can decide which one best satisfies our demands. The first thing to look for is a complete working solution that already provides synchronization with various data sources.

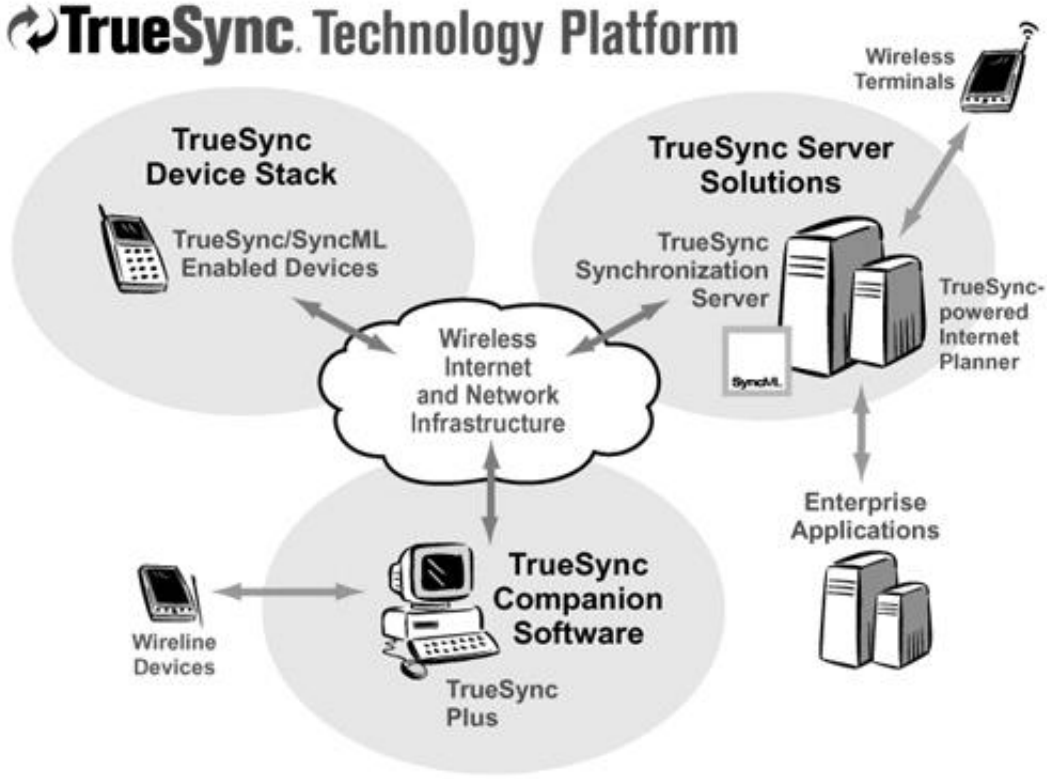
Starfish TrueSync

When it comes to the terms of synchronization software, the first one to mention is Starfish‘s TrueSync platform. Starfish is a company that was founded in 1994 in California. It started out with a simple desktop organizer software but soon went over to address data synchronization problems. Its main product is the TrueSync platform. This platform can be adopted to almost every PIM client, and many PDAs. In 1998 Starfish even started creating an own server software.

Starfish was also a founding member of the SyncML initiative. Since all their products are based on the same TrueSync platform, they could easily be adopted to the new SyncML platform.

Many vendors of small devices now licensed this software for use in their own devices. The Starfish TrueSync platform is probably the most widely used in current handheld devices.

Figure 3-1. How starfish sees its own products.



Starfish TrueSync addresses all mayor issues in synchronizing data across PIM devices and desktop-based solutions. It works with the most common used organizer programs such as Microsoft Outlook and Lotus Notes. It also works with all PalmOS based PDAs, all Motorola cell phones, and many others.

There are two ways TrueSync supports a product: The manufacturer can license the TrueSync software directly. The TrueSync platform is then adapted for this particular device. The second way is by adding an apdapter which translates the device specific commands to TrueSyncs own protocol.

But TrueSync does not only support client devices. There are desktop based solutions called TrueSync Plus, TrueSync Express, and TrueSync SDK. TrueSync Plus is a Personal Information Manager software. It supports the standard features for a desktop PIM software, such as calender and address book. TrueSync Express is just the adaptor between the TrueSync Protocol and existing PIM Software, such as Lotus Notes or MS Outlook. TrueSync SDK is a sofware development kit that can be used to adapt an own software to the TrueSync Protocol.

To complete its software spectrum, StarFish also offers server solutions and even an Internet Planner to store and access the data.

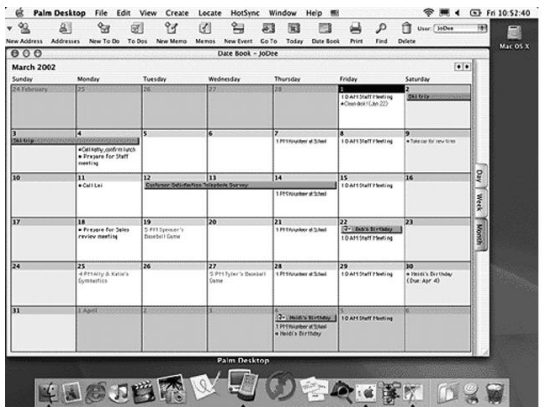
For communication between the different software parts StarFish uses either a proprietary TrueSync protocol or the newer SyncML protocol. This enables the software to interoperate with all other SyncML conform applications and devices.

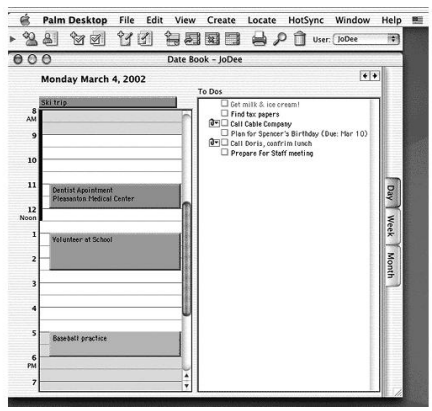
So why not just use TrueSync? It has everything that one could possibly want. The answer is a question of money: Starfish currently does “not sell directly to end users or in small volumes”. This makes it difficult if not impossible for an end user to actually use this product.

Palm Desktop

Palm Desktop is another complete software suite that deserves to be mentioned. Its main purpose is to synchronize any desktop PIM software with a Palm Pilot.

Figure 3-2. Palm Desktop showing monthly and daily calendar view.





The Palm Pilot was one of the first PIM devices available, long before other devices such as mobile phones learned the capabilities for holding a sufficient amount of personal information.

This new device also had the capabilities to load new programs and extend its functionality. Therefore a communication with a PC and some kind of transmission program had to be build.

And, most important, people would not enter all their data in the Palm Pilot itself, but rather wanted to use a full size keyboard and their desktop PIM programs.

So, Palm made a software that could do all that: load programs, backup data, and synchronize it with other databases. In case you do not have another database, the Palm Desktop program itself is a complete PIM solution: It has a calender, address book, and to-do list build in.

All other software can be synchronized with a Palm Pilot via plug-ins. Such a plug-in is called *Conduit*. Conduits exist for almost all PC organizers, such as Outlook, Gnome-pim, and KOrganizer.

Palm Desktop has two major shortcomings: The first one is very obvious: It only works with devices that run Palm OS. This leaves out the second mayor PIM device platform Windows CE and all current cell phones.

The second shortcoming is that Palm Desktop is build for synchronization of the Palm Pilot with only one database. You could synchronize one Palm Pilot with two computers, for example at work and at home, but this gives very strange results when it comes to things such as deleting database entries.

Palm Desktop provides an extensible platform. It is a very good tool to access the data on a Palm Pilot. The conduit concept makes it easy to extend, and this is what I would use in future versions to access data on a Pilot.

Chapter 4. Protocols and data formats

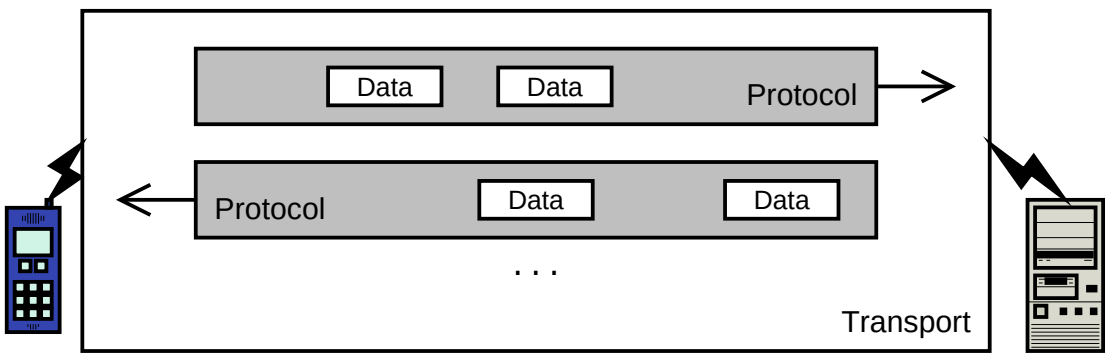
It´s a well known fact that computing devices such as the abacus were invented thousands of years ago. But it's not well known that the first use of a common computer protocol occured in the Old Testament. This, of course, was when Moses aborted the Egyptians‘ process with a control-sea...

Tom Galloway

Protocol, transport, data

To classify the following specifications, we have to clarify first what is meant by the terms transport, protocol, and data.

Figure 4-1. Transport, Protocol, Data



A transport does the actual connection. Its purpose is to establish a connection between two machines so that they can exchange data. The most commonly used is TCP, but it could also be HTTP, OBEX, WML or IRDA.

The data is the actual data to be synchronized. It is usually marked up with very little extra information to be easily parsable. The data formats explained here are vCard, vCalender and iCalender.

A protocol describes how the data can be exchanged during a session. Some protocols demand special data structures (e.g. LDAP), others provide support for different ones (e.g. SyncML).

Data types

Another thing that has to be clarified is what types of data has to be managed and what the requiremens for each data type are. The most common PIM data types are calendar, address book, notes and to-do list.

Calendar

: A calendar item specifies something that happens at a certain date, time or range of either. It can be a one time event or repeat itself. PIM applications ususally support some kind of alarm prior to the scheduled date.

Address book

: Address book entries contain all kinds of whitepage information: First name, last name, title, birthdate, private address, business address, picture, phone numbers, email addresses, and many more.

Notes

: A note is a small text or a graphic, optionally with a title.

To-do

: To-do items are things that have to be done once. To-do items can be marked as complete once their done. Some To-do items might have a due date.

One feature that I have not seen implemented in any PIM application is an automatically apprearing to-do item prior to schduled events. For example: I would like the to-do item “buy present” to appear no earlier than two weeks before a birthday. Or I would like the item “buy train ticket for next month” to appear no earlier than the 25th of each month.

For resons of simplification, we will pick calendar or address book data when explaining some things in detail. The same information ususally also applies to notes and to-do entries.

The Versit format

One of the most important decisions is how the PIM data is encoded whithin a data file or database. The format must be extensible and support a rich set of features, but

must still be easy to handle.

We take a suggestion from the SyncML specification: Any SyncML server that supports a contact database must support the vCard 2.1 (see vCard21) and the vCard 3.0 (see RFC 2425 and RFC 2426) format.

The “v” in vCard stands for the versit consortium. This consortium has also published other standards, such as vCalender and vTodo. Although the versit consortium itself does not exist anymore, those standards are still the mostly used and most widely accepted. Many open source applications use vCard internally as data format and many E-mail programs have the capability to attach business cards in vCard format.

vCard structure

The vCard format is uses the standard 7-bit ASCII character set for its contents. A vCard is a line oriented text file. Each line consists of a property, a colon, and a value. Multiple vCards may exist in one file: the two special lines `BEGIN:VCARD` and `END:VCARD` define the begin and end of a vCard entry. The vCard format itself is very easy human readable, so let us just take a look at Example 4-1:

Example 4-1. Minimal version of my personal vCard

```
BEGIN:VCARD
VERSION:2.1
N:Berger;Max
EMAIL;INTERNET:max.berger@xslt.de
END:VCARD
```

8-bit encoding

When it comes to 8-bit enconding, the Versit format shows its origin in the US: 8-bit encoding itself is simple, character set selection is not. For encoding of 8-Bit data the vCard standard defines encode parameters for “quoted-printable” and “base64”. This allows vCards to contain other data such as photographs (`PHOTO`), company logos (`LOGO`), public cryptographic keys (`KEY;PGP`) and others.

The character set selection has to happen on a higher level, outside the actual vCard data stream. This makes 8-bit charachters such as German umlauts dependent on the processing system. vCard 2.1 defined a way to specify the character set of single entries, but this is dropped in the newer 3.0 version.

Selected vCard properties

The vCard specification defines many properties. Most of them are self explanatory and not really relevant for the sync process. Some fields have special functions and need to be explained:

VERSION

Defines the vCard version. Can be either 2.1 or 3.0. Depending on the version the processing has to be a little different. This is explained in the changes section.

FN

This field contains the formatted name for a person. Although it is not handled specially in any way, this is the field we want to use when referring to a card in display outputs, like a debug log.

N

The N property contains the name parts for a person, separated by semicolons. They are: family name, given name, additional names, name prefix and name suffix. For comparison, theses five fields are considered like five separate properties.

UID

The Unique Identifier for this card. Although it is supposed to be unique, it might differ from client to client. So we have to know about it to change for every client.

REV

The REV property contains the revision on an element or, more commonly speaking, the last changed date. This is used for dynamically building transaction logs.

For a complete list of fields see vCard21 and RFC 2426.

Changes in vCard 3.0

Although it lacks some very nice additions, the vCard 2.1 format is still the most widely used standard. The newer 3.0 version, however, has some new features:

In vCard 2.1 parameters are just added to the properties with a semicolon. In vCard 3.0 those parameters are described with the type keyword.

The vCard 3.0 format offers some new fields and new types. Those have to be removed when syncing with 2.1 clients.

And last, but not least, the vCard 3.0 standard defines quoting of 8-bit characters a little different than 2.1 did. It no longer supports the “quoted-printable” format.

Here is my vCard from Example 4-1 again, this time in 3.0 format:

Example 4-2. Minimal version of my personal vCard, version 3.0

```
BEGIN:VCARD VERSION:3.0
N:Berger;Max
FN:Max Berger
EMAIL;TYPE=INTERNET:max.berger@xslt.de
END:VCARD
```

Summary

The versit format is a widely accepted standard. Most clients use it and there is no reason not to do so in this project. It is extensible enough to support many features, yet it is simple enough to be easily debuggable.

iCalender and iTIP

The iCalender protocol is specified in RFC 2445. It is basically the next version of the vCalender format. The iCalender format follows the general rules for the Versit format. It specifies event, to-do, journal, free/busy, and time zone data. Events and to-do items may also have alarm data.

VEVENT

: An event is anything that starts at a certain time and has a specific duration. This includes things such as meetings, lectures, seminars, birthday parties, your favorite TV show and so on.

VTODO

: A to-do item is something that has to be done, optional with a due date. For example: Sign up for tests, buy birthday present, clean up room.

VJOURNAL

: A journal entry stores text or other data for a specified date and time, usually in the past

VFREEBUSY

: Free and Busy time schedules are needed for coordinating meetings with different people. This information is usually made available to others.

VTIMEZONE

: Instead of using the system time zone data iCalender defines its own format to specify timezones.

iCalender is just a way of storing the data. Synchronizing it is specified in RFC 2446. This protocol is called *iTIP* (iCalendar Transport-Independent Interoperability Protocol). So iTIP is what is actually interesting.

Unfortunately, iTIP does not provide the requested features. iTIP is a protocol for synchronizing events, such as meetings between different people, each with their own address books, but not different address book for one person.

It provides mostly features for scheduling events. A person can publish her free and busy time to a group or publically. Anyone can request a meeting, and iTIP offers support for accepting, declining or making a counter proposal.

Last, but not least, the iCalendar Message-Based Interoperability Protocol (iMIP, specified in RFC 2447) defines how iTIP messages can be embedded into E-mails for automatic processing by combined mail and scheduling programs such as Outlook and Evolution.

The iCalendar / iTIP / iMIP solution provides good management for personal and corporate scheduling. It it fully implemented in Evolution, and partially in Outlook. Unfortunately, it does not solve the problem of synchronizing personal schedules across multiple calendar programs.

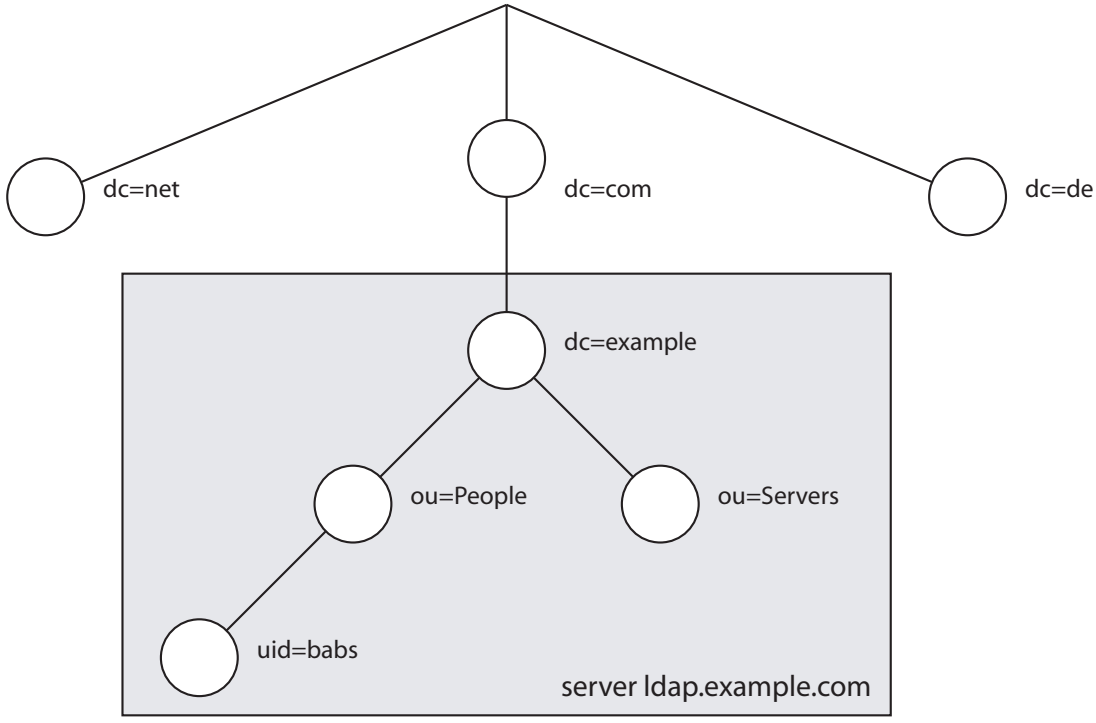
Lightweight Directory Access Protocol (LDAP)

The name LDAP is short for Lightweight Directory Access Protocol. Its current version is 3. It is specified in RFC 2251, and RFC 2252, with additional information in RFC 2253, RFC 2254, RFC 2255, and RFC 2256.

There is a free LDAP implementation of a server and client library. It is available at http://www.openldap.org. It implements the currently used versions 2 and 3 of the LDAP protocol.

LDAP is a database access protocol optimized for reading. It organizes data in a tree structure. The tree structure is adopted from the well known DNS schema. This enables us to find and uniquely identify data.

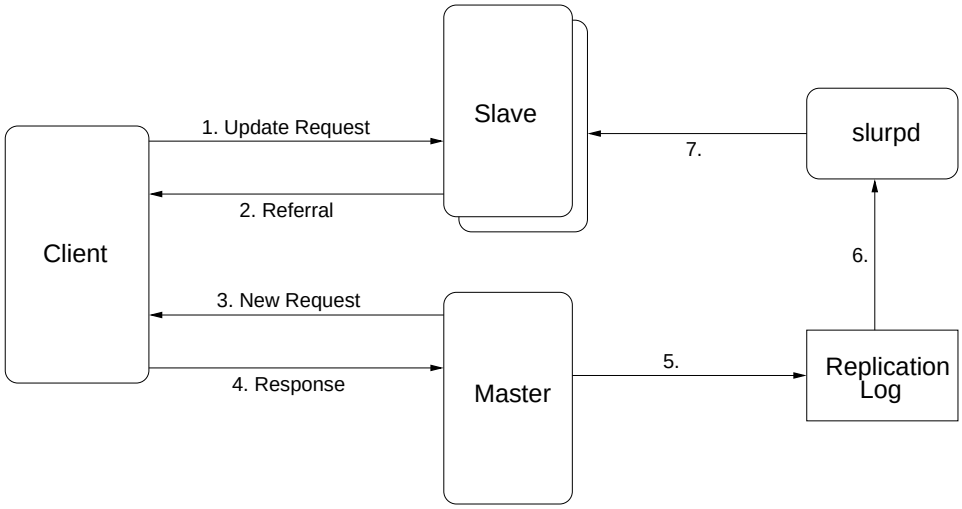
Figure 4-2. An example LDAP tree



Each LDAP server can forward requests to other LDAP servers it knows about. This makes LDAP very easily distributable around the world. Results or whole trees can be cached and replicated to allow disconnected operations.

Writing to LDAP is far more difficult. Each LDAP node can only be changed on its authoritative server. There is no merge protocol. The conflict problem is solved by avoiding it. Whenever a client tries to write into the LDAP tree on a replicated server, it gets back a referral request to the authoritative server.

Figure 4-3. Writing to a replicated LDAP node



Each LDAP node implements one ore more schemas. A schema contains a list of attributes and what they mean. Standards exist for some of the more common used schemas.

When enumerating nodes, these schemas can be taken into account. If, for example, we want to enumerate all address book entries, we would look for nodes implementing the “inetOrgPerson” schema.

Data organized in LDAP is not limited to information about people. LDAP is actually used for administration of large computer clusters. In those, LDAP is used to store computer dependant configuration such as IP addresses, network MAC addresses, and user login information.

Microsoft‘s “Active Directory” is basically just the addition of LDAP to their file sharing protocol. This enables Microsoft to include all these nice distribution features.

LDAP is a very good protocol for data that can not be changed by the end user, like public address books. It provides other nice features such as support for account management. It is a very good source of additional data. But it is not suited for individual, personal information.

SyncML

Since there is no other protocol that specifically addresses the problem of having multiple personal information managers, the SyncML initiative was founded. It is not surprising that the two vendors of the solutions mentioned before, Palm and Starfish are both founding members of this initiative.

The purpose of this initiative was to create a standard that addresses the synchronization of personal information for one person on multiple devices. The main idea here was the problem of a cell phone and its address books: A phone usually has very small keys and people would much rather like to use a full sized keyboard to type in phone numbers, but want to have them available on the mobile device.

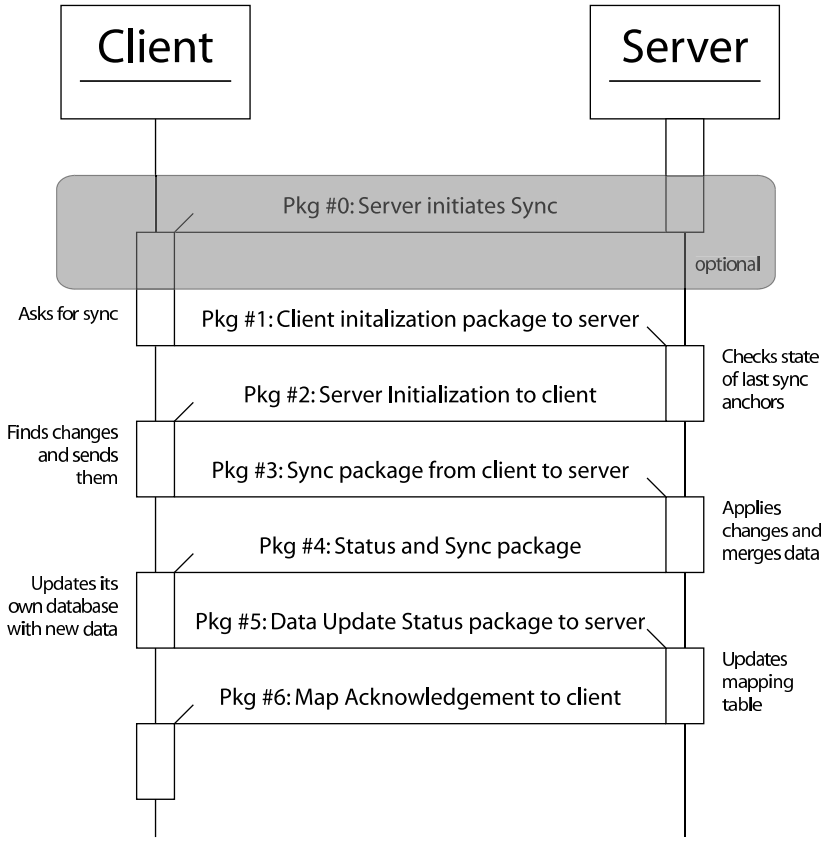
Unlike many other standards, the SyncML protocol by itself is not a complete solution. It needs a transport protocol and a data protocol. The SyncML specification defines encapsulation over HTTP, E-mail, and OBEX, but others are also possible. It even supports different data protocols to be synchronized, such as vCard, vCalender, and iCalender.

On the first impression this may look like a shortcoming - but it is not. It makes the SyncML protocol very extensible. SyncML can be used to synchronize almost every data format.

The SyncML protocol itself is specified as an XML and WBXML application. The XML representation ensures that the protocol is human readable. It also takes care of all 8-bit encoding issues, since these are already specified by the XML specification. The WBXML representation make the protocol small for wireless links, and is thought for mobile devices.

A typical SyncML session consists of 6 data packages that are exchanged between the server and client. Figure 4-4 shows an overview:

Figure 4-4. SyncML session time line



Usually a SyncML session is initiated by the client. It might, however, be server initialized. This adds an extra optional first package.

After the transport has established the session, the SyncML protocol takes over, and does its own handshake. This usually includes an exchange of credentials.

Now both machines have to agree with which type of synchronization to continue. The client requests a type and the server confirms this or suggests an other type.

One reason for suggesting a different type of sync is possible inconsistency: During initialization both machines also exchange their last sync anchors. If they differ the server initiates a slow sync as described in the Section called *Slow sync* in Chapter 6.

Then the client sends its modified data to the server. The server processes this data, merges it with its own, resolves any possible conflicts and sends back its modified data. The client updates its database accordingly.

The client might have assigned new UIDs to its data. Therefore it sends back a mapping table to be stored by the server. At last, the server acknowledges the mappings and the session is terminated.

The SyncML protocol does not specify how conflicts are resolved. But it does specify

many messages that can be used in conflict resolution. One example is the slow sync, others are merging, overwriting on client or server, or duplicating. SyncML also addresses the problem of differing UIDs on different machines.

The promises from the SyncML protocol are many. The cell phone companies are pushing it, and its targeted as an industry standard. The only thing missing now are actual working implementations.

Chapter 5. Existing SyncML implementations

The nice thing about egotists is that they don't talk about other people.

Lucille S. Harper

When said there are no working implementations, this is not fully true. There are two library frameworks that implement the SyncML protocol to some extent. Let us take a look at them:

sync4j

Sync4j is an approach to create a free implementation of the SyncML protocol in Java. It can be found at http://sync4j.sourceforge.net. It is still in an alpha / planning state, so the things mentioned here might already be incorrect or out of date.

Sync4j has a layered architecture. The layers are: core layer, transport layer, framework layer, and application layer

The core layer is responsible for the actual SyncML handling. It takes care of XML parsing and conversion of the SyncML markup to an internal object representation. It can also reverse this and convert this internal object representation to SyncML text. During this process it makes sure the SyncML protocol syntax and semantics are correct. It also defines a standard set of exceptions.

The transport layer defines standard transport interfaces. Transports can be added by implementing these interfaces. The standards transports, HTTP, OBEX, and WSP, will be implemented in the future.

The framework layer contains two frameworks for building SyncML applications: One for servers and one for clients.

The application layer implements both frameworks. This gives example applications that use sync4j for the actual synchronization.

Although the sync4j project has not actually published any source, it is still under active development.

The sync4j concept seems well thought-through. Although it is in a very alpha stage the development plan is clearly laid out. I hope this toolkit will be available soon for everyone to develop cross-platform SyncML applications with Java.

This might actually happen very soon: On April 8th two students from the University of Fribourg forked from the original project and try to work on a complete solution as a diploma thesis and a semester project. I am looking forward to test my software with theirs!

Unfortunately the Java environment is what keeps me from using sync4j. Java has been known to be very slow and consume much memory. This is a shortcoming on thin clients such as PDAs and servers with heavy load.

SyncML Reference Toolkit (RTK)

To establish a standard and to proof that it is actually implementable the creator usually develops a reference implementation. For ISO standards this is even mandatory.

The same thing happened with the SyncML specification. A reference toolkit (RTK) was published with a very unrestrictive license on the web site.

After the standard established itself, however the policy for the toolkit changed. First, the newer toolkit (now called SCTS) was only available to attendees of a so-called syncfest. Then, they decided to take the toolkit totally off the web site and make it available with “promoter membership” only. The only problem with that is that this promoter membership currently costs $20.000 per year. For this reason, the version described here is the last freely available one.

The RTK is written in pure C. It takes care of the parsing of the XML commands and the creation of SyncML messages. This is equivalent to the core layer of sync4j. It also implements basic transports.

Using the RTK requires an in-depth knowledge of the SyncML specifications. Most commands are simply mapped to C functions.

The RTK would have been a good base for the start of own projects. Its major shortcomings are the use of plain C and the need of deeper knowledge. But since the current versions are not free anymore, this opportunity ceased to exist.

Hardware devices

During the course of the last two years, several hardware devices were developed that implement the SyncML specification. A complete list of officially compliant implementations is available at http://www.syncml.org/interop/interop-compliant.html. Most of the devices are mobile phones. Some companies chose to test each device, others just tested their protocol stack for conformance. Since I do not have acces to a hardware devices that supports SyncML I cannot go into more detail.

Currently, only the top of the line phones support SyncML. But this situation will hopefully get better, when the SyncML protocol stack will become a standard part of any mobile phone.

II. Synchronization concepts

Chapter 6. Synchronization basics

Basic research is what I am doing when I don't know what I am doing.

Wernher von Braun

Before we can start creating synchronization applications, we have to take a look at certain synchronization concepts first. And even before that we need to find out what is meant by synchronziation.

What is synchronization?

We define two databases as synchronized whenever their contents are equivalent. Whenever their contents are not equivalent, the databases are unsynchronized or out of sync. It is important to note that equivalent does not necessary mean exactly equal.

Having said that, how do we get two databases to be synchronized and how do they get out of sync?

The easiest way of synchronizing two databases is by replication. With replication the master database is simply copied over the content of the client database. After that process, the former client data is lost, but both databases have the same content.

Databases get out of sync, when at least one database operation is applied to one database and not the other.

Database operations

Ususally database operations fall into one of the three following groups:

Add

A new entry is created.

Modify

An entry is modified. Some data might have changed or some details might have been added.

Delete

An entry is deleted. It no longer exists in a database.

To make it even simpler, the “add” operation is just a special case of the “modify” operation: It is the modification of a non-existing item.

Soft deletion and hard deletion

There also has to be a special handling of the delete operation. Not every client has enough space for every database item. Sometimes we want to remove an item from just one client, but not from the others. This operation is called a soft delete (deletion on one device) as opposed to a hard delete (deletion on all devices).

One way would be for the client to keep track of the phased out items. But this would not solve the problem: The client would still have to know about all records it tried not to know about. Therefore a soft delete has to be handled internally on the server: The server keeps the record that this entry is invisible to a particular client.

Disconnected operation

If all the databases would stay connected, a database operation could be passed on to the other databases. The same modification would be done, and the databases would be synchronized again. Many databases rely on this system. Whenever they get disconnected, they do not allow write access. Most information found in literature describes databases that are connected most of the time.

PIM devices on the other hand are disconnected most of the time. We need to find a way to keep track of the database operations and synchronize them whenever a connection exists.

Unique identifiers

To keep track of an item and its database operations, the item will have to be identified first. A scheduling event, for example, could move to a different time and get a different description. This makes synchronization an almost impossible task.

The solution to this problem is very simple: Have at least one field that never changes. This field would uniquely identify an entry and is therefore called Unique IDentifier (*UID*).

A UID is assigned to an item upon its creation. The UID must not be used again until this particular item has been deleted from all databases.

Unfortunately, different clients might have assigned the same UID to different entries or an entry might have been assigned different UIDs by different clients. Thus we must distinguish between local UIDs (*LUID*) and global UIDs (*GUID*).

A local UID is valid on one particular client. This client knows only about its own local UIDs and uses them for synchronization.

A global UID is basically a local UID for a server. The difference here is that the server tries to assign this own UID to as many clients as possible. Since this is not always possible, the server has to keep a translation map between its own GUIDs and the clients LUIDs.

Transaction logs

Now that we know how to identify particular items we can keep track of them with a transaction log. A transaction log is a history of events that happened since the last sync. This includes the database operations mentioned before: adding, deleting or modifying entries. For a good synchronization, a transaction log has to be kept on every client. The server, on the other side, must keep enough transaction information to know about all changes that happened since the last connect of any client.

Keeping a full transaction log would require much space. Fortunately, this transaction log information can easily be recreated if we have the date of the last change for an entry. We can find the modified entries by comparing their modification date with the date of the last sync. Now, special care is only needed for addition and deletion.

Adding an item is very easy: Since adding is the same as modification of an non-existing item, it can be handled like modification.

Deletion is not as easy. However, there are two simple ways to handle deletion: The first one is to keep empty records, which contain nothing but the UID and the last changed date. When such a record is synchrinized, it is treated as a deletion notice. The second way requires keeping extra information: At every sync we keep a list of items that were synchronized. An item that is missing during the next sync was deleted.

Both ways do not provide a way to distinguish between soft and hard deletion. This is no problem for a server, since it needs only hard deletion. For a client, however, this issue remains unsolved.

Regular sync

Having explained how to identify the items and how to find out what to sync, the only thing left is to explain how an actual data sync process takes place:

- A client connects to a server.
- The client sends all its changed data to the server for processing.

- The server applies the changes and sends back all other changes since the last sync, with new UIDs for new entries.
- The client applies the changes and sends back a mapping table for those UIDs it could take accept for some reason.

Slow sync

A regular sync is only possible when both databases where synchronized before and when both have their transaction logs or are able to recreate them. If this is not possible they have to initiate a so called “slow sync”.

During a slow sync the client sends its complete database to the server. The server compares the entries, merges them as necessary and sends back a complete new database to the client. The comparison itself can either be done automatically or with user intervention. Since this particular server implementation should not require any user intervention, some ways for automatic comparison have to be found.

One-way sync

Another special case of synchronization is a one way sync. During a one way sync the actual sync data is only sent in one direction. This could be used for a public server that gives out information, like the “Drehscheibe”, which is a university calendar. Any student could connect to it and download her schedule, but not change anything.

Even a sending only client seems possible. Some E-mail programs, for example, have the capability to automatically add every person you have sent mail to to your address book. This would be a one way sync requesting the addition of an entry, if not already present.

Chapter 7. Handling conflicts

> *No doubt there are other important things in life besides conflict, but there are not many other things so inevitably interesting. The very saints interest us most when we think of them as engaged in a conflict with the Devil.*
>
> *Robert Lynd, The Blue Lion*

In an ideal scenario, any entry would only be changed or deleted on one client and then immediately synchronized with the server and all other clients. Unfortunately this is rather rarely the case. In practical use, the time between synchronizations may be very long. In the mean time, the same item gets changed on different clients, or even deleted on other clients.

These possible inconsistencies are called conflicts. Usually it is up to the server to detect these conflicts and provide a resolution. We will now discuss which types of conflicts can occur and how they could be solved.

Changed on two clients

The easiest conflict is an item that has been changed on two different clients: Both clients have synchronized at some point in time. Then, the same entry has changed on both clients. When the first client connects to the server, a regular sync happens. Then, when the second client connects, the server detects that this entry has changed on both, the client and the server. It does this by comparing the date of the last sync with the date of the last change of the item. Now both entries have to be merged as explained in the next section:

Merging entries

There are different ways to merge two entries. To minimize information loss, the merging is done on a field-per-field basis. Several things can happen:

- Both fields are identical or both fields are not set. This is trivial.
- A field is set in one version but not the other. The server verison could be kept, or the client version. Just keeping this field ensures minimal data loss.

- A field is set in both the client and server version. There is no way of automatically detecting which one to keep. The server or the client version could be kept.

The best solution to this problem would be to keep a modify timestamp for each data field. Unfortunately this would need way to much memory on thin clients. Even on servers this would greatly increase overhead. So we have to find another way:

It is not so obvious how this situation could be handeld. To decide on which version to keep we will take a look the this example first: A server has synchronized with two different clients. All three contain equivalent data records:

Table 7-1. Merge example setup

Data	Client A	Server	Client B
FN	Max Berger	Max Berger	Max Berger
Ermail;Internet	max@xslt.de	max@xslt.de	max@xslt.de
Phone;Work	089 / 289 2xxxx	089 / 289 2xxxx	089 / 289 2xxxx
REV	01/01/02	01/01/02	01/01/02

Now the data gets changed on both clients:

Table 7-2. Modified data

Data	Client A	Server	Client B
FN	Max Berger	Max Berger	Max Berger
Email;Internet	m@xslt.de	max@xslt.de	max@xslt.de
Phone;Work	089 / 289 2xxxx	089 / 289 2xxxx	089 / 289 1yyyy
REV	02/02/02	01/01/02	03/03/02

Then client A synchronizes. The data has not changed on the server. So no conflict occurs, the server keeps the new data from client A:

Table 7-3. Client A has synchronized

Data	Server	Client B
FN	Max Berger	Max Berger
Email;Internet	m@xslt.de	max@xslt.de
Phone;Work	089 / 289 2xxxx	089 / 289 1yyyy
REV	02/02/02	03/03/02

When client B synchronizes, a conflict occurs and the data has to be merged. It is up to the server to decide which version to keep. For illustration, we will show both:

Table 7-4. Data after merge

Data	Server keeping own version	Server keeping client version	Client B
FN	Max Berger	Max Berger	Max Berger
Email;Internet	m@xslt.de	max@xslt.de	max@xslt.de
Phone;Work	089 / 289 2xxxx	089 / 289 1yyyy	089 / 289 1yyyy
REV	02/02/02	03/03/02	03/03/02

Although some modifications get lost, it seems better to keep the server version. We do not know how much time has passed between both synchronizations. Other clients might have synchronized inbetween. Keeping the client version would allow the client to overwrite data with an old version.

Deletion conflicts

A deletion conflict happens whenever an item is soft deleted that has previously been hard deleted from the database. In this case the soft delete can be safely ignored.

Detecting existing entries

In case of a slow sync or any addition of a supposedly new item it is necessary to find out if an identical or similar item already exists in the database. Usually this is what UIDs are for. But unfortunately, we cannot rely on a UID since an entry most likely will have two different UIDs when originating from two different clients. Or maybe the same entry has been entered in two different address books which were not able to sync until now. So we have to find identical items. But how similar is identical? Or otherwise, how do we know which items can be safely merged?

Comparison by points

To find identical items we have to compare entries on a per-field basis. First of all, there are the two trivial cases: All fields identical and no fields identical. In the first case, one can safely assume that the same entry can be used while in the second case a new entry can safely be created.

We have to define a numerical “uniqueness” of every field to find out which items are identical. A phone number, for example, might be a good indicator for uniqueness. However, if two people share a work phone, this is not enough. But an E-mail address combined with a phone number? Or maybe first name, last name and phone number?

To add to this chaos, items might be more or less unique depending on the user. If you usually contact only one person in a company, a company name or a work address might be unique. Therefore, any algorithm must be user configurable.

The solution is to use a point system. Points are added for every identical item and subtracted for every differing item. An item that exists in one but not the other entry is ignored. If the points are more than a certain number, the items are taken as identical. Of course, the point distribution itself is fully user configurable.

Example

Let us consider the following configuration:

Table 7-5. Example Setup

Field	Identical	Different
First Name	+10	-20
Last Name	+10	-40
Email;Internet	+10	-20
Phone;Home	+10	-20
Phone;Work	+10	-20
...	...	...

Points needed: 25

And the following user Entries:

Table 7-6. Example Data (Client)

	1. Entry	2. Entry
First Name	Max	Test
Last Name	Berger	User
Email;Internet	max.berger@xslt.de	
Phone;Work		
Phone;Home	089 / 8971xxxx	089 / yyyyyyyy

Table 7-7. Example Data (Server)

	1. Entry	2. Entry
First Name	Max	Another
Last Name	Berger	User
Email;Internet	max.berger@xslt.de	

	1. Entry	2. Entry
Phone;Work	089 / 289 - zzzzz	
Phone;Home		089 / yyyyyyyy

When comparing these entries we get numerical results. The following table tries to visualize this: we draw a matrix, putting the entries originating from the client database on top and those from the server database on the left side. The table contents in the middle are the comparison points:

Table 7-8. Comparison Points

	Max Berger	Test User
Max Berger	+10+10+10 = 30 > 25	-20-40 = -60 < 25
Another User	-20-40-20 = 0 < 25	-20+10+10 = 0 < 25

In this case, both “Max Berger” entries are considered identical while “Test User” and “Another User” are considered different. Both “Max Berger” items are merged and now we get the following results in the server:

Table 7-9. Merged Example Data

	1. Entry	2. Entry	3. Entry
First Name	Max	Test	Another
Last Name	Berger	User	User
Email;Internet	max.berger@xslt.de		
Phone;Work	089 / 289 - zzzzz		
Phone;Home	089 / 8971xxxx	089 / yyyyyyyy	089 / yyyyyyyy

Special last name handling

One problem with the point system is that every entry in one database has to be compared with every entry in the other database. In my personal setup with about 100 contact entries this multiplies to 10,000 comparisons. This is far to much.

The solution: Find some kind of preselection. A field that, if present, usually does not differ on different clients. And it should be a field that is present in almost any entry. Possible fields are:

First Name

Unfortunately a first name has often different spellings. Most people use nicknames instead of the real first name, and might not do so on all clients.

Birth date

: A birth date never changes. Unfortunately, birth dates are usually not the thing people put on their business cards.

Last Name

: There are only two ways a last name changes: either by marriage or when it is simply misspelled. It last name could also differ if it is not set.

So the decision is on the “Last Name” field: Entries are only considered for comparison if the last name equals. This lets us optimize the database for last name comparison. In my personal setup this reduces the comparison of entries to one or two in the most cases, and once up to six. This reduces the number of full comparisons needed to about 150

III. Realization

Chapter 8. Raw design

Make everything as simple as possible, but not simpler.
Albert Einstein

Now that we know how to handle all the details, the only thing left is how all the pieces fit together.

Requirements

To understand which decisions are being made during the design, we have to take a look at the following discussion points and their rationales:

Use the SyncML protocol

Based on the discussion in Chapter 4 we chose the SyncML protocol for handling the actual synchronization process.

Put all functionality in a common library

It must be very easy to adopt any existing client or server to the new protocol. Therefore, all protocol specific functions must be hidden under a layer of common functions.

Extensibility

Internet standards evolve quickly. The programs must be written with extensibility and many possible future uses in mind.

Speed

The library must be fast. A server could have a lot of requests and should be able to handle all of them within reasonable time.

Low memory footprint

The core functionality should also be available on low end computers or handhelds. Usually they have very little RAM available.

Secure and error proof

This does not only apply to other broken implementations, but also maliciously sent false packets. The framework must not crash or compromise security, no matter what it receives.

The concept

The project will consist of three parts, which are:

libsyncml

The core library. All protocol related functions are kept here. Also, all connection related functions are in here.

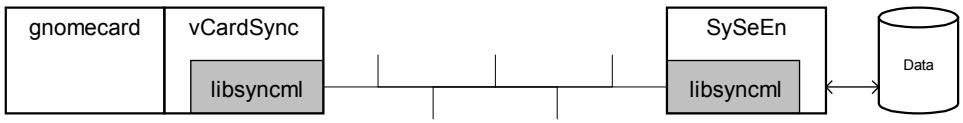
SySeEn

SySeEn stands for Sync Server Engine. This is the server module. It should be very small and basically an adaptor for the library to an SQL back-end. It also handles conflict resolving.

vCardSync

Instead of writing a new client, an existing one is used: Gnomecard. vCardSync will be the adaptor from Gnomecard to libsyncml.

Figure 8-1. Concept overview



The last major decision is the choice of a programming language. For extensibility and reusability an object oriented approach seems reasonable. Counting only the currently most widespread languages this leaves a choice between C++, Java and Python. Java and Python are more portable, but when it comes to speed they fall far behind. Also, Java is known to be a huge memory hog. So the decision was for C++.

Chapter 9. Libsyncml

> *There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.*
>
> *C. A. R. Hoare*

When designing the core library, several different aspects had to be taken care of. This chapter describes which problem occurred and how the design decisions are made.

Design issues

Event parsing or tree parsing?

There are two common methods of parsing incoming XML messages.

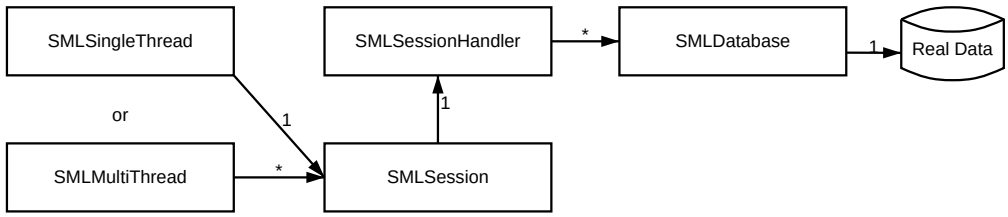
One is to parse the document word by word and take all XML tags as events. An event handler is called for every item: tag open, tag close and text. This is a fairly easy approach with a low memory footprint. However, it has some drawbacks: There is no guarantee that the parsed document is actually valid in the XML sense. Nodes could be opened and never closed.

The other approach is to parse the whole document, building up a tree in memory and then passing it to the program. This approach makes handling the contents very easy: We can freely move along the data. And the tree in memory is always valid. However, this method has other drawbacks: The first one is memory usage: A whole packet has to be kept in memory. This is acceptable for desktop computers and servers, but removes the possibility to use the software on thin hand-held devices. The other problem is that some events should be handled as soon as they are received.

So, what is the solution? It is a combined approach: Take advantage of both and leave out the disadvantages. We take a standard XML parser and use it to receive the XML events. With this information a tree is build in memory. As soon as an event is received completely it is handled if possible. After that the used information in the tree is freed.

Multiple sessions, single databases?

Figure 9-1. Overview of SMLSingle/MultiThread, SMLSession, SMLSessionHandler and SMLDatabase



A single sync session with a single database would be no problem. Unfortunately, there might be multiple sync sessions going on at the same time or multiple databases on one server. Figure 9-1 shows how this is represented in the class model.

User visible

The following classes and definitions are visible for the user of the library:

SMLType

Data nodes within the SyncML package tree can be attributed with the meta information `Type` and `Format`. `Type` specifies the media type of the content. It uses the standard MIME content-types. The default format is `text/plain`. The `Format` field specifies the encoding format for this data field. The most important encoding formats are `chr` and `b64`. `chr` is the default format and means clear-text or specified someplace else. `b64` is for Base64 encoding, which is used for binary data.

The `SMLType` class handles these values and their default values. It is responsible for inserting the meta information into packages where needed and leave them out where the default values are set.

SMLURI

Another thing that has to be handled correctly are URIs within the SyncML package tree. Some URIs might be absolute, and some relative. It is the purpose of this class to give a unique representation, so URIs can be comparable.

Another purpose of this class is to handle the `LocName` property of SyncML URIs. This property is not used during the sync process itself, but may be used to describe URIs for the user in program outputs.

SMLDevInf

The SyncML specification defines a way to exchange device information. Device information contains things such as the type and vendor of a device, serial numbers, firmware versions, etc. It also contains vital information such as the maximum message size and the space left in the device.

This device information is, of course, exchanged in an XML representation. To hide all this from the user, a `SMLDevInf` object is used. The `SMLDevInf` object contains all device information for one device and provides access functions for it.

SMLSessionHandler

The `SMLSessionHandler` is the main class that the users of this library have to derive from. It contains a lot of callbacks vital for session handling, such as get / receive device information, or find out who we talk to. A SyncML program might have multiple sessions (usually servers) or just one session (usually clients).

SyncMLDatabase

The `SMLDatabase` is the adaptor for the real databases. A `SMLDatabase` object is needed for each real database. The `SMLDatabase` object has to tell the session handler which database entries have changed. It also receives the change information of the remote device.

It is important to note, that different session handlers might have access to the same `SMLDatabase` object. It is therefore mandatory to take care of locking issues in a multitasking environment.

SyncMLSingleThread

The `SyncMLSingleThread` class is responsible for connecting incoming and outgoing connections with a session handler. It can only handle one session at a time. This ensures that the user has not to deal with multi-threading issues.

SyncMLMultiThread

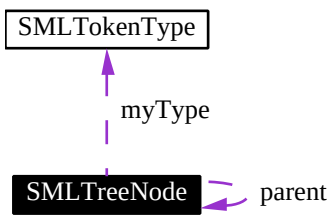
The `SyncMLMultiThread` also connects incoming requests to a session handler. As the name suggests, it is capable of handling multiple requests at the same time.

Internal

SMLTreeNode

XML defines a standard way to describe tree like structures. To keep them in memory a standard approach is used: each tree node is an object, with a reference to its parent and a list of children.

Figure 9-2. A tree node object



All SyncML packages are internally represented in this tree notation. A tree is represented by a pointer to its head node.

SMLNamespaceContainer

To mix XML documents from different sources, the XML specification defines namespaces. The SyncML protocol itself uses three different namespaces: One for the protocol itself, one for device information, and one for meta information. But these are not the only namespaces that can occur in a package: If the data itself is represented in XML, then it might also have its own tags and namespaces.

Internally, however, a tree node type is not represented by the actual string and its namespace. This would be way to expensive for comparison. Therefore internally a numeric representation is used.

The `SMLNamespaceContainer` takes care of all this. It maps the numerical representation to its string representation and vice versa. It has all SyncML namespaces built in and extends itself for foreign tags and namespaces.

SMLFlattener

Keeping the tree in memory is nice, but sometimes it has to be sent out to another device or maybe saved to disc. The `SMLFlattener` is responsible for creating an XML representation of the SyncML package tree. This class can be extended: The `SMLNiceFlattener` for example takes care of formatting the output with line breaks and indention.

SMLResponsePacket

Before a response package can be sent out, it has to be built first. The `SMLResponsePacket` class handles the creation of the response packets. It starts out with a reasonable default that can be changed. It also makes sure that the resulting packet conforms with the specification. It even handles such things as the actual sending.

SyncMLParserCallback

This is an interface class for callback from `SyncMLParser`. It is used so that the actual XML parser can be exchanged, and no other code would have to be changed in the library.

SyncMLParser

The `SyncMLParser` class is an adapter for an XML parser. It currently uses Libxml from the gnome project. But it is planned to also support Xerces (from the Apache project) in the future.

SMLSession

The `SMLSession` class does the actual session handling. It knows about the incoming and outgoing connection. It receives the SyncML commands and calls the appropriate functions from the session handler or the database adapter. It is also responsible for error handling.

Chapter 10. Sync Server Engine

Computers are useless. They can only give you answers.
Pablo Picasso

The server sleeps on one machine and listens for TCP connections on a specified port. Whenever a client connects, it auto detects whether it is raw TCP, HTTP, or HTTPS encapsulated.Then it starts a new SyncML session with the connected client.

Whenever synchronization conflicts occur, the server uses the methods described in Chapter 7 to resolve them

Configuration

The server needs some kind of configuration file. It needs to know which port to listen to and how to connect to its database. Instead of looking for a config file library or even writing a new one, a much simple solution is used: The configuration file itself is specified in XML. Libsyncml has to be linked with an XML parser anyway, so using the same parser for config files adds no extra dependency.

Back-end database

Instead of writing our own database, an existing database is used. There are many public available databases: Libdb, Mysql, and Postgresql are the most common. Unfortunately, each database has its own access library. To solve this issue, several people have written global database access libraries. When looking for a meta library, the things it should have are:

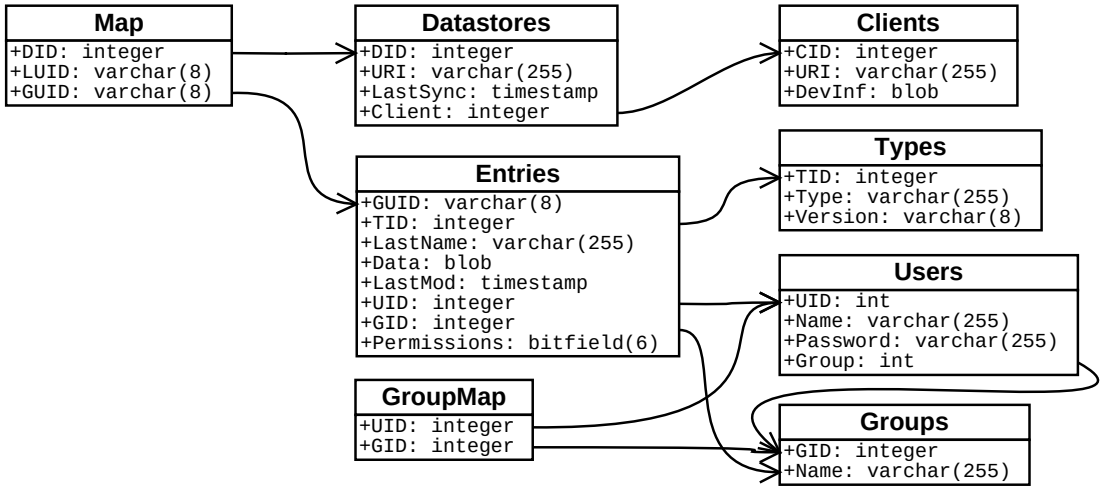
- It should be easy to prgram.
- It should not require many other libraries.
- It should support as many databases as possible.

One of these meta libraries is iODBC (http://www.iodbc.org/). It also has a nice C++ wrapper called Sqlxx (http://www.ailis.de/~k/projects/sqlxx/). There is no particular reason in chosing excactly these libraries, excapt that they fulfill the requirements mentioned above.

Database model

Now that we know where to store the data, we also have to look into how we store the data. The database model show in Figure 10-1 seems reasonable:

Figure 10-1. SySeEns database model



Users

This table holds the basic user information. The UID is only used internally for reference from the other tables. The name and password are used for authentication via the SyncML layer. The password is stored in clear text. The Group property holds the group that newly created entries will belong to.

Groups

This provides a mapping between the group id (GID) and a name for that group.

GroupMap

There are two ways a user can belong to a group. One is having the group entry in the Users table. The other way is an entry in this table

Map

This table maps the client (local) UIDs (LUID) to the server (global) UIDs (GUID) for every client. This table will be quite large, since one entry has to exist for every possible mapping. If an item is soft-deleted on a client then the LUID field will be empty.

Client

The Client table contains information about every client syncing with this server. The numeric client ID is used internally. The LastSync is used to find out which objects have changed. Also the clients device information is cached in here.

Types

The type table maps an internally used TID to a MIME type and its version.

Entries

This table holds the actual data. The GUID is what would have been the UID within the data field. TID holds a reference into the types table. LastName is used for speedup as explained in the Section called *Special last name handling* in Chapter 7. LastMod is used to find out if this object has been changed since the last sync. UID, GID and Permissions are used for access control. Finally, Data holds the actual data in the specified format.

Security

There are three different security aspects that have to be considered on a server: transport security, access security, and storage security.

For transport security we delegate the issue to the underlying transport protocol. One example is to use HTTPS instead of HTTP. The server is configurable to accept one or the other from different IP addresses. So I could use the unsecured transport in a secure environment, like a private network and allow Internet based access via a secured transport only.

For access security we use a model close to the standard Unix file security model. Each entry has read and write bits for owner, group and other. A user can specify which entries are public, somehow public and private. The default flags can be specified in the configuration file.

The storage security is delegated to the underlaying database and the underlying file system. This usually means that the database administrator and the system administrator are able to read all data. But this is a common practice, and there are usually much more valuable files on a system than personal schedules.

Chapter 11. vCardSync

> *The reasonable man adapts himself to the world; the unreasonable man persists in trying to adapt the world to himself. Therefore, all progress depends on the unreasonable man.*
>
> *George Bernard Shaw*

Now we could create another new graphical client, with all the functionality one could possiblly want. But this would be far beyond the scope of this work. Instead the well known Gomecard is used as an application. It supportss all mayor features needed in an addressbook. Its data is stored in a single addressbook file. This file is in vCard 2.1 format. Gnomecard uses Libversit for accessing this data file.

Libversit

Libversit was once the reference implementation for the Versit data format, as described in the Section called *The Versit format* in Chapter 4. But the Versit consortium ceased to exist and so did the original source of this library. Different projects, however, still used this reference implementation in their own programs. Two of these projects were Evolution and Gnome-pim.

Instead of copying the code from one of these two projects, my idea was to split Libversit out of both projects and make it its own library again. During this this process I became the co-maintainer of Libversit and with this a Gnome developer.

Libversit handles reading and writing of Versit data. It takes care about encoding and decoding of binary data. Since the same library is used in Gnome-pim, it is assured that the data is fully interchangeable.

Invoking vCardSync

vCardSync is called manually by the user. It might be integrated into future versions of Gnomecard. It needs two parameters to do its work: The vCard data file and a server configuration file:

The vCard data file is a plain text file in vCard format. This is the format that Gnomecard uses to store its data.

The server configuration file must hold certain information: First, it must contain information on how to connect to the server. Also, necessary authentication data has to be specified. Then, it must keep which data entries have been synchronized in the last session with this server. And it must contain the LastSync timestamp. This is important to recreate the changelog information needed for the sync process as specified in the Section called *Transaction logs* in Chapter 6.

The sync process

First of all, the changelog data is recreated. The rest is pretty straight forward: Establish a connection to the server. Authenticate, if necessary. Then find out which entries have been added, deleted or changed. Send them. Receive the new information from the server. Back up the old database and then overwrite it with the new data. Send back UID mapping information and store all necessary information for the next sync.

IV. Perspective

Chapter 12. Application and future uses

Every advantage in the past is judged in the light of the final issue.

Demosthenes, first Olynthiac

Now that it is possible to synchronize a client with a server, the scenario from Figure 2-1 comes closer to reality. I could place my own server on some machine that is reachable from all my other machines. This server would hold all my personal information. It would hold my address book, and most important my schedule.

On my desktop computer at work I would use my very comfortable commercial PIM program. This program would have a lot of pseudo-intelligence, making my daily tasks much easier. It would remind me of scheduled meetings ahead of time.

On my desktop computer at home I would use a free PIM program. Although this program lacks some of the features, it still uses the same database.

Whenever I send an E-mail, no matter whether from home or from work, the recepients would automatically be added to my address book. I compose a new E-Mail and my address book would be searched through for possible receipients, thus reducing the possibility of failure.

But not only desktop computers are part of this. Whenever I meet someone, I would just take out my Palm Pilot and note down the contact infomation. Then we need to schedule an appointment. I would always have my complete schedule with me, so this is no problem.

Phone numbers would automatically be donwloaded into my cell phone. No more searching through other address books to find a number. No more wondering: I got called from this number. But who could it be? And no more calling home: “Could you please look in my address book on the desk and find me the number of xy?”

The synchronization, however, is not limited to one person. A group of people could share the same server. Whenever they schedule a meeting, this entry will automatically be available on every ones PIM client. Also, new people would automatically end up in the contact database. This would give simple groupware possibilities at no extra cost.

Also, the synchronization process is not limited to personal information. Other things can be synchronized too: A digital camera could use the sync process to download its images onto the computer. It would only download the new pictures. Those pictures could be made available to friends and family via the same sync server.

What I would very much like to see is the usage of E-Mail as a SyncML transport. This would put up many new demands on both client and server, but it would enable dial-up lines to even do the synchronization process asynchronousely.

V. Appendix

Appendix A. Used software and tools

The final version of this document was edited with Adobe Framemaker using the DocBook SGML application. It was then processed with GNU Make, GNU Sed, and Recode to produde a correct DocBook output. This was converted into a printable format by using Jade, JadeTex and Norman Welsh´ DSSSL StyleSheets for DocBook.

Intermediate versions were edited with Vi, Emacs and Word Perfect. They were processed with Xalan or Libxslt using Normal Welsh‘ XSLT StyleSheets for DocBook. For a final printable format Fop and PassiveTex were tested.

The graphics in this document were drawn using Adobe Illustrator, Microsoft Visio, XFig, Dia and Adobe Photoshop. They were converted with Imagemagick and Ghostscript.

The software is written with GNU compiling utilities: GNU Make, GCC and GNU LD. It uses the libraries Libversit and Libxml from the Gnome project, Libsqlxx from Klaus Reimer and Libcommonc++ from the GNU project.

Appendix B. Acknowledgements

Special Thanks go to

Prof. J. Schlichter

for accepting my work as a Diploma Thesis.

Dr. M. Koch

For supporting me in my work and keeping me on the right threads.

Dr. E. Berger

for proof-reading and stylistic suggestions.

Cand. Phys. B. Liebscher

for proof-reading and layout suggestions.

Norman Welsh

for DocBook, Jade, and his StyleSheets.

Glossary

Gnu Public License

GPL

One of the three most common used licenses in free software. Software derived from or linked with GPL software also has to be licensed under the GPL.

Hyper Text Transport Protocol

HTTP

The stuff that the World Wide Web is made of. A protocol to transport text files across TCP networks.

iCalender

Next generation of the vCalendar format. Explained in the Section called *iCalender and iTIP* in Chapter 4.

Infrared Data Association

IrDA

The Infrared Data Association defined a standard how electronic devices connect and exchange data using infrared signals.

Lightweight Directory Acces Protocol

LDAP

A database access protocol. Explained in the Section called *Lightweight Directory Access Protocol (LDAP)* in Chapter 4.

Gnu Lesser Public License

LGPL

Software derived from LGPL software also has to be licensed under the LGPL. Unlike the GPL, software linked with LGPL software can be published under

any license.

Multipurpose Internet Mail Extension

MIME

A standard to describe the type and encoding of data outside of the actual data.

Object Exchange

OBEX

The Object Exchange protocol is used when a Palm Pilot connects with a PC.

Personal information

The information people used to have in their little black notebook. The most common personal infomation is schedule, to-do list, notes, and address book.

Personal Information Manager

PIM

Any device or program that handles *Personal information*. Some of the most common programs are MS Outlook, Gnomecard and Gnomecal, Kab and KOrganizer.

Simple API for XML

SAX

A standard for XML parsers.

Transmission Control Protocol

TCP

The TCP allows two computers to exchange data streams.

Unique Identifier

UID

Usually a number or another string that exists only once thus uniquely identifying an entry. Explained in the Section called *Unique identifiers* in Chapter 6.

vCalender

Versit format for calendar and scheduling information.. Explained in the Section called *The Versit format* in Chapter 4.

vCard

Versit card format for business cards. Explained in the Section called *The Versit format* in Chapter 4.

WBXML

A binary representaion of the XML format. Used for small devices and wireless links.

eXtensible Markup Language

XML

The idea of structured documents is actually as old as document processing itself. With the internet hype came the XML hype, and that is why many current standards are described in XML. More on the w3c website.

Bibliography

Books and Papers

[Borghoff] Uwe Borghoff and Johann Schlichter, 1995, *Rechnergestütze Gruppenarbeit, 3-540-58119-7, Addison-Wesley.*

[Vossen] Gottfried Vossen and Margret Groß-Hardt, 1993, *Grundlagen der Transaktionsverarbeitung, 3-89319-576-9, Addison-Wesley.*

[SynchroXML] Mirko Mrowczynski, October 30, 2001, *Synchronisation von Terminplanern mittels XML, Diplomarbeit an der TU Chemnitz.*

Specifications

[vCard21] versit Consortium, September 18, 1996, 2.1, *vCard: The Electronic Business Card.*

http://www.imc.org/pdi/

[RFC 2251] *Lightweight Directory Access Protocol (v3).*

[RFC 2252] *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions.*

[RFC 2253] *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names.*

[RFC 2254] *The String Representation of LDAP Search Filters.*

[RFC 2255] *The LDAP URL Format.*

[RFC 2256] *A Summary of the X.500(96) User Schema for use with LDAPv3.*

[RFC 2425] T. Howes, M. Smith, and F. Dawson, September 1998, *A MIME Content-Type for Directory Information.*

[RFC 2426] F. Dawson and T. Howes, September 1998, *vCard MIME Directory Profile.*

[RFC 2445] *Internet Calendaring and Scheduling Core Object Specification: iCalendar*.

[RFC 2446] *iCalendar Transport-Independent Interoperability Protocol (iTIP): Scheduling Events, BusyTime, To-dos and Journal Entries.*

[RFC 2447] *iCalendar Message-Based Interoperability Protocol (iMIP).*

Online Resources

[http://www.ailis.de/~k/projects/sqlxx/] *K's cluttered loft - Projects: Project: sqlxx.*

[http://www.gnome.org] *GNOME: Computing made easy.*

[http://www.gnu.org] *GNU's Not Unix!*.

[http://www.iodbc.org/] *Platform Independent OBDC*.

[http://www.openldap.org] *OpenLDAP: Community developed LDAP software.*

[http://www.palm.com] *Palm.com: Products, Services & Company Information.*

[http://www.starfish.com] *Starfish Software: Smart connected solutions.*

[http://sync4j.sourceforge.net] *sync4j homepage.*

[http://www.syncml.org] *SyncML: The new era in data synchronization.*