

Collaborative scheduling using context-awareness

ALEXANDER RIEDEL



**KTH Information and
Communication Technology**

Master of Science Thesis
Stockholm, Sweden 2010

TRITA-ICT-EX-2010:35

Collaborative scheduling using context-awareness

Master of Science Thesis

Alexander Riedel

ariedel@kth.se

March 25, 2010

School of Information and Communication Technology,
Royal Institute of Technology (KTH), Sweden

Examiner and Supervisor: Prof. G. Q. Maguire Jr.

Industrial Supervisor: Henrik Gustafsson, Telepo

ABSTRACT

Today most cellular phones, personal digital assistants, PCs, etc. offer an electronic calendar. Electronic calendars are especially useful for people who have many different meetings each day and who need to know when the meetings start and who is involved in each meeting. With the aid of a program a calendar can be published on web, shared with other people to enable collaboration, or synchronized between different devices.

Current calendaring software offers an almost unlimited set of features and services. However, today such software does not utilize context-awareness, for example exploiting knowledge of the user's location.

When people collaborate they often need to meet in order to do a task jointly or discuss something. It can be difficult to plan a meeting because people have booked their available time differently in their calendars. Because of this there is a need to automatically schedule certain types of meetings. In this thesis, a program that schedules meetings automatically is designed, implemented and evaluated. This program facilitates collaboration by finding a commonly available time and/or meeting place for a meeting, thus making it easier for the meeting people to agree.

When meetings are scheduled without requiring too much attention from a user and the number of human errors can be reduced while planing a meeting, users do not need to expends as much effort as it goes into scheduling meetings today. Because today a company planing a collaboration task collectively spends a lot of time and effort searching for a commonly available time with this effort increasing non-linearly with increased numbers of participants companies can obviously benefit from automated scheduling systems.

Testing with the application reveals that incorporating of user's location information into scheduling is a great tool to facilitate collaboration. The survey also shows the need for extensions to the developed application; with the new features utilizing location information. The evaluation also shows that the developed scheduling program has managed to reduce the time and effort spent while scheduling meetings.

SAMMANFATTNING

Idag finns det en elektronisk kalender i de flesta mobiltelefoner, datorer och PDA:n. Elektroniska kalendrar är användbara framför allt för människor som har flera möten varje dag och som behöver veta när mötena startar vilka som ska delta. Vissa elektroniska kalendrar kan publiceras på webben, delas med andra människor för att möjliggöra samarbete och synkroniseras mellan till exempel mobiltelefoner, datorer och PDA:n.

Kalendermjukvara erbjuder idag ett nästan obegränsat antal funktioner och nyttiga tjänster. Denna typ av mjukvara är dock generellt sett inte medveten om information såsom användarens position, vilket i sammanhanget kallas context-awareness.

När människor ska samarbeta krävs ofta att de träffas för att utföra uppgifter tillsammans eller diskutera viktiga ämnen. För att kunna ha möten krävs att möten först planeras, vilket kan vara svårt då de inbjudna är olika uppbokade i sina respektive kalendrar. Av den anledningen finns ett behov av att automatisera schemalaggningen för vissa typer av möten. I detta examensarbete skall ett program för automatisk schemalaggning designas, utvecklas och evalueras. Programmet skall underlätta samarbete mellan mötesdeltagare genom att ta över uppgiften att hitta en gemensam tid och/eller plats för ett möte. Programmet skall därmed också underlätta för mötesdeltagarna att komma överens.

När möten kan schemaläggas utan att det kräver för mycket uppmärksamhet från användarna och antalet mänskliga fel kan reduceras när möten planeras, behöver man inte lägga lika mycket arbete, som idag, på att schemalägga möten. Eftersom det för tillfället krävs mycket tid och resurser för ett företag för att schemalägga ett möte, samtidigt som tiden för att planera ett möte inte ökar linjärt med antalet deltagare, kommer företag antagligen att dra nytta av ett automatiserat schemalaggningsystem.

En undersökning genomförd av ett antal testpersoner som använt applikationen visade på att användarens position var en viktig parameter som kunde förbättra schemalaggningen av möten. Undersökningen visade också att applikationen hade ett stort behov av att vidareutvecklas genom nya potentiella funktioner som tar hänsyn till användarens position. Men viktigast av allt så visade undersökningen på att applikationen lyckats med att reducera tiden det tar för att planera möten.

Acknowledgment

I would like to express gratitude to my academic supervisor and examiner Prof. Gerald Q. Maguire Jr. and my industrial supervisor Henrik Gustafsson for their help and support throughout the whole project. I would not be able to produce this report without the ideas and suggestions I received from both supervisors. In addition, I have gained enormous amounts of knowledge during this project, at the same time I understand how important a supervisor's helping hand might be. Thank you for this.

I would like to thank the Telepo for providing me both with a work station and a mobile device to program and use for testing. I would to thank all the employees and other people who participated in my survey. My close friends and my girlfriend have supported me spiritually and that was also important. Thank you all!

Table of contents

<i>Abstract</i>	i
<i>Sammanfattning</i>	ii
<i>Acknowledgment</i>	iii
<i>List of figures</i>	viii
<i>Acronyms and abbreviations</i>	xii
1 Introduction	1
1.1 Electronic calendaring	1
1.2 Problem description	2
1.3 Use cases	2
1.3.1 Use Case #1: Eating lunch at your favorite restaurant	2
1.3.2 Use Case #2: Discussion with your teacher	3
1.3.3 Use Case #3: Just another day at work	4
1.3.4 Use Case #4: Meet your friend	5
1.4 Objectives	6
1.5 Methodology	7
1.6 Thesis overview	7
2 Terminology and Concepts	8
2.1 Context-awareness	8
2.1.1 What is context?	8
2.1.2 Context parameters and context sensing	9
2.2 Calendaring framework	9
2.2.1 iCalendar	9
2.2.2 iTIP	14
2.2.3 iMIP	16
2.2.4 The CalDAV protocol	16
2.3 Location	22
2.3.1 Detecting nearness and co-location	22
2.3.2 Geopriv	23
2.3.3 Demands on positioning systems	25
2.3.4 Location representation	25
2.3.5 Location in calendar events	27
2.3.6 Positioning techniques and services	27

3	Related Work	30
3.1	Calendar software	30
3.1.1	Chandler	30
3.1.2	Bedework	33
3.1.3	Peer-to-peer calendaring	34
3.1.4	Other calendar alternatives	36
3.1.5	Conclusions	36
3.2	Location-aware applications	37
3.2.1	Location-aware browsing	37
3.2.2	Google latitude	38
3.2.3	The monger application	39
3.2.4	Lcron	39
3.3	Automated meeting scheduling system	40
4	Realizing of the Scheduling Application	42
4.1	Application overview	42
4.2	Step #1: Meeting setup	43
4.3	Step #2: Pre-scheduling	43
4.3.1	Meeting place priorities	43
4.3.2	Learning from past events	44
4.3.3	Example: Choosing the closest meeting place	46
4.3.4	Meeting place candidates	47
4.3.5	Example: Eliminating meeting place candidates	47
4.4	Step #3a: Relaxing constraints	49
4.5	Step #3b: Agreeing upon meeting occasion(s)	49
4.5.1	Filtering of meeting occasions	50
4.6	Step #4: Smart notifying	51
4.6.1	Participant's arrival status notifications	51
4.6.2	Reminders of a meeting	52
4.7	Moving a meeting to a different time	53
4.8	The communication approach	54
4.8.1	Communication approach comparison	55
4.8.2	Calendar view synchronization	56
5	Implementation	57
5.1	Design choices and their effect	57
5.1.1	Choose of the Android platform	57
5.1.2	Developing on Android	58
5.2	Communication between peers	61
5.2.1	Receiving messages	62
5.2.2	Sending messages	63
5.2.3	Types of messages and their meaning	63
5.3	The protocol for scheduling a meeting	65
5.3.1	Choosing the communication approach	65
5.3.2	The time negotiation	66

5.3.3	Rescheduling a meeting	68
5.3.4	The elements of free/busy information	69
5.4	Application overview	73
5.4.1	Meeting parameters	73
5.4.2	Meeting list	75
5.4.3	Meeting scheduling states	77
5.4.4	Notifications	79
6	Experiments with the application	80
6.1	Evaluation method	80
6.1.1	Evaluation questions	80
6.1.2	The evaluation step-by-step	82
6.1.3	Testing scenarios	82
6.2	Expectations	83
6.3	Evaluation results	84
6.3.1	Group A questions	84
6.3.2	Group B questions	86
7	Conclusions and Future Work	88
7.1	Application improvements	88
7.1.1	Suggesting a meeting place	88
7.1.2	Enabling changes to the meeting details	89
7.1.3	Customizing notifications for meetings	89
7.1.4	Additional meeting parameters	89
7.1.5	Supporting multi-device users	91
7.1.6	A better meeting list	92
7.1.7	A better proposal manager	92
7.1.8	Optimize the “schedule now” feature	93
7.1.9	Remove-from-calendar issue	93
7.1.10	The issue about late proposals	93
7.2	Interesting future project	94
7.3	Conclusion	94
7.3.1	Meeting the goals of the project	94
7.3.2	Security and privacy	97
7.3.3	General conclusions	98
7.3.4	The “iCalendar standards”	98
7.3.5	Conclusions related to the evaluation	98
7.3.6	Following up the use cases	99
	Bibliography	101
	Appendix:	
	A The Questionnaire	106
	B The results of the survey	109

C	Introduction to the Scheduling Application (SCAP)	112
C.1	Meeting parameters	112
C.1.1	Attendee parameters	113
C.2	Meeting list	114
C.3	Meeting scheduling states	115
C.4	Rescheduling a meeting	116
C.5	Notifications	117

List of Figures

2.1	A “calendar file” showing the mandatory parts of a calendar object at the beginning and end of the file.	10
2.2	An example of a calendar entity representing a meeting.	11
2.3	An example of a calendar entity representing a task.	12
2.4	An example of a calendar entity representing a request for a free/busy time window.	12
2.5	An example of a calendar entity representing a reply to a previous request (see 2.4) for a free/busy time window.	13
2.6	A communication example between two clients using a CalDAV calendar server.	18
2.7	Example of a client request for creating a new calendar supporting components that contain only VTIMEZONE components.	20
2.8	Example of a client request for creating a new event.	21
2.9	The four primary Geopriv Entities and their interaction, adapted from RFC 3693 [14].	24
3.1	The ‘home’ page of a Chandler user	31
3.2	It is possible to see additional events in a list and to set the “status” of an event to now, done, or later.	32
3.3	The free/busy view of all participants in Bedework	34
3.4	My current location determined by the ‘Gears’ and shown on a map.	38
3.5	The architecture of the automated meeting scheduler, adapted from figure 1 in [32]. The user interacts with the scheduling system via the User Interface , the User Preferences module stores the preferences of the user, the Working Memory contains the traces of meeting negotiations, the Negotiation Module contains the scheduling logic, the Calendar Manipulator allows the user to access and modify the user’s schedule through a calendaring program, and the Message constructor/decoder component is the interface to send and receive e-mail messages.	40

4.1	The “communication end points” between meeting participants. The dashed arrow between the attendees indicates that they may exchange only a subset of the scheduling information, or no scheduling information at all, between each other, compared to the information they might exchange with the organizer. There might be 3 different types of message pools to choose from, because of the 3 different combinations of end points: 1) organizer → attendee, 2) attendee → organizer, and 3) attendee → attendee.	54
5.1	The Android SDK debug monitor showing that the virtual machine verifier rejects a class that references a non-existing method. This class is part of the CalDAV library that we were not able to use because of the broken reference.	58
5.2	The interaction between the scheduling application (SCAP-marked in green) and other “components” involved. SCAP reads the contacts database, manipulates the calendar database, asks the Location Provider for the current location of the device, and interacts with the k9mail application in order to learn about incoming e-mail messages and in order to send messages. The calendar database on the Android device is synchronized “in the background” with a calendar server with the aid of the calendar synchronizer.	59
5.3	The settings menu of the SCAP application.	64
5.4	A communication example of the scheduling application, where the organizer requests for free busy information after that both attendees have accepted the meeting invitation. The colors of the vertical bars indicate what state the meeting is in for that user at the time. The meaning of meeting states is explained in section 5.4.3.1.	66
5.5	A communication example of the scheduling application, where one of the attendees declines a meeting invitation. The colors of the vertical bars indicate what state the meeting is in for that user at the time being. The meaning of meeting states is explained in section 5.4.3.1.	67
5.6	The figure shows how a time negotiation may look like between 3 participants after that every attendee has accepted the meeting invitation. The colors of the vertical bars indicate what state the meeting is in for that user at the time being. The meaning of meeting states is explained in section 5.4.3.1.	68
5.7	The postpone meeting view.	69
5.8	Each circle represents an area where one user’s true position could be. Here all users are considered to be “close” since each circle shares a common area with all other circles (filled in gray).	70
5.9	An example of a free busy component with a GEO property. Additionally, the GEO property has an accuracy and a time stamp telling when this location information was measured.	71

5.10	An example of a free busy component containing prioritized periods. There are 5 periods given with their start and end times, respectively, to make sure that the receiving peer knows what periods are meant. The sender of this component (user1@host.com) has chosen to give highest priority (5) to the latest period (that will be ending when the related meeting's deadline applies)	72
5.11	The time tab of a meeting editor in SCAP.	73
5.12	The attendee editor, when adding a new attendee. The value of the role of an attendee does not have any effect on the scheduling of the meeting with that attendee. The value of the participation status can not be set, as it is controlled by SCAP and will either be set to ACCEPTED or DECLINED when this attendee has replied.	74
5.13	The list of all attendees in the current meeting showing their display name (= e-mail address and their name, if it exists) in the upper list item view and their participation status in the lower view. The participation status has the initial value NEEDS_ACTION and will be set to either ACCEPTED or DECLINED when this attendee has replied.	75
5.14	The different option menu choices, when pressing the menu button on the Android device, while viewing the meeting list.	76
5.15	The meeting list view. Users can see the most important meeting details, like meeting summary and scheduled time, in each list item. The meeting state can be identified by the color of the vertical bar immediately to the left of each meeting, respectively. See section 5.4.3.1 for a detailed description of the scheduling states. The blue-green folder-like icon to the right indicates that the user of this device is not the organizer of this meeting. Organizers can also see a summary of the current participation statuses of all attendees: P stands for the number of participants, A for accepted and B for declined. Each list item also displays the meeting start time.	77
5.16	The meeting scheduling states and their possible transitions. For a detailed description of the transition conditions, see section 5.4.3.1. Once the meeting is created, it is in the setup state, while the final meeting state is occurred.	78
6.1	The first question for determining group affiliation of test users.	84
6.2	The second question for determining group affiliation of test users.	85
6.3	The achieved score.	85
6.4	This figure presents the results acquired from question 5 in the survey.	86
B.1	This table shows the answers to all evaluation questions having a Likert scale. The description of this table is given in appendix B.	110
B.2	This figure presents the results to question 6.	111
B.3	This figure presents the results to question 11.	111
C.1	The time tab of a meeting editor in SCAP.	112

C.2 The attendee editor, when adding a new attendee. The value of the role of an attendee does not have any effect on the scheduling of the meeting with that attendee. The value of the participation status can not be set, as it is controlled by SCAP and will either be set to **ACCEPTED** or **DECLINED** when this attendee has replied. 113

C.3 The list of all attendees in the current meeting showing their display name (= e-mail address and their name, if exists) in the upper list item view and their participation status in the lower view. The participation status has the initial value **NEEDS_ACTION** and will be set to either **ACCEPTED** or **DECLINED** when this attendee has replied. . 114

C.4 The different option menu choices, when pressing the menu button on the android device, while viewing the meeting list. 115

C.5 The meeting list view. Users can see the most important meeting details, like meeting summary and scheduled time, in each list item. The meeting state can be identified by the color of the vertical bar immediately to the left of each meeting, respectively. See section C.3 for a detailed description of the scheduling states. The blue-green folder-like icon to the right indicates that the user of this device is not the organizer of this meeting. Organizers can also see a summary of the current participation statuses of all attendees: P stands for the number of participants, A for accepted and B for declined. Each list item also displays the meeting start time. 115

118

Acronyms and abbreviations

ACL	Access Control Protocol
API	Application Programming Interface
CalDAV	Calendaring Extensions to WebDAV
ETag	Entity Tag
FM	Frequency Modulation
GPS	Global Positioning System
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
iMIP	iCalendar Message-Based Interoperability Protocol
iTIP	iCalendar Transport-Independent Interoperability Protocol
IP	Internet Protocol
MIME	Multi-purpose Internet Mail Extensions
PC	Personal Computer
RFC	Request for Comments
SCAP	Scheduling Application
SDK	Software Development Kit
UID	Unique Identifier
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WebDAV	Web-based Distributed Authoring and Versioning
Wi-Fi	Wireless Fidelity

1. Introduction

In a highly developed country there are few people who do not carry around portable electronic devices such as an MP3-player, cellular phone, game console, laptop, etc. Today we are able to integrate lots of (different) functionality into a single device, rather than having our pockets full of devices. For example, a modern mobile phone not only implements all of a traditional phone's functions, but commonly offers the functions of a watch, FM receiver, dictaphone, built-in camera, and provides web access. Not surprising is the fact that a report from 2007 [1, p. 19] made by Statistiska centralbyrån* shows that 94 % of all persons interviewed use a mobile phone, while the corresponding number for any age-group is greater or equal to 80 %. A similar report from 2008 [2, p. 30] indicates that for any age-group, at least 85 % of all persons interviewed use a mobile phone.

1.1 Electronic calendaring

This thesis focuses on an application running on almost all mobile devices (including a large fraction of all mobile phones): a simple electronic calendar where it is possible to view events by day and by time slots. Such an application usually allows its users to also maintain a to-do list consisting of tasks, where each task typically has a due date. The device's internal clock can be used to trigger an alarm at a pre-defined time and date. On a higher abstraction level, tasks and events are associated with a certain calendar and a single user may use different calendars, e.g. one for work and one for personal use.

Because today it is quite common to have different devices to access and view calendars, e.g. a mobile device, computer at work, computer at home, it is handy having a calendaring server where all data is saved so that devices can be synchronized with a common calendar. This is called *subscribing to a calendar*. Users can subscribe to different calendar servers and see their calendars when logged in. Since most calendar servers store calendars for multiple users, it is easy to share calendars with others, while allowing one user to see multiple overlapping calendars at the same time. Calendar events can be marked as public or private events. A common use of this public information is to learn when another user is free/busy in order to schedule a meeting. Yet another helpful feature of calendaring servers, when arranging meetings, is that a contact list can be maintained by the calendaring server - so that a change in the scheduling of the meeting can be propagated to all of the planned participants. By using an integrated mail client all meeting participants can be informed of the scheduled meeting in an invitation by email. There are

*“Statistiska centralbyrån” or Statistics Sweden is an authority managing and collecting information in order to ease decision making. See www.scb.se

many different calendaring programs and each of them implement some subset of the discussed functionality[†].

1.2 Problem description

In today's society people are used to the fact that dozens of events occur at any given time of day, in different locations, and with different persons being involved. Some events are important for us to attend, while other events might be annoying and therefore be on a personal list of events to be ignored. Because it is not easy to keep track of all possible events, many of us take advantage of an electronic aid in the form of a hand-held device that we regularly carry with us.

Some might say that we are almost addicted to our equipment, as many people are dependent on these devices in their every-day life. However, many people are not completely satisfied with the services currently provided. Although some devices are aware of the context around them, most are not. An interesting question is what additional context information would improve the perceived functionality of the device, such that the user would be able to deal with the many events taking place and that they must attend to, re-schedule, or avoid. An example of additional context information is the extension of an electronic calendar to know about the user's current location (see for example [3]).

To make it easier for people to coordinate events, their personal devices need the ability to automatically schedule events based on deadlines rather than being a simple static calendar solely based on date and time. In addition, utilizing context-awareness would allow events to be rescheduled and reminders could be triggered based on factors such as the involved people's location, time to arrive at a place from a known location, traveling speed, personal event priority, deadline(s), existing calendar events and tasks, etc.

In this thesis project we want to (1) exploit location information in calendars for a specific event and users and (2) augment calendars with current location information to fill in details of an unplanned meeting by suggesting a time and a location that would be suitable for the necessary people to meet. We also want to achieve both of these without increasing the burden upon the user, hence we wish to find automatic means of both locating the user and manipulating their calendar(s).

1.3 Use cases

Below we present some use cases to gain a better understanding of the problem addressed in this thesis. In addition to offering potential real scenarios for our future application, these examples will also further demonstrate cases where location could play an important role.

1.3.1 Use Case #1: Eating lunch at your favorite restaurant

Today, you want to eat lunch at your favorite restaurant together with your work colleagues and you know that they will go to this restaurant since they do so each

[†]Examples can be found at http://en.wikipedia.org/wiki/Calendaring_software

Friday. You do not know when they will gather on this date or any other Friday since that depends a little bit on their individual circumstances. Today, Fred needs to finish his business call with a loyal customer; Claudia is in a meeting with a potential recruit; Ryan needs to write “just one more” bug report; even you might be just about to enter the bathroom missing the opportunity to join your fellows. No one can figure out when the time is right.

1.3.1.1 Analysis

The problem addressed in this use case is that people may wait for each other unnecessarily without knowing how long they should or should not wait. A possible solution could be a tool enabling everyone to “register”, indicating that the registered people want to do a task jointly - in this case go out to eat. The tool (a program) could signal everyone when the time is right, for example when all participants have clicked on a button to indicate that they are now ready to go.

Once this kind of “meeting” is setup, the people could save a lot of time asking colleagues if it is time to go (with no one giving a clear yes or no answer in any case). It is easy to click on a button and at lunch time few people will forget to be hungry, giving a guarantee that the “meeting” will start. In addition, when people know that they can trust the service, they can focus on their working (tasks) instead of running around asking others if they are ready to go to lunch.

1.3.2 Use Case #2: Discussion with your teacher

You have agreed to meet with your teacher and another student to discuss the topic of a written report you are going to produce during the next 2 weeks. However, the other student gets stuck in traffic and unfortunately can not make it to the meeting in time. You and the teacher are already waiting, but your classmate has not contacted you in advance, since he is stressed and in addition distracted by the music playing loudly inside his car. Ten minutes after the meeting’s planned start he has still not called, so you give him a call and learn that he will be delayed at least 10 more minutes. Unfortunately, the teacher has many students he needs to talk to and he had only allocated twenty minutes for your meeting.

1.3.2.1 Analysis

The problem of the second use case is that important resources may be wasted, although some might say that the problem actually is the fellow student who did not act responsibly by calling. If this student had called, could the other student and the teacher know that the meeting was going to be cancelled? Could the teacher and the student learn about the delay even without calling each other? It is hard to tell whether the meeting would be cancelled or not - as this depends on the accuracy of the delay that the student driving to the meeting (or his/her agent) determines the likely delay to be, but people can (and should) learn about the delay as early as possible, so that a decision can be made to more easily cancel or reschedule) the meeting.

A possible solution for that use case is a “meeting application”. For example, the teacher could book the meeting (using the application) between himself and the two

students. Because the meeting is planned in advance, a meeting room (and other necessary resources) would be allocated to the meeting. The meeting place would be defined to be that meeting room, thus all participants can plan their traveling routes and know exactly when their travel should begin. Simply knowing all these details reduces the probability of being late, that is, for the start of the meeting be delayed. In case the meeting will be delayed or cancelled because of someone being too late, then one of the following probably occurred: a meeting participant did not start traveling at the planned time, or their travel took more time than planned (this is frequently based on an estimate of the time it usually would take); a meeting participant planned to travel from location A, but this participant is not at that location when it is time to travel, so that he/she must re-plan his/her route (and may be late because it is not possible to complete the travel by this new route in time to arrive at the planned time).

This “application” could provide a user interface for planing and saving a route. The application needs to track the user’s location so that the user could be warned when he/she should start traveling, but has not yet started to travel for example. Either the user takes the planned route, or a new route could be provided by the user or the application, when the user’s originating location no longer matches the originally planned starting location. Either way, as soon as the user’s location deviates from the expected location, the application could inform other participants that this user is going to be late. As soon as this user or the application has recalculated the route, a new estimate of the traveling time and the user’s delay, can be propagated to all the other participants.

1.3.3 Use Case #3: Just another day at work

Its quite common to have meetings with colleagues to discuss and plan the next day’s tasks. For example, if you are a project manager in a software development company, you most likely have many different types of meetings, each with different people involved. As an example, you might have a project status meeting with the developers every day at 9 a.m. except for Monday when this meeting is at 11 a.m. due to an earlier company-wide meeting. On 10 a.m. each Tuesday there is an all employee meeting for status updates. To plan the testing phase and decide what needs to be done during the next weeks (e.g. what bugs need to be fixed, what new functionality is to be added, etc.) there might also be testing and additional development meetings.

You have prepared yourself for the next meeting, but before you start walking to the meeting room you think briefly about calling everyone. However, since there might be a dozen participants, you realize that it is impractically to call each of them. Unfortunately, some of these participants are absent, but their input is needed so the meeting can not start without them. There are many reasons why people might be absent ranging from stopping at the coffee machine, while some are still in the opposite side of the building because their previous meeting just ended, or they are still stuck in traffic due to an accident on the main artery into the city. However, you need to get the meeting started to avoid people waiting for each other unnecessarily. How do you know when to go ahead with the meeting?

1.3.3.1 Analysis

Once again in this use case, people would be happy if they did not have to wait for each other unnecessarily. The problem addressed here is similar to the problem in the first use case: when do people know that the meeting has started without worrying about wasting time by being there before the meeting really starts and without missing the start of the meeting? Again, the “tool-program” introduced in section 1.3.1.1, could be used as a possible solution. However, some might find, that the program could be a bit smarter by knowing where people are, thus learning which people are “ready” or waiting in the meeting room. Depending on the meeting, some participants might already be waiting in a meeting room and they could start the meeting. The application can not know for sure, by knowing the participants’ locations, if people sitting in the meeting room have already started the meeting or are simply waiting for the other participants to show up. However, the meeting details may provide additional information. For example, knowing how many people are participating, which of them are required, and which already are sitting in the meeting room can give a good estimate about the meeting’s status.

The meeting status (whether estimated, or manually given by the organizer, if necessary) could be propagated to all attendees even if they are not currently participating, in order to remind the potential attendees that the meeting has started. Such potential attendees could then start walking to the meeting room, enabling them to potentially avoid missing much of the meeting. In this matter if the potential attendees are in the same company office then everyone can make it to the meeting “just in time”. Additionally, people already in the meeting room could proceed with their collaboration tasks because they know that all the required participants are there, and that all the other people are probably on their way. The application if it functions well removes the burden from the organizer of having to call everyone to remind them of the impending start of a meeting. If some of the participants currently are out of the office, it could take him/her a considerable amount of time to get to the meeting room, hence the application could arrange a conference call, once the meeting has begun.

1.3.4 Use Case #4: Meet your friend

Almost one year ago one of your childhood friends moved from your home town to a different city 500 kilometers away. You miss him and try to keep in touch, but it is not always possible because both of you are concerned with your own problems and do not have much time left over for social activities. Because your friend’s parents are still living in the same city as you do, he and his family sometimes come by and visit them, but they do not always stop by your place. However, you want to see your friend when he is next in town, no matter what he is doing here.

1.3.4.1 Analysis

From the previous use case, it is noted that an application needs to keep track of users’ location information. Such an application desires some kind of context server where location information could be saved for further analysis. In order for users to be notified about a “good opportunity” to meet, the area where a user usually

resides could be considered in order to discover “entry” by other users into that area. When an entry is detected, and both users have configured that they want to meet “as soon as there is a good opportunity available”, a meeting could be booked for both of them at a commonly available (free) time. The problem of knowing where a user *usually* is can be hard to solve, but one algorithm could create a circled area to represent a user’s whereabouts. The circle could be centered at a geographically centered point between all by the users frequently visited places and the circles area must at least contain all such places. It may be up to the individual users what being close to a place means, as closeness also decides how much users are willing to travel in order to meet.

The benefits of such an application are really great, since users could configure it to schedule such meetings automatically for every contact the users might have. People would never have to worry about forgetting to meet. However, one problem is that additional information might be needed in order for such meetings to actually be scheduled, because if people do not go outside their homes, there will not be any good meeting opportunities. Another point here is that such an application must be “taught” to produce desired results, therefore, it needs a representative and rich history of the users’ location information.

1.4 Objectives

Section 1.3 introduced some use cases interesting for this thesis. This section maps these use cases to the objectives necessary for the application to function.

1. From the use case study we have seen that an application could be used in order to specify who is meeting who, in order to undertake something. Because traditional calendaring software exists, the application should be used in conjunction with traditional calendaring software.
2. The communication protocols used for the application must be extensible. In addition, calendar standards and standardized protocols would make it easier for the application to cooperate with existing software.
3. The basic concept is that the application should not demand too much attention from a user, while at the same time keeping the users updated about the scheduling process. It should be relatively easy and fast to set up a meeting and the users should (where possible) be able to configure the level of interaction with the application. Our expectation is that when (and if) the application provides what is promised at an interaction level suitable for the users, they will be satisfied with the service.
4. Almost all use cases require location information to provide a better service. User location can be used in order for the application to create location specific notifications and/or to trigger a certain function of the program. The location information could be saved on a context server and be processed later, for example to extract behavior patterns and learn where users usually are. (Note that for privacy reasons, this information is not public by default).

5. Location information could also be used in order to suggest a (co-located) meeting place. This is a valuable feature for planing *ad hoc* meetings, since such meetings can be conducted anywhere provided a suitable meeting place close to the meeting participants' current/prior position(s). Moreover, the application could make use of location information (together with the information from calendars) to determine a meeting's ability to be rescheduled, i.e. the "cost" of rescheduling a meeting to a different point in time and/or location. This could be useful for setting priorities for meetings, thus making them comparable and deciding which meetings are most important to attend (because of their inability to be moved to a different time).

1.5 Methodology

Before an approach is made to accomplish the objectives of this thesis project, we first need to understand what context is and how it can be acquired and where. Furthermore we need to understand what (standard and extensible) protocols exists so that a decision can be made which protocols are going to be used in the future application and why. Because a device's location information is going to be useful, the concept of location, nearness, and privacy is relevant. For example what does it mean to receive a device's location information? How can location be represented, the distance between two points calculated *and* why location plays an important role in this thesis project?

In order to design an application existing calendaring software might be of importance. Therefore, we will present calendaring software available on the market and draw some conclusions regarding the presented applications' visual and architectural designs. It is important to know how collaboration is facilitated in calendaring applications to get some ideas for this thesis application.

We need to understand what information might be useful in order for an application to propose a meeting place and to automatically schedule a meeting. The concept of notifying a user needs to be described. How can notifications be designed so that they do not demand too much attention from a user?

An application's chosen design needs to be covered and the application needs to be evaluated. What implications does a design choice have and why has a given approach been chosen? What is best and worst with the developed application? Finally, how did the application meet the objectives?

1.6 Thesis overview

This thesis is structured in the following way. Chapter 2 introduces the basic terminology and concepts that are used in the rest of the thesis, while Chapter 3 describes related work. Chapter 4 describes the method to be applied in the thesis project to solve the problem. Chapter 5 gives details of the implementation of this solution. Chapter 6 analyzes some experiments that have been carried out to evaluate the solution and finally Chapter 7 draws some conclusions from the evaluation and suggests some future work.

2. Terminology and Concepts

2.1 Context-awareness

In this section we give an introduction to context and try to give an abstract formulation of the meaning of context that will be used throughout this thesis.

2.1.1 What is context?

Hans-W. Gellersen, Albrecht Schmidt, and Michael Beigl define context as follows:

Context is what surrounds, and in mobile and ubiquitous computing the term is primarily used in reference to the physical world that surrounds the use of a mobile device. [4, p. 2].

Context is something that can change over time and has a greater variation seen from a mobile phone than a device designed for stationary use (even if a fixed device also has context). This rather abstract definition does not tell us exactly what attributes are captured for describing the surrounding world; which is why this also has to be investigated and stated (later in this chapter). Moreover, the use of the term “context” is ambiguous. Besides meaning the actual state of the surrounding world such as the local weather, context might refer to an aspect of the situation, e.g. the speed of the wind or humidity [4, p. 2]. Sometimes context can mean a special occurrence of an event; for example, the start of winter time or the location of an event such as in the Electrum building in Kista.

How would you describe the place where you currently are? You might specify a geographical location such as a street address or you could describe what you see at this particular location, so that someone nearby could easily recognize it. The more details of the view you include, the lower the probability of confusion. There are different kinds of context, for example *global* and *local view contexts*. The Global Positioning Service (GPS) is a world-wide service with information about time and location of satellites - that enables a receiver to compute their local time and location. In contrast, a home heating system is generally only concerned with the temperature inside and outside a particular house*. *Situational context* is used to describe a particular situation that something or someone is in: a rapidly moving car, a ringing phone, a sleeping person, boiling water in a kettle, blinking lights, empty cup, etc.

*In some municipalities this system might also have to consider local laws regarding when heat/cooling has to be supplied during different periods of the year or the available heating capacity that a remote heating/cooling system can deliver, so as to distribute the available heat/cooling in a fair manner.

2.1.2 Context parameters and context sensing

As mentioned in the previous section, there are several parameters/attributes that can be used to describe something, thus application developers need to decide what is of most interest and what best describes which the target is concerned with. We can often easily measure physical magnitudes and describe them with numbers, but in most cases that might be too low a level of detail for our applications. For example, if we were to design an orientation-sensitive display such that no matter how it is held, all its content is upright and can be read by looking at it from the front, we could install sensors to learn the direction of the gravitational force on the device. After determining the display's orientation (based on the results from the sensors), we could instruct the device to rotate the picture if necessary in order that it is always displayed in the proper orientation.

Sometimes it is necessary to combine data from multiple sensors in order to derive suitable context information, while in other cases a different and perhaps smaller set of sensors might be sufficient to describe the situation so that an application can adapt to its context. Therefore one of the keys to designing a context-aware system is to think about what sensors are needed and what information on a higher abstraction level can be achieved by sensor fusion. *Multi-sensor based context-awareness* is an interesting and popular topic nowadays.

Our conclusions about why context is important are:

1. Context is essential in order for applications to adapt to their environment.
2. Adapting applications to their environment can improve human-computer interaction.

2.2 Calendaring framework

In this section the most important calendar components are investigated, a standard way of storing and representing calendar data (events, tasks, alarms, etc.) is introduced, a protocol for managing calendars from a remote machine/device is described, and finally two examples of calendar servers are given. Also several different alternatives are explored for a distributed calendaring system that supports *peer-to-peer* architectures, rather than strict client-server systems. The subsequent decision of what model to use for the prototype application will be based on the background given in this section.

2.2.1 iCalendar

In brief, iCalendar introduced a plain text file format for calendaring and scheduling information. [6, p. 1] iCalendar is sometimes referred to as iCal[†] and was defined in RFC 2445 [6], but this definition has been obsoleted by the newer RFC 5545 [7]. iCalendar files have the extension `ics` or `ifb` (`iCal`, `iFBf` for a Mac) and can consist of different calendaring components including events, to-dos, and journal entries in case of `ics`, but they can also be used to convey free/busy information in case of `ifb` (see section 2.2.1.4). Different methods can be used to process the

[†]This should not be confused with Apple's calendar client application also called "iCal".

information contained in these components for scheduling purposes. For example, an application could automatically compose messages to request an event to be scheduled, propose a date/time for an event, make a counter proposal, etc. RFC 2446 [8] describes the iCalendar Transport-Independent Interoperability Protocol (iTIP) for scheduling using such methods. You can find the Internet e-mail binding for iTIP, called iMIP, in RFC 2447[9]. In the following paragraphs some of the basic and most widely used iCalendar components are described. Please refer to RFC 5545 for other components and their details.

2.2.1.1 iCalendar components

The basic structure of all components is quite straight forward. The content information is organized in content lines and components, each line shorter than 75 octets and delimited by a line break [6, p.13]. It starts with a ‘name’ of a property followed by a semicolon-separated list of parameters followed by a colon and terminated by a line break. Every iCalendar object must contain at least one iCalendar component, the `VERSION` (2.0 is the version corresponding to RFC 5545), and the `PRODID` (identifying the product that created the iCalendar object) properties. The main iCalendar object might look like the example shown in figure 2.1.

```
BEGIN:VCALENDAR
VERSION: 2.0
PRODID: //MY Company//My product//EN
{ other iCalendar component(s) are placed here }
END:VCALENDAR
```

Figure 2.1: A “calendar file” showing the mandatory parts of a calendar object at the beginning and end of the file.

Before going into the details of other components, some of their important properties that they have in common are described. The `UID` property, for example, is contained in each `VEVENT` and `VTODO` (see sections 2.2.1.2 and 2.2.1.3) and is a globally unique identifier. It consists of the current date/time on the host where it is created; along with some other unique identifier, for example a process id, on the left hand side of an at-sign (@); while on the right hand side is the host domain name. This `UID` plays an important role for group scheduling - since its value, will be used to match requests with replies. Another common property is the date-time stamp (`DTSTAMP`) indicating when this instance of the iCalendar object was created.

2.2.1.2 VEVENT

The `VEVENT` component is a container for other components that together represent a scheduled amount of time in a calendar. Typically a `VEVENT` represents a meeting, an activity, or an anniversary. The type of event is represented by the `CATEGORIES` property. A `VEVENT` can contain a `VALARM` that is used for generating time-based alarms.

The `DTSTART` property inside a `VEVENT` specifies the inclusive start date/time of an event, but can also specify the first occurrence of recurring events from a


```
BEGIN:VEVENT
UID: 20090611T080045Z-0052@host1.com
DTSTAMP:20090901T1300Z
DTSTART:20090903T163000Z
DTEND:20090903T190000Z
SUMMARY:Plan the party on Friday
CLASS: PRIVATE
CATEGORIES: PERSONAL
TRANSP: TRANSPARENT
ORGANIZER:mailto:user1@host.com
ATTENDEE;ROLE=REQ-PARTICIPANT:mailto:user2@host.com
ATTENDEE;ROLE=OPT-PARTICIPANT:mailto:user3@host.com
END:VEVENT
```

Figure 2.2: An example of a calendar entity representing a meeting.

recurrence set. Correspondingly, `DTEND` is the optional non-inclusive ending time of the event. If not specified, the event's end is either the end of the calendar date or the date and time of day specified by `DTSTART` because `DTSTART` can have either a value type of a date or a date and time of day. Recurring events, such as a daily reminder, have the value type date for the property `DTSTART` and do not have a `DTEND`.

It is not possible to nest a `VEVENT` within another calendar component. However it is possible to refer to a `VEVENT`, `VTODO`, or `VJOURNAL` (not discussed in this thesis) component by using the `RELATED-TO` property. The following example of a `VEVENT` represents a meeting that will be transparent to searches for busy time (as indicated by the transparency property `TRANSP`), i.e. this time will be shown as free. See figure 2.2.

The `SUMMARY` property is a short summary/subject for the component. `CLASS` defines the access classification, typically having one of `PUBLIC`, `PRIVATE`, or `CONFIDENTIAL` as value. The property `CLASS` does not enforce access to a particular iCalendar component, but can be used together with another system to keep track of user rights and authentication. `CATEGORIES` is a useful property when searching for a particular type of events, such as `MEETING`, `EDUCATION`, `HOLIDAY`, etc. `TRANSP` defines if the event is transparent to busy time searches or not. Its standard values are `TRANSPARENT` and `OPAQUE`; as noted above, the first means that this time period is not seen as busy, while the later indicates that the time is seen as busy.

The `ORGANIZER` property specifies the person's e-mail address that organizes the meeting, while each `ATTENDEE` property specifies all the other participants. The `ROLE` parameter here is used in order to indicate that the first attendee is required to participate, while the second attendee is optional to come.

2.2.1.3 VTODO

`VTODO` is yet another widely used component of the iCalendar format. As the name already suggests, it is used for describing tasks that needs to be done. The list of possible properties of a `VTODO` is similar to a `VEVENT`; although a `VTODO` can include a `DUE` property containing a due date/time, a priority (`PRIORITY`) encoded

```
BEGIN:VTODO
UID:20090913T130000Z-123404@host.com
DTSTAMP:20090913T1300Z
DTSTART:20090913T133000Z
DUE:20090916T215959Z
SUMMARY:Buy a laptop
CLASS:PRIVATE
CATEGORIES:WORK,FINANCE
PRIORITY:1
STATUS:NEEDS-ACTION
END:VTODO
```

Figure 2.3: An example of a calendar entity representing a task.

as an integer and a `STATUS` property. The standard status values are `TENTATIVE`, `CONFIRMED`, and `CANCELLED`. Figure 2.3 shows an example.

2.2.1.4 VFREEBUSY

The `VFREEBUSY` component is used to specify a free or busy time interval. It can represent a request for, reply to a request, or a published set of time information. In the case of a request, the `ATTENDEE` property specifies the users whose time information is being requested, the `ORGANIZER` is the requester, the amount of time between `DTSTART` and `DTEND` is the interval of time for which the request is made, and the `DTSTAMP` and `UID` properties are there to assist in sequencing multiple free/busy requests. Figure 2.4 shows an example of a request.

```
BEGIN:VFREEBUSY
ORGANIZER:MAILTO:user1@host1.com
ATTENDEE:MAILTO:user2@host2.com
DTSTART:20071120T070000Z
DTEND:20071220T080000Z
DTSTAMP:20071120T070000Z
END:VFREEBUSY
```

Figure 2.4: An example of a calendar entity representing a request for a free/busy time window.

In case of a reply, the `ORGANIZER` is the user who originally asked for the time information, `ATTENDEE` is the user responding, `FREEBUSY` (if present) specifies the requested time information, and the `DTSTAMP` and `UID` properties have the same meaning as above. `FREEBUSY` provides a representation of time periods and can exist more than once. Figure 2.5 shows an example of a reply. When publishing free/busy time information, the meaning of all the properties is as described above.

```
BEGIN:VFREEBUSY
ORGANIZER:MAILTO:user1@host1.com
ATTENDEE:MAILTO:user2@host2.com
DTSTAMP:20071120T070000Z
FREEBUSY;VALUE=PERIOD:20071120T080000Z/PT8H30M,
20071120T160000Z/PT5H30M,20071120T190000Z/PT6H30M
URL:http://host2.com/pub/busy/jpublic-01.ifb
COMMENT:This iCalendar file contains busy time information
for the next week.
END:VFREEBUSY
```

Figure 2.5: An example of a calendar entity representing a reply to a previous request (see 2.4) for a free/busy time window.

2.2.1.5 Extending iCalendar

Non-standard properties can be added to iCalendar objects. They start with “X-” (the capital letter X and a hyphen-minus character) followed by the vendor’s id. Such a property can have any type of value (although the default type is text). The vendor’s id is included in order to ensure that fewer name collisions occur. If a user agent does not support a particular non-standard property, it can be ignored. An example of an extension for the vendor “Teccopro” would be:

```
X-TECCOPRO-ENTRANCEFEE;CONCURRENCY=SEK:50
```

2.2.1.6 Conclusion

As already seen, the format is simply plain text which makes it possible to intercept the messages and read their contents. From a security point of view, several different attacks are possible. For example, RFC 2445 states that although the **ORGANIZER** is the only person authorized to make changes to an existing **VEVENT**, **VTODO**, or **VFREEBUSY**, anyone can pretend to be organizer, then for example cancel these changes [6, p. 10]. Another security issue is that programs that are executed as part of a **VALARM** can be subject to virus or malicious attacks. In the newest version of the iCalendar specification, RFC 5545 [7, p. 148-149], Desruisseaux points out that “[...] it is up to the actual protocol specification such as iTIP [2446bis], iMIP [2447bis], and Calendaring Extensions to WebDAV (CalDAV) [RFC4791] to describe the threats [...], as well as ways in which to mitigate them. ”. Section 2.2.4 will return to these issues.

In section 2.2.1.5 it was noted that iCalendar is extensible as the format is not limited to the defined iCalendar components/properties. However, this causes problems because applications that extend iCalendar will immediately encounter compatibility problems with other applications using the same format. The result is that, when implementing or modifying a calendar program or trying to synchronize with such an application, this added functionality will not be available (examples of different calendar applications with potential compatibility problems are given in section 3.1.4 on page 36).

2.2.2 iTIP

The peers running the prototype application will need to communicate to each other in some way. The iCalendar Transport-Independent Interoperability Protocol defines one way to communicate between different systems (peers). An application that implements iTIP works with bindings for the store-and-forward transport, the real time transport, or both. For example, the protocol can be used to send calendar entries in order to publish calendaring information or request to schedule a meeting. iTIP defines the following methods:

REQUEST	is used to schedule a “calendar entry”. A request requires the recipients of the request message to respond by using the reply method. For example, a request message can contain VTODOs to be assigned to other participants by the organizer or update the status of a calendar entry.
PUBLISH	is used to publish a calendar entry for the purpose of information. No interactivity between the publisher and anyone else receiving the message with this method is required.
REPLY	is used to respond to a request. The most common way to use replies is to respond to meeting/task requests.
ADD	is used to add one or more instances to an existing “calendar entry” that specifies a recurring VEVENT, VTODo, or VJOURNAL.
CANCEL	is used to add one or more instances of an existing VEVENT, VTODo or VJOURNAL
COUNTER	is used by an attendee to propose a change in the “calendar entry”, such as to change a date.
DECLINECOUNTER	is used by the organizer to decline the counter-proposal.
REFRESH	is used by an attendee to request the latest version of a “calendar entry”.

iTIP strictly defines the organizer to be the “Calendar User” who initiates an exchange (such as scheduling a meeting) and the attendees are Calendar Users who are asked to participate. Therefore, the methods allowed to be used only by the organizer are PUBLISH, REQUEST, ADD, CANCEL, and DECLINECOUNTER. Accordingly, the rest of the methods can be used only by the attendees; except that an attendee also can use the REQUEST method when delegating [8].

There are two distinct states in iTIP: The state of an entry and the state associated with an attendee to that entry. An example of a state of an entry is

CONFIRMED as a value for the property **STATUS**. There is no default value for the “status property” and it can only be set by the organizer. An attendee controls “his/her” **parstat** (participation status) parameter in the **ATTENDEE** property which initially is set by the organizer to **NEEDS-ACTION**. For instance, if Bob is one of the attendees and he is going to attend to the meeting previously requested by Alice, he will modify his **parstat** parameter to **ACCEPTED** in the reply message. In addition, reply messages contain a status reply (specified by the **REQUEST-STATUS** property) specified as return status code in the form of two digits separated from each other by a dot (“.”). For example, the return code 2.4 means “Success, unknown non-standard property ignored”, 3.5 means “Invalid date or time”.

2.2.2.1 Revision and sequencing of components

iTIP has several mechanisms to identify the “calendar entry” that an attendee is referring to in a reply message. The **UID** property is used to match the corresponding entry in the main object (sent by the organizer), but sometimes the **SEQUENCE**[‡] number is also needed to identify a particular version of this entry (details of how sequence numbering works will be discussed later). When two (or more) components have the same **UID** and **SEQUENCE**, the **DTSTAMP** is used as the tie-breaker. The component created last, thus with the highest value of **DTSTAMP**, obsoletes all other components. To refer to an instance of a recurring component an **UID** is used together with an **RECURRENCE-ID**.

Starting at 0, the monotonically incrementing sequence number is incremented by one, each time the organizer (or an attendee, where possible) makes a “major change”. The change could for example involve adding or deleting an instance of a recurring component (by using the **ADD** or **CANCEL** methods), changing the organizer, or rescheduling an event. A change in a component’s description other than time information such as location, description, status, categories, etc., is considered a “minor change” and must **not** increment the sequence number. By definition, a component with a higher sequence number obsoletes all other revisions of the older component with lower values (and the same **UID**). This applies to requests *and* replies [8, p. 10-11, p.].

The sequence number should not be used by organizers as a mechanism for requesting a response. The parameter string **RSVP=TRUE** specified in the **ATTENDEE** property indicates that a reply is expected [7, p. 138-139].

2.2.2.2 Security considerations and conclusion

In principle, iTIP has the same security problems as already seen in the iCalendar format in section 2.2.1.6. In addition to these problems, unauthorized replacement of the organizer, eavesdropping, flooding a calendar, and replying to a refresh that is not sent by an attendee are potential security threats. There are mechanisms describing how to mitigate these threats and the reader is encouraged to read more about these issues in [8, p. 103-105].

Despite the fact that iTIP may be a quite complex protocol, applications that support RFC 2446 do not have to implement all of the defined methods and

[‡]This could be the case when a “minor change”, such as changing the meeting location, to a component description is made that by definition will not result in a different sequence number.

functions. Silverberg, et al. in [8, p. 97-101] describe a standard way to fall back along with the status code to return for applications that do not support the complete protocol.

2.2.3 iMIP

Obviously, the peers of the prototype application should be able to communicate to each other. One way of doing it is by sending email messages - this way, devices do not need to run the scheduling application to receive messages, implying that (battery driven) devices need **not** be turned on at all times (but they must be able to check for/receive new email messages from time to time). The iCalendar Message-Based Interoperability Protocol (iMIP) offers a binding from iTIP over MIME [9].

The proposed content type for iMIP messages is `text/calendar`. RFC 2447 points out that confidentiality and authentication can be achieved by using RFC 1847 that specified Security Multiparts for MIME. Both signing and encrypting could be done as stated in RFC 1847, by using two MIME headers: one for encryption and one for the control information necessary to remove the encryption. Authentication is performed with public/private key certificates.

The MIME content type header field must include the type parameter method and it must have the same value as the value of the method property within the iCalendar object. Thus multiple iCalendar objects with different methods must be encapsulated with a `multipart/mixed` MIME entity (alternatively, they could just be sent in different emails).

2.2.3.1 Security considerations

As a transport protocol, iMIP identifies and addresses most of the security threats/problems a calendaring user might experience when using iTIP. For example, spoofing of the organizer/attendee is almost impossible when using encrypted messages. However, there is no description given for a mechanism for applications to make a decision about whether a calendar user has authorized someone else to operate on his/her behalf.

2.2.4 The CalDAV protocol

The goals of this thesis project are to design, implement, and evaluate an application for scheduling events/meetings. Because the application could make use of the iCalendar format and most people have different devices to access calendar(s) and share them with others, the CalDAV protocol (see RFC 4791 [12]) must be studied. CalDAV is a standard and widely used protocol for accessing calendar data on a server. Before describing CalDAV, we will be a short introduction to WebDAV, as CalDAV is based on WebDAV.

2.2.4.1 WebDAV

WebDAV as defined in RFC 2518 [10] is an extension to the Hypertext Transfer Protocol (HTTP). WebDAV allows computer users to manage resources (files and

directories) collaboratively on (remote)[§] World Wide Web servers. For example, WebDAV can be used for storing/retrieving files on/from servers; querying for various information about the resources, such as the author, date of modification; move resources from one URI to another; protecting resources with (shared) locks; etc.

WebDAV resources are said to be part of a repository that can be identified using a single root URL. A repository does not have to have resources of a single type and should not be assumed to be part of another repository higher in the hierarchy. For example, the URL

`http://www.myhost.com/rep`

on a web server may contain WebDAV resources. However, the root URL

`http://www.myhost.com/`

need not be a WebDAV repository.

2.2.4.2 Principles of CalDAV

The Calendaring and Scheduling Consortium (CalConnect) created CalDAV. They describe CalDAV as follows:

CalDAV is a calendaring and scheduling client/server protocol designed to allow users to access calendar data on a server, and to schedule meetings with other users on that server or other servers [5].

CalDAV operates on calendar collections (a collection contains events, tasks, or other calendar components within a single calendar) and each collection or calendar object is considered a resource (see section 2.2.4.3). Because they are independent resources[¶], it is possible to perform operations on each of them individually.

As mentioned in section 2.2.1, all calendar data is stored in a file. This is why user requests for changing calendar data will logically result in requests for changes in the calendar file^{||}. The features of the WebDAV protocol, as discussed in section 2.2.4.1, allow sharing calendars via the web. Not surprisingly, this is done by giving out a URL pointing to a calendar object resource, enabling both the client and the server to uniquely identify this particular calendar resource (see figure 2.6). Another interesting feature of CalDAV is that users can edit their events, tasks, journals, etc. offline (on a hand-held device or other computer), since the whole calendar file is downloaded during a subscription. Later, the changes are synchronized with the server. However, this leads to a problem because calendars might be shared by

[§]While in most cases they may be remote, there is no real limitation to them being that, as they could also be local - the only requirement is that the server talks HTTP with the WebDAV extension.

[¶]The contents of the resources might in fact be dependent, but this is up to the applications that manipulate these resources to maintain.

^{||}Some calendar servers might store calendars in files, while others have a database to mitigate the problem of having a huge file with several years of history. However, implementing the CalDAV protocol means also supporting iCalendar which in turn means that exporting/importing of ical-files is trivial to implement.

multiple users and/or accessed from several different devices by a single user. Thus synchronizing clients must be prepared for changes in the calendar since the last synchronization. How a client learns if a calendar object has been modified, when the client attempts to submit its version of the same object, is discussed in section 2.2.4.5.

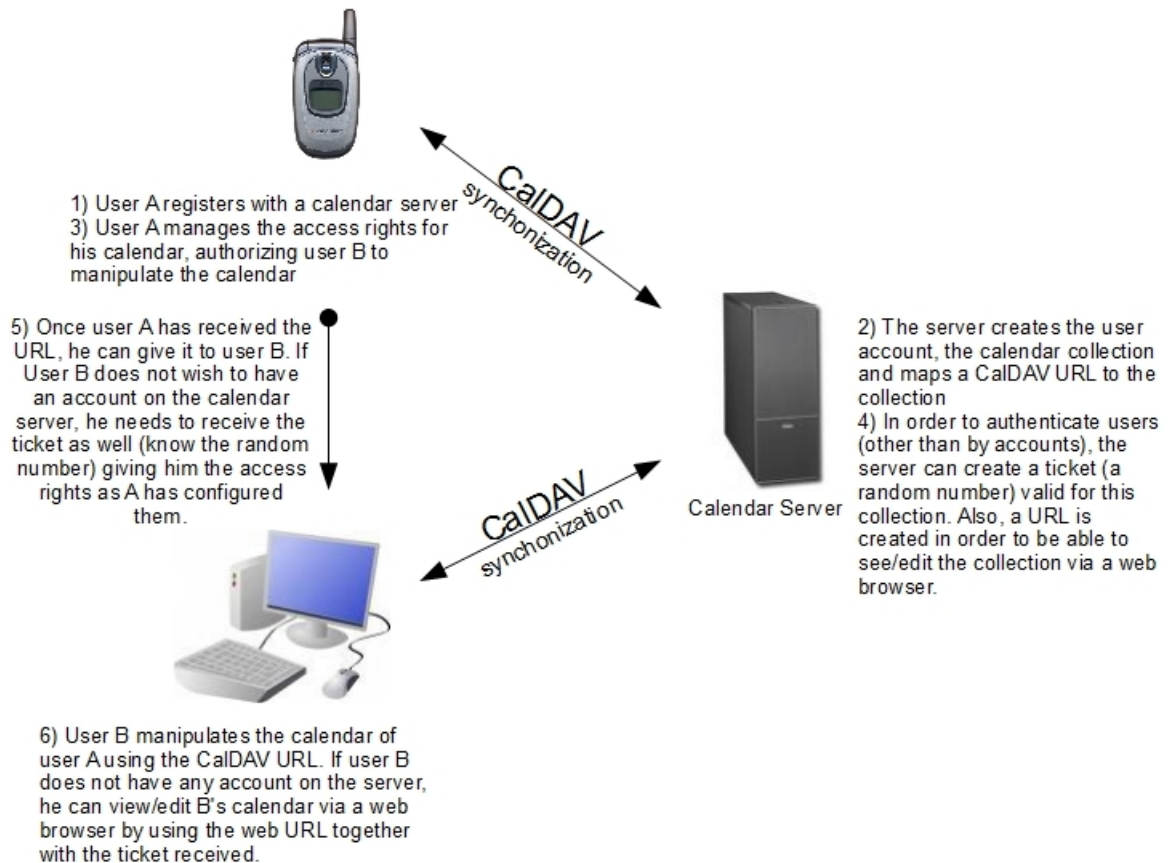


Figure 2.6: A communication example between two clients using a CalDAV calendar server.

2.2.4.3 Calendar resources

Calendar resources refers to either calendar object resources or calendar collections. Calendar collections contain only calendar data and can be created both by a client (using the `MKCALENDAR` CalDAV method in the HTTP request header) and a server (for example when a new user account is created). A client must deal with the problem of not knowing whether a URL requesting the calendar collection is or is not yet bound to any resource (this is discussed in section 2.2.4.4). Furthermore, the attempt by a client to create a calendar collection may fail because the calendar server may not support this feature or might not allow a given client to have more than one calendar. If a collection is created, a (specific) URL will be associated with it. A URL to a calendar collection or a calendar resource object need not be related to the contents in the calendar or the calendar object, thus the URL may be chosen arbitrarily (i.e. it should be a unique identifier but can be opaque). An example of a collection URL is:

```
https://www.myhost.com/rep/calendars/user1/
```

Example of a calendar resource object URL is:

```
https://www.myhost.com/rep/calendars/user1/kjh783.ics
```

Calendar object resources contain a single calendar component (such as `VEVENT`, or `VTODO`), with the exception of a `VTIMEZONE` component, and do not have any iCalendar method property specified. Calendar objects with different UIDs must be stored in different calendars.

2.2.4.4 Creating calendar resources

A new calendar collection resource is created by using the `MKCALENDAR` method inside a HTTP request header. The human readable calendar name is specified in the `DAV:displayname` property and should not be empty (See figure 2.7). `C:calendar-description` specifies the collection description.

`C:supported-calendar-component-set` is used to restrict the component types that a particular collection may have. The value of a such property is often initialized when a client creates a new calendar collection. By default, a collection is assumed to support both `VEVENT` and `VTODO` (and implicitly also `VTIMEZONE`). Thus *Alice's Calendar*, shown in example 2.7, may have calendar object resources that contain only `VTIMEZONE` components.

```
MKCALENDAR /rep/calendars/alice/ HTTP/1.1
Host: www.myhost.com
Content-Type: application/xml; charset='utf-8'
Content-Length: xyz

<?xml version='1.0' encoding='utf-8' ?>
<C:mkcalendar xmlns:D='DAV:' xmlns:C='urn:ietf:params:xml:ns:caldav'>
  <D:set>
    <D:prop>
      <D:displayname>Alice's Calendar</D:displayname>
      <C:calendar-description xml:lang=en'>
        Personal Calendar.
      </C:calendar-description>
      <C:supported-calendar-component-set>
        <C:comp name='VEVENT' />
      </C:supported-calendar-component-set>
      <C:calendar-timezone>
        <![CDATA[
          CALENDAR DATA PLACED HERE
        >]]>
      </C:calendar-timezone>
    </D:prop>
  </D:set>
</C:mkcalendar>
```

Figure 2.7: Example of a client request for creating a new calendar supporting components that contain only VTIMEZONE components.

As mentioned in above, a new calendar object resource is created by using the PUT method inside a HTTP request header, specifying a target URL such as:

```
http://www.my_host.com/caldav/user1/uygfd83.ifb
```

and attaching the calendar object in the HTTP body. However, if the given URL is already mapped to a calendar object resource, then the existing object will be overwritten by the submitted object. Because a client does not want to examine a collection's URLs to see what URL are already bound, to avoid naming collisions, an additional request header can be used.

By specifying `If-None-Match: * target_URL` (where `target_URL` is the URL where the client wants his resource to reside), the client ensures that existing resources will not be overwritten. See figure 2.8.

```
PUT /rep/calendars/alice/idsknf39280.ics HTTP/1.1
If-None-Match: *
Host: www.myhost.com
Content-Type: text/calendar
Content-Length: xyz

BEGIN:VCALENDAR
VERSION:2.0
BEGIN:VEVENT
UID:20090715T162145Z4332101@domain.com
DTSTAMP:20090712T182145Z
DTSTART:20090714T170000Z
DTEND:20090715T040000Z
SUMMARY:Party
END:VEVENT
END:VCALENDAR
```

Figure 2.8: Example of a client request for creating a new event.

2.2.4.5 Calendar object resource entity tag

The entity tag (ETag) is a property of the CalDAV protocol used by clients to detect whether a particular object has been changed between the current access to the object and the time when this client last accessed it. The ETag is a number returned by the server, after the client has created the object (using the PUT method). A client can also get the ETag of a stored calendar object by specifically asking the server for it (with the PROPFIND and the GET method) and compare the last seen ETag to the value returned by the server. If they are equal, then the object has not been modified. Otherwise, a client should retrieve the modified calendar object before making further changes - rather than using the “outdated” version of the calendar object.

The key idea underlying ETags is defined in RFC 2616 [16, p. 29]. There are two different versions of ETags: weak and strong. Two strong entity tags are equal only if the corresponding resources are equal octet by octet, while a weak entity tag may be shared by two equivalent resources (i.e. two files containing the same source code but different comments). Servers implementing CalDAV must set strong entity tags on all calendar object resources in a response to a GET and PUT requests.

2.2.4.6 Access control

Data that is on the web must be protected from unauthorized users and the WebDAV Access Control Protocol (WebDAV ACL) (see RFC 3744 [15]) defines a set of methods for doing this. Some privileges of the WebDAV ACL are: read, write, unlock, bind, unbind, etc. Each privilege can be specified on a WebDAV collection or any WebDAV resource individually. Every CalDAV server must implement the WebDAV ACL and support Transport Layer Security. In addition, each calendaring server must implement the CALDAV:read-free-busy calendaring privilege. If granted, this privilege allows a user to only read the time when the

desired calendar's user is busy, but may not read other event information such as summary, attendees, etc.

There is an additional property to easily find a user's calendar collection set. Typically users will group all their collections under a common collection; the URL of this common collection can be set by the `CALDAV:calendar-home-set` property.

2.3 Location

As a user's location will be included in the scheduling system, there is need to discuss how to represent location and what the requirements of location determination services are. Users need to trust the application, so whenever there is an opportunity for people to meet because they are in the same area (what that means is yet to be discussed) at the same time, the scheduling application should be able to detect this co-location and inform the users of this opportunity for a meeting. Thus, the location service is required to be operational all of the time, in order to provide real-time location information.

Because sending, receiving or otherwise using location information of an object is a privacy issue, we have studied Geopriv in section 2.3.2. Geopriv is a working group that wants to make sure, that no privacy issues are left unaddressed related to location. In section 2.3.5, we mention a few words about location in calendars and how location could be recorded in a calendar. Finally, different techniques for measuring location are presented.

2.3.1 Detecting nearness and co-location

What does it mean to “be near” someone? For example, when people are sitting in a meeting room, they are probably sitting next to each other and depending on the area of the meeting room, the maximum distance between any two persons is the length of the longest diagonal in the room. Another situation is when someone is asking someone how to find a particular object in a city or when a group of people are walking together along a street or in a park **and** talking. In these cases, the distance between people is rather small, of the order of some few meters. As a counter example, consider two people that have their homes in different cities. Maybe they would like to meet when both of them are in the same city, thus “near” would in this case mean a whole city's diameter!

Clearly, it is hard to set an efficient static distance for all groups of people meeting. The distance considered highly depends on where people usually reside, when they stay at a particular place and for how long. One can think of a mean distance between a pair of individuals as a function of time predicting the user's future location. This function could be used in order to suggest a meeting place both for *ad hoc* and planned meetings. Suggesting a co-located meeting place could be a research area on its own and, unfortunately, does not fit into this thesis. Section 7.1.1 is related to this topic.

Note that mobile phones could learn about two users being near each other by listening for mobile devices in promiscuous mode. Both bluetooth and wireless local area network, when used promiscuous mode, do not depend on any network architecture and can be used beneficially when detecting nearness. When a mobile

device's microphone can hear voices of other people then that also means that these people are nearby (although a voice recognition algorithm is needed in order to identify these people, while, when listening for Media Access Control addresses, a simple string comparison against known Media Access Control addresses would do the recognition).

When participating in a meeting by phone, a meeting does not have to be at fixed location. Similarly, consider the case when two or more employees are taking a flight from Stockholm to Berlin - even though they are moving, they are in the same plane. If they could arrange that they sit next to each other, they might be able to conduct a meeting on the way. The reader might note that it is important to explore the conditions necessary for a user to interact with others. When generalizing the user's location to periods of time, the path that the user is traveling and the type of vehicle that is used must be considered. Valuable information also includes what types of communication possibilities exist when the user is traveling and who is traveling with him/her (along the same path, or in the same vehicle).

2.3.2 Geopriv

Geopriv is a "Geographic Location/Privacy" working group and part of the Internet Engineering Task Force. As the name suggests, Geopriv is concerned with developing mechanisms/protocols for representation for and transmission of so called Location Objects that contain location information. Because many application today are in need for location information, it is important for Geopriv to analyze the authorization, privacy requirements, and the integrity requirements that must be met when such representations of locations are created. The working group wants to make sure that organizations that are building applications that store, create, or use location information in any other way, are working according to the requirements of Geopriv and that no "additional security or privacy issues related to location are left unaddressed [13]". As a result, a suite of location aware protocols have been designed.

The key document "Geopriv Requirements", RFC 3693 [14], states that the privacy of an entity defining the privacy rules has to be protected. These rules have to be followed by any other entity that has access to, computes, or derives a target's location information, and if these other entities can be trusted to both know and follow the relevant privacy rules. The main concerns of RFC 3693 are:

- Security of the transmission of Location Objects
- The "Rule Maker" who creates the Privacy Rules
- Filtering of Location Information, i.e. reducing the location precision
- Expressing and including Privacy Rules in Location Objects
- Decoupling of Personal Identifiable Information from location information
- Preserving a user's anonymity by protecting the user from entities participating in the location-aware protocol.

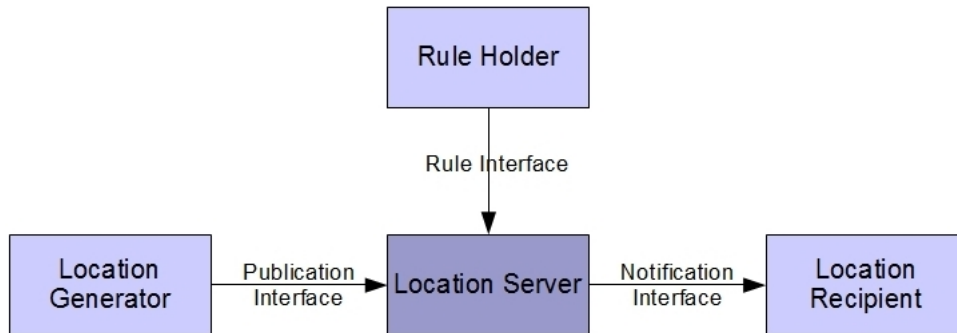


Figure 2.9: The four primary Geopriv Entities and their interaction, adapted from RFC 3693 [14].

To guide Geopriv in their work, a list of entities involved in location-aware protocols has been enumerated (see figure 2.9). The location issues Privacy Rules for receiving, filtering, and distributing location information. The location can be queried by the Location Server to retrieve a set of rules or the location can push the rules to the Location Server. As mentioned previously, the Rule Maker populates the rules (by placing them in the location). The Location Server receives publications of Location Objects from Location Generators and may receive subscriptions from Location Recipients. The Location Server must apply the rules (learned from the location) to Location Objects before it notifies the Location Recipients. Subsequently, the Location Recipient receives Location Objects via the the notification interface. The Location Generator determines the location information of the target, creates Location Objects and publishes them to the Location Server. Note that the interfaces are not necessarily protocol interfaces, but could be inside one device. In addition, the Location Generator, Rule Holder, and Location Server might be implemented in the same device (for example a mobile phone).

2.3.2.1 Geopriv entities in our application

It can be hard to follow the roles of different abstract entities without a concrete example. In our prototype scheduling application, location of user's will play an important role for suggesting a meeting place. In order to locate a user, the location could be acquired by this user's mobile device, when requested to do so by the organizer. Hence, the rule maker is the user who's location is requested. Such users should put their rules (telling what the organizer is allowed to do with the received location) together with their location information, so that these rules are known by the organizer.

Depending on, if a user stores his/her location in his/her device or not, the device could act as a "location server", thus allowing other users to query it. Even if "the target" in most cases will be the device, we will be talking about the location of the user carrying the device. The user's device is also the location generator in the case GPS (see section 2.3.6.2) is used, since the device then can calculate its location by itself after that it has collected the satellite data. If GSM positioning is used (section 2.3.6.1) and the devices location is calculated by the mobile network

operator (and then sent to the device as a location object - this is called *indirect remote-positioning*, see section 2.3.6.1 for other types of positioning techniques), then the network operator is the location generator. In our program, the user requesting the location information of another user is the location recipient (the organizer could be the location recipient).

2.3.3 Demands on positioning systems

Obviously, for scheduling purposes, a system that gives our position with a maximal error of one meter is not needed. Instead, tolerating an error of dozens of meters would be sufficiently accurate, because it is still possible to coordinate the meeting of two individuals. It is often sufficient for people who want to meet to know that they are for example in the same building, on the same street, same district, etc., i.e. the maximum current distance d of any 2 attendees lies within X meters. The value of X that should be used is highly dependent on the involved people's usual tasks, i.e. the location of the tasks they usually perform. For example, a person having his office at location A and his/hers home at location B, would during the working time normally be at location A, otherwise they are most likely to be at location B.

Because people meet at different geographical places, it is not enough for the application to know only the name of some local entity (such as a street's name or a building's floor and room number) as a meeting location, as there is a need for additional information - since there may be (globally) many streets with this same name and many buildings with the same floor and room numbers. Geographical coordinates seem to be more appropriate for our purpose. Having this kind of location representation, the application will not only be able to suggest a meeting place (either geographically centralized or a location near some of the participants), but also be able to give a warning to someone who is supposed to be on their way to a meeting, but is not yet there (or alternatively is not going to make it in time) for the meeting.

2.3.4 Location representation

Latitude and longitude are geographical coordinates measured in degrees to represent a point on earth's surface. The equator is a line of latitude and defines 0 degrees of latitude. In the earth's Northern Hemisphere, the latitude has a positive degree value and can be at most 90, at the North Pole. Correspondingly, the south pole is at -90 degrees. More precisely, the latitude value is *the angle at the center of the earth between a point on the earth's surface and the equator plane*. In contrast to longitudes, all latitude lines are parallel.

All longitudes, also called meridians, run perpendicular to the equator and pass through both poles. As there is no obvious "zero-meridian", the Prime Meridian line going through Greenwich, England, has been given a zero value to by international conventions. There are 180 degrees of longitude in east and west directions, with westwards longitudes having negative values [23]. Thus, longitudes represent *the angle between east or west of the north-south-pole meridian and another meridian that passes through an arbitrary point*.

Both coordinates can further be subdivided into minutes (') and seconds ("). A degree has 60 minutes and a minute 60 seconds (an example of a latitude in this

form is $-34^{\circ}25'16''$). Alternatively, degrees can be represented as decimals** (for example $-34^{\circ}25'16''$ corresponds to -34.421111). Using at least 6 decimals allow for accuracy to within one meter of geographical position [7, p. 86].

2.3.4.1 Distance calculation

How to compute the distance between two points expressed in latitudes and longitudes? There are many ways how to do that and each way is based on a model. For example, one naive method for distance calculation is the Pythagorean distance equation

$$\Delta d = r \sqrt{(\Delta\phi)^2 + (\Delta\lambda)^2}$$

r is substituted by the earth radius, $\Delta\phi$ and $\Delta\lambda$ by the difference of the points latitudes and longitudes, respectively. However Pythagorean distance formula works for coordinate systems with constant unit length - so in the case of geographical coordinates, because the length of one degree of longitude varies depending on the latitude, the Pythagorean distance is inappropriate for the distance calculations. Additionally, this method would just give the distance between two points measured on a straight line.

The geographical distance can be calculated using a formula for the exact distance between two points on a circle's surface. Assuming that the Earth is a sphere, the *great circle distance* method could be used. The main idea behind it is the concept of great circles. A great circle has the same center point as the sphere and it cuts the sphere into two equal halves. If two points on a sphere's surface are not directly opposite to each other, a unique great circle exists (passing through both points). The inter point distance, i.e. *the great circle distance*, is defined as the length of this circle's arc going between the two points [11]. If (ϕ_1, λ_1) and (ϕ_2, λ_2) are the coordinates for the starting and the destination point, respectively, the spherical angular distance $\hat{\sigma}$ is:

$$\hat{\sigma} = r * \arccos(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\Delta\lambda))$$

The Earth can be approximated as a sphere with a radius $r = 6371.01$ km (although the Earth is more like a spheroid^{††}) with extreme values for the radius of 6378.137 km at the equator and 6356.752 km at the poles. There is another slightly more complex formula for calculating the distance between two surface points of a spheroid referred to as *Vincenty's formula*. The method is based on an iterative approach where the programmer (up to some point) can decide how accurate a solution should returned. In principle, the longer the program runs the greater the accuracy of the returned value. The details of the formula will not be presented in this thesis as they are rather extensive. However, the reader can refer to [22].

**It is possible to represent geo-location in iCalendar by specifying the value pair latitude longitude after the GEO property. For example: GEO:-67.836152;-140.840214. The geo-location property can be part of VEVENT or VTODO calendar components.

††A spheroid is obtained by rotating an ellipse along one of its axes.

2.3.5 Location in calendar events

Some events are predictable because they are recurring (even if not explicitly represented as recurring events in a personal calendar), other events are unusual, but at the same time have a higher risk of being forgotten and thus a higher probability of corresponding to an entry in a calendar. Therefore, a highly desirable feature in this thesis' application is exploiting the location information already present in a calendar.

2.3.5.1 Future events and planned events

From our practical personal experience we already know that it is common to specify a location for an event. Provided that the location is globally known, the application could use this information when selecting a good meeting place. For example, the application can learn that in the free time between two events with different locations, the person at some time *will* be on the way to the later event. If both events are at the same location and there is a short unscheduled time between them, the person *will* be likely to be at this same location for the start of the second event. More interesting for the application is that the person probably *will not* change his/her location during a short time, thus this location will be a potential meeting place for meetings with others. When a user's calendar for a particular day has a single event or events consecutively following after each other, i.e. with little or no free time in between, then a meeting could be scheduled either before or after this/these event(s). In both cases, we also can take advantage of the location being already present in the first/last event. Note that "short time" is relative to the individual's personal scheduling preferences and not an absolute short duration of time (i.e. 1 minute).

2.3.6 Positioning techniques and services

There are different approaches to determine a device's location. Depending on the network the device is a part of, the infrastructure could collect information about signal strength and by knowing the propagation properties of the signal it could estimate the location of the device. This can be done the case of wireless Local Area Network equipped devices. Alternatively, the device could gather enough information about transmitters nearby to calculate its position by itself. The Global Positioning Service (GPS) will be discussed in section 2.3.6.2. The next section discusses positioning systems used with cellular phones.

2.3.6.1 GSM positioning

Since 2001 it is possible for operators to accurately locate mobile phones in the Global System for Mobile Communications (GSM) in order to give this information to an emergency assistance service [24]. This is an important service because many calls to an emergency service are made from mobile phones and often the callers can not provide accurate location information by themselves. As a result, in a number of countries it is required that the mobile operator be able to locate the phone and provide this location to emergency services (and depending upon the regulations potentially to others). However, this is not the only reason why locating mobile

callers is useful. Examples of additional applications include location sensitive billing, providing the mobile with Internet settings for a particular country, or using the location information to provide another location based service (see section 3.2.2 on page 38 for another example).

There are many techniques (such as propagation time, time advance, Time Difference of Arrival, and Angle of Arrival [17]) that can be used (by themselves or in combinations) for locating a mobile device, but the details of these methods are outside the scope of this report. We can classify different positioning systems into classes based on where the location information is measured and where it is used. This leads to three classes: *self-positioning*, *remote positioning*, and a mix - *indirect positioning*. In a self-positioning system the mobile device listens for geographically distributed transmitters, then calculates its position autonomously. In remote positioning signals originating from the object to be positioned are received at multiple receivers and transmitted to some kind of central unit to combine the information and to calculate the device's position. Indirect positioning means that the position determined by a remote positioning system is sent to the located object (this is called *indirect self-positioning*) or vice versa (*indirect remote positioning*).

Mobile-based positioning is a form of self-positioning in GSM networks. Similarly, the Global Positioning Service (GPS) can also be used for self-positioning. Advantages of self-positioning are that it preserves the privacy of the device (user) location and this approach is highly scalable (as each device performs position by itself). A major disadvantage of self-positioning is that the device has to collect and process all of the information itself (as this may require considerable time or energy or both).

GSM systems can also do remote positioning using the fact that the base station keeps track of the mobile device based upon reports that the mobile device sends periodically (approximately twice per second). The base station also knows which antenna (in a multi-sectored cell) is being used for this mobile. The base station also estimates the rough distance to the mobile, as it needs to maintain a parameter known as "time advance" - this parameter is used by the mobile to adjust the time when it transmits so that its signal will arrive at the base station in its time slot (as GSM uses a time slotted media access and control protocol). Knowing the antenna and time advance enables the base station to estimate the mobile's location as within a sector of a cylinder. However, this area can be quite large - especially in rural areas where the density of base stations is low.

Although in this section we were talking about cellular phones as mobile units and their cell towers as base stations with antennas, a similar approach can be used for any wireless network.

2.3.6.2 The Global Positioning System - GPS

There are many different systems for location sensing and GPS is probably one of the most well known and widely utilized methods. It is a satellite-based system that provides positioning and timing information anywhere in the world. GPS provides two levels of positioning: the Precise Positioning Service and the Standard Positioning Service [25, p. 9]. The non-military standard service is free of charge and can be used by anyone. However, its signal is very weak and can hardly (or not at all) be received through walls, which makes it hard to use the service indoors.

Because signals from several satellites are needed by a GPS receiver in order for the receiver to work out its position, other obstacles such as trees and buildings degrade GPS positioning even outdoors. However, these problems have not stopped GPS from being used, this is evident from the very large numbers of vehicles equipped with GPS receivers and vehicle navigation systems - as well as the very large numbers of hikers and others who use hand-held GPS receivers. Another GPS disadvantage in mobile devices is that the battery consumption is high compared to GSM.

For use indoors, the GPS signal can be received by an antenna placed where it has line of sight to the satellites, and then retransmitted at a higher transmission strength inside buildings thus using repeaters to penetrate walls. Or the GPS signal could be sent by the same antenna through a data link to devices attached to this link (providing indirect GPS coverage). Additionally, it is possible to place pseudolites indoors - these devices transmit a signal just like a GPS satellite, but located on earth rather than in space.

3. Related Work

3.1 Calendaring software

In section 1.1 we mentioned that there are many calendar servers, each implementing different functionality. In this section we give a brief introduction to two cross-platform calendaring server solutions: Chandler and Bedework. Following this, we introduce other popular server-based calendaring applications in section 3.1.4. We have examined various calendaring systems in order to learn what features for calendar sharing exists, as we should consider these methods when devising and evaluating our proposed solution. We also want to learn more about how to synchronize calendars between different devices, because we expect a typical user to have multiple devices that they use to access their calendars. Also we want to see how much can be done through the web interface, because users might visit their calendar from a device that does not have a calendar client application installed, but does have a web browser. If all of the operations can be done via a web page, then the user can access the calendar via their web browser; hence the user may not need to install any software on their device.

3.1.1 Chandler

The Chandler [17] cross-platform calendaring client and its server-bundle are open-source and licensed under the Apache Software License, version 2.0. Although Cosmo, the Chandler Server, has full support for its tightly-coupled desktop client: Chandler (not discussed in this thesis), the calendaring server is not limited to being used together with Chandler, because Cosmo supports the CalDAV protocol. The server bundle is composed of several different components. The major components are a Derby database, Tomcat servlet engine, MySQL Connector/J, and the Chandler Server webapp [18]. Various settings can be made within the server-bundle, for scaling purposes and/or connecting to a different database than the embedded database. However, we will not go into these details in this thesis, the interest reader can find these details in [18].

The initial server setup is rather easy: once the Java Runtime Environment is installed, the server can be started either in debug or normal mode. By default, the server is accessed from a web browser using the URL `http://localhost:8080`. New users can sign up via the web interface. There is a separate web page for administration of user accounts and checking the memory usage of the server. This administration page is accessible once the user is logged in with the administrator's password. For a non-administrator the first page is as shown in figure 3.1.

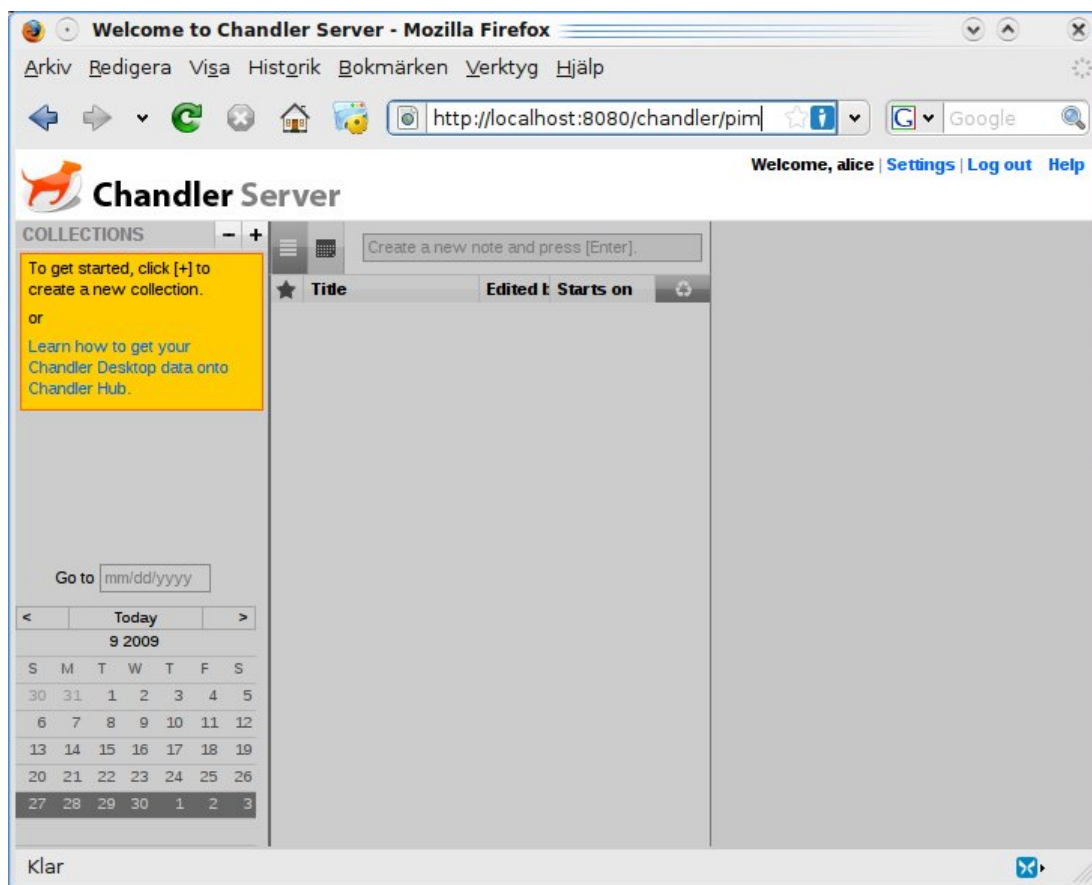


Figure 3.1: The ‘home’ page of a Chandler user

3.1.1.1 Chandler basics

Chandler adapts the collection concept of the CalDAV protocol. In Chandler a collection means a calendar collection, i.e. all events and tasks that are associated with this calendar are called a collection. The user can create several collections for what ever purposes the user desires and give each of these collections a specific name. Once this is done, it is possible to switch between a task and an event view combined with the collection to be shown. Creating events, tasks, and switching between different weeks in the calendar is rather straight forward, although it is not possible to view a whole month at a time. Unfortunately, the settings for the time zone are not saved and must be entered manually each time.

In the settings the user can include his/her name, password, and email address. Additionally, the user can delete his own account by himself. Once access to a so called Account Browser is turned on, the user can use the Account Browser to get an overview of all (of his/her) collections: including those created, submitted to, or shared with others. This is the place where (access control) tickets* can be

*A ticket is basically a pseudo-random number generated by the server. Appending a (valid) ticket to a URL submitted to a server will grant permissions that the ticket has been configured to give. In addition, in case of a calendar server, a ticket is often associated with only one collection and could be time constrained.

granted/revoked by the user thus offering basic calendar sharing options - such as granting read, write, or seeing free/busy rights.

What is unique in Chandler is that it can list all the added tasks/events so that the user can easily give them different priorities/status. Each task/event can be labeled as to do “now” or “later”. If an task/event has already been done/occurred it can be marked as “done”. However, in Chandler there is no difference between events and tasks. Hence the interface offers meaningless operation as there is no point in manually setting a meeting’s ‘chandler status’ to done if the meeting has not occurred yet. Unfortunately, tasks can not be added via the web interface. Figure 3.2 shows what the “Chandler status” display looks like.

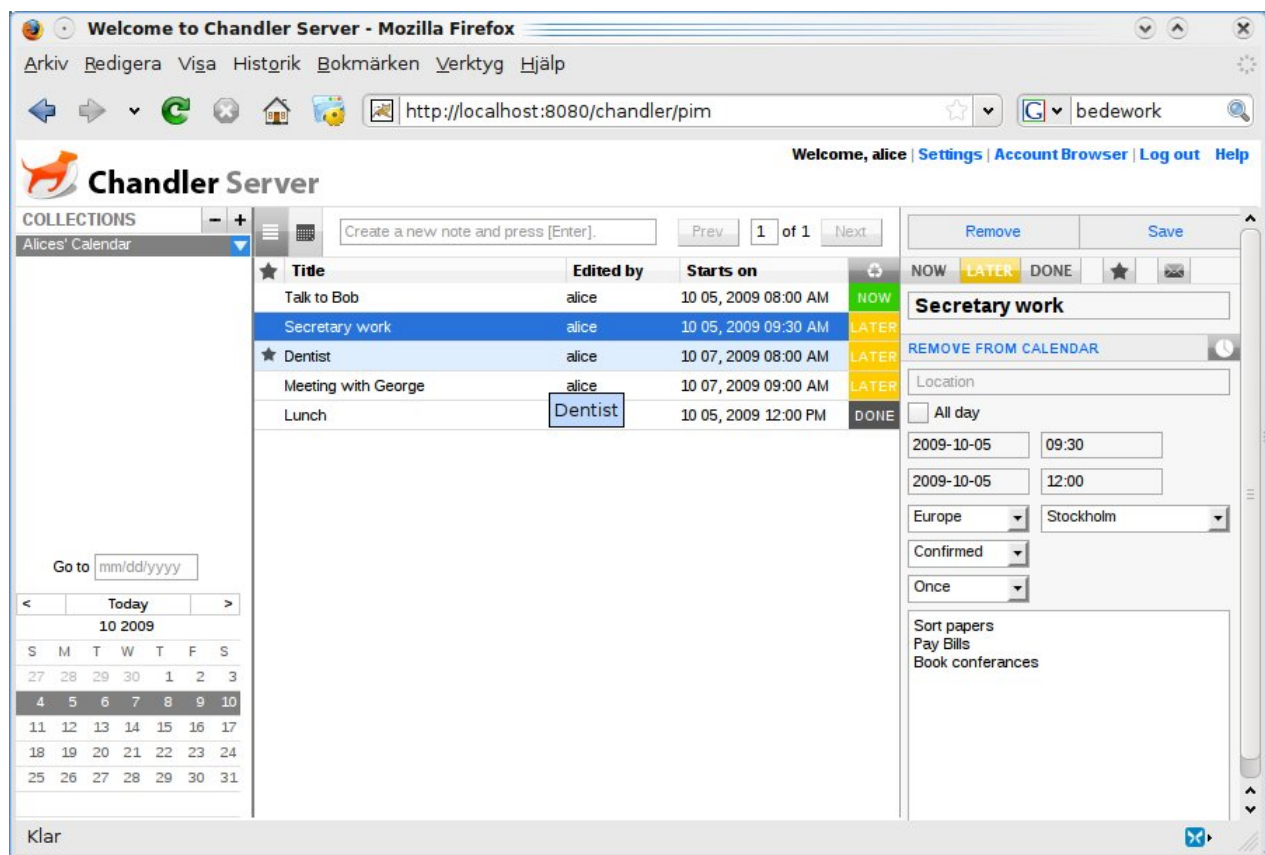


Figure 3.2: It is possible to see additional events in a list and to set the “status” of an event to now, done, or later.

That Chandler does not distinguish between an event and a task may be seen as a disadvantage by people working with traditional calendaring programs. However, when people collaborate in order to achieve a common goal, they often correlate meetings with tasks and that is why an “event-task-hybrid” is not such a bad idea.

3.1.1.2 Collaboration in chandler

Each collection can be published on the web with the ticket function described above. The ticket is basically an URL with a random number of sufficient size that it can be given to trusted users, so that they will be able to interact with the Cosmo

Server through a web interface. Similarly, a CalDAV URL can be given out for a subscription.

The Chandler Server does not differentiate between a meeting or any other type of event, therefore it has no options for scheduling meetings. As a result, it is not possible to invite or select participants for a meeting. The only feature for collaboration is that there is a mail symbol for sending an email when creating events. Clicking on it will open the default mail program and insert some event-related data about the event to be scheduled into the mail message.

3.1.2 Bedework

Bedework is another open source cross-platform calendaring server. It is also packaged together with many components. In comparison to Chandler, it offers some additional functionality for group servers (for example Lightweight Directory Access Protocol directory support, managing of user groups and rights, public event submission, etc.) because it supports both CalDAV and Storage of Groupware Objects in WebDAV [19]. It is relatively easy to set up the server and by default Bedework can be accessed via a URL (just like Chandler).

The calendaring functionality is a bit more advanced than Chandler, although the graphical user interface seems more lightweight. First of all, events and tasks are different entities in Bedework. Next, the user can select between a daily, weekly, or monthly view or display (some part of) the events in a list. Finally, unlike Chandler, the settings for the time zone, working days, and some other event-related settings are maintained once they have been saved. Except for the user's email address, all other user-related settings in Bedework must be managed via a command line interface or placed directly in the Lightweight Directory Access Protocol database.

3.1.2.1 Bedework's event properties

Bedework includes location settings. New locations can be added in a separate settings tab. These location can later be used when creating new events. Section 2.2.1 showed that some of the properties of an event in addition to location are categories. In Bedework categories can, like locations, be added and later used. Moreover, Bedework's users can provide each event with a link and specify per event how events are to be treated by a free/busy request (i.e. shown as opaque or not).

3.1.2.2 Collaboration in bedework

The scheduling functions in Bedework are traditional: After adding participants (required or not) it is possible to see all of their free/busy information in a single view (see figure 3.3). By default, the invitees are sent invitations to their 'Inbox' and, unless otherwise specified, the invitees need to answer these meeting requests (this corresponds to the event status 'needs action').

Bedework has rather complex and not necessarily useful access control options. For example, when user A is granted the privilege to write to user B's calendar, user B will not be able to see those events because he is not their owner (and the owner has all the permissions). B must explicitly grant himself the right to read what A

posted into his calendar. On the other hand, users can be grouped together and the access rights can be managed on the group level. There are also predefined groups of users: owner, (un)authenticated, and everyone. Authenticated users can always read the free/busy information of any other user and send them scheduling requests.

Schedule Meeting or Task continue

Add attendees

Role: Status:

Attendees

attendee	role	status
mailto:admin@mysite.edu	required participant	needs action
mailto:vbede@mysite.edu	chair	accepted

Freebusy for all attendees

	AM								PM							
	12	12	12	12	12	12	2	4	6	8	10	12	2	4		
10-1																
10-2																
10-3																
10-4																

2009-10-1
7:00 AM
all free

Figure 3.3: The free/busy view of all participants in Bedework

3.1.3 Peer-to-peer calendaring

So far applications have been discussed that implement the CalDAV server-client protocol for managing a calendar over the network. Tungle (<http://www.tungle.com>) is a peer-to-peer calendar *accelerator* that integrates together with Outlook, Google, and iCal Calendars [21] (these calendar alternatives are further described in section 3.1.4).

Tungle allows easy scheduling of meetings among “peers” where the organizer can view all of the participant’s calendars, much as in Bedework, but without a group server[†]. Not only is this solution scalable, but it also supports many different calendars. The only requirement for Tungle to work is an existing calendar, a web browser, and a mail account. Not all participants of a meeting are required to have a Tungle account, because invitations are send and received by e-mail.

Tungle is free of charge, therefore the Tungle company is not liable for any errors in the scheduling process nor can it guarantee that the scheduling information is shared without disruptions. Tungle can not be used for commercial purposes, including offering Tungle as part of commercial offering. However, Tungle offers commercial service with a different set of terms. More about Tungle license terms can be found on their homepage.[‡]

[†]Note that by a group server we mean a server where (among other data) everyone’s calendar data is stored. Tangle is a peer-to-peer application because it interconnects calendars of different flavors stored on different servers.

[‡]www.tungle.com.

3.1.3.1 How meetings are booked in Tungle

A person having a Tungle account, can publish via a web-page available and by him/her chosen times. The web-page is called Tungle.me and is also an interface for users who want to book the meeting with the person publishing his/her free times. A person without a Tungle account can select several on the web page published time slots that work for him/her and fill in other meeting details. After that, the publishing user will receive a meeting invitation and can via a web view see all the proposed times overlayed on his/her calendar. That user selects one slot and then the meeting is added to his/her calendar and the organizing person receives a confirmation e-mail. Because users can synchronize the Tungle.me page with their calendars, times that they choose to be available, are automatically removed from the web view when booked.

A person with a Tungle account can also create a meeting with several attendees that do not have a Tungle account. In order to do that, meeting details are filled in on the web and the organizing person selects time slots that work for him/her. After that, the attendees will be sent an invitation e-mail with a link to a page where they may select time slots that work for them. When everyone has done that, the organizer can from a web view select the meeting time by himself/herself, or configure Tungle to choose the first common time slot available. After that the meeting time is selected, the meeting is added to the organizer's calendar and a confirmation e-mail is sent to all attendees.

Tungle users can share calendars among each other. The sharing person can specify if he/she wants to share free/busy information only, or if meeting details (subjects) also are shown. Tungle users can edit their sharing settings or stop sharing calendars at any time. Booking a meeting among Tungle users works in the similar way as described above with the addition, that everyone gets the meeting object added to their respective calendar, once scheduled. Another advantage for Tungle users booking a meeting is that the organizer can see sharing users availability overlayed on his/her calendar, thus it is easier for the organizer to select commonly available time slots.

3.1.3.2 Conclusion

Tungle is a nice tool for interconnecting different calendar systems and easy to use. Tungle allows users to share calendars by a person by person basis, although a Tungle account is required to do that. It may be powerful that users without a Tungle account can book a meeting with someone they know having one, but booking in this way only allows exactly two participants. If more than two people want to meet, the organizer must be a person with a Tungle account.

An organizer creating a meeting from his/her Tungle account may find that it convenient to select what times he/she would like the meeting to be scheduled at. However, this might also be a problem, because the selected time slots may not be sufficiently many in order for non-Tungle users to find a suitable time slot. Even if there might be sufficiently many time slots originally proposed by the organizer, there is no guarantee that everyone will select the same time slot. Once people have agreed that they want to meet, it is more important for the meeting to occur at any time and not on a specific time slot proposed by the organizer. Selecting availability

for a meeting may compromise its scheduling.

3.1.4 Other calendar alternatives

There are many different calendar applications. Here we consider Yahoo!, Google Calendar, and Microsoft's Outlook Calendar as three examples of widely used calendar applications. Both Yahoo! and Google Calendar support a read only web view of a calendar for non-registered users. For viewing/editing a Yahoo! calendar, the URI is of the form:

```
http://calendar.yahoo.com/userID@mailDomain.com
```

where userID is the user ID and the mail domain can be one of the following: ymail.com, yahoo.com, or rocketmail.com. Google Calendars can be viewed using a URI:

```
http://www.google.com/calendar/embed?src=userID
```

where user ID is the email address of the targeted person. Often an additional parameter is passed in that URI indicating the time zone. Both companies also offer advanced alerts to remind users of their events via email, messenger (in case of Yahoo!), or Short Message Service (SMS). Although Yahoo and Google implement CalDAV, the only supported client applications are Apple's iCal and Mozilla's Lightning[§]. The iCalendar format ics itself is extensible (and has been extended) so all the custom components together with their properties must be taken into consideration. A nice feature of the Google Calendar is that it can show holidays for your country. An application called "Calendar" that runs on Android devices makes it possible to synchronize these devices with Google Calendar.

As part of the Microsoft Office 2007 package, Outlook integrates with the Microsoft Exchange Server. Because of this integration with Exchange Server, besides a calendar, Outlook offers a lot of additional functionality [20]. The calendaring functionality is similar to Yahoo's or Google's, but Outlook is only able to communicate/synchronize with the Exchange Server using the Microsoft protocol, while both Google and Yahoo are working to make it possible to synchronize their calendars with the Outlook Calendar. Today, many business people are using Microsoft Outlook on a PC, or a similar calendar application on a mobile phone, in order to synchronize to the Exchange Server.

3.1.5 Conclusions

What we previously mentioned, and as have seen in examples, different calendar application implements some subset of calendaring functionality. The iCalendar standard offers a predefined set of calendaring components, but not all of these components are needed, therefore not all are supported or implemented at all in all applications.

[§]Mozilla's Thunderbird is an application to receive and send emails and Lightning is an extension to this application that allows managing calendars through Thunderbird. Sunbird has exactly the same functionalities as Lightning, but is a stand alone application.

We have also seen that today's calendar software predominantly use a client-server architecture. Even if services such as Tungle make it possible to connect different calendars, Tungle is still not a pure peer-to-peer program, either from the point of view of a centralized web scheduling management system or in terms of occasional users who have their own agents running on server machine(s). Tungle accommodates server calendaring applications of today, rather than inventing a new peer-to-peer approach for collaborative calendaring.

3.2 Location-aware applications

Because the concept of location is important (especially for this thesis) we will present a number of different applications that utilize location information. The applications were selected because they represent some of the variety of applications that exist and some of them have specific features that we would like our own application to have.

3.2.1 Location-aware browsing

There are many services that could benefit from knowing the user's geographical location when he/she access a web page. For example, when searching for a nearby restaurant, shopping center, or patrol station, a map of nearby located services could be provided to the user just by entering the type of service desired, but without the need to explicitly input the user's location. Another example of a location aware web browsing service is when a user's location is automatically provided to a local transport service, thus setting an origin point for a route to some point in the town.

Since the release of Mozilla's Firefox 3.5 it is possible for the user to learn his/her location (in terms of latitude and longitude) and to easily pass this information to an application [27]. When a web page is accessed that wants to know where the visitor is, Firefox will first ask for the visitor's permission to share this information. If permission is granted, Firefox will gather information about nearby access points (and the device's IP), then send this information to Google's Localization Services that will calculate the position. The accuracy of the location determination can vary depending on where the page is accessed from. There is no guarantee whatsoever of the accuracy or precision of the location that is returned (hence this service should never be used in emergency situations). However, this does not mean that the service is necessarily bad. The user's integrity and privacy are very well protected because no names or other personal information are shared with Google Location Services or with Mozilla. Also, all information is sent encrypted and the user can always revoke the previously given permission for a web site to learn about the user's location when the site is visited the next time.

In her thesis [28], Jenny Charvandeh utilized this feature of Firefox. She combines the user's URL with the user's location information (which is what is done when a user permits Firefox to provide the user's location to a web server). For doing this she has implemented a simple web browser for the Windows Mobile Operating System, but she has also pointed out that a web proxy could be used for doing the same task, thus enabling any web browser to be used.

3.2.2 Google latitude

Google Latitude is yet another application utilizing Google Location Services. It provides you with the possibility to see the location of your friends and publish your own location information so that other Google users can see where you are on the map (Figure 3.4 shows an example of this). The published location information can be accessed both from a mobile device and a computer[¶], but only with the agreement of both parties.

Seeing each other's photos on a map and contacting your 'Google friends' in one of many ways is not the only things that you can do with Google Latitude. If you want to meet your friend, you can get directions for how to get to your friend's location or you can search for places nearby you or your friend. You can decide if you want to share your location only on a city level and with whom you want to share it.

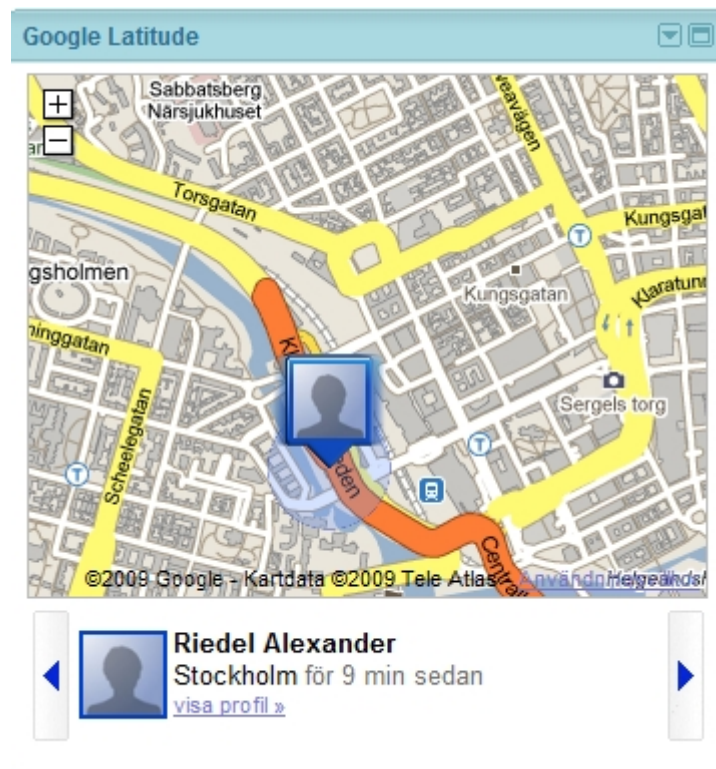


Figure 3.4: My current location determined by the 'Gears' and shown on a map.

Various technologies are used for location determination. If no Wi-Fi APIs are available, the location determination will (on most supported phones) default to utilizing a cell tower ID (this service is called "My Location"). The Wi-Fi and cell tower databases that are used by Google are incomplete and can be a source of error. Additionally, My Location is still in development. For greater accuracy, GPS can be used on phones that support it, although GPS will not be used for background

[¶]Google Latitude is available for a various of different mobile platforms and in various countries. The iGoogle gadget and Gears are needed for supporting Google Latitude on a computer. More information on http://www.google.com/intl/en_us/latitude/intro.html

location updates to reduce battery power usage. How often the device's location is updated depends on the battery's charge level, along with if and how fast you are moving.

3.2.3 The monger application

The student Benjamin Özmen has proposed ideas for a prototype application called Monger [26]. Its main goals are to inform traffic users of different traffic situations. The key observation is that once there is a traffic accident on a road, typically drivers that pass by will ask for emergency assistance thus making calls to the police, fire brigade, etc., while most other people in traffic on their way to this location are uninformed or only informed much later. With the Monger application running on the users' PDAs, everyone in the vicinity of a place, where any obstacles would block the flow of traffic, is given the possibility of getting a voice message that was previously submitted by another Monger user. In other words, everything that has been observed by someone near something, could be distributed to everyone within (or approaching) the area.

Monger could be designed as a component of a Traffic System and should not depend on the underlying topology of the network. Thus, per design, Monger should be a peer-to-peer application with bounds on the delay experience by the messages. Although the ideas and concepts are interesting, Monger was never finished.

3.2.4 Lcron

Lcron is a program that allows user-configured events to be triggered when the user's location changes [31]. Lcron is most useful for devices that connect to several different networks or a single network in order to trigger events such as sending e-mail or synchronize with a disconnected file system. The program is also able to trigger events (on a laptop computer) when the computer gets connected to a power source (when "at" the "AC-powered" virtual location [31][2], in order to run "housekeeping" jobs, such as defragmenting a disc, indexing of folders, checking for viruses, etc.

However, one of the main contribution of Lcron that is interesting for this thesis, is how Lcron utilizes location information in order to acquire high-level context. For example, although many (mobile) devices are able to determine their location, most people can not do anything with that information. The location information must be put into a context of the current/future tasks that people (or their personal devices) are doing/will be doing, in order to derive more useful information. The paper describing Lcron also points out that a polling of location services may be required in order to discover a change in location. Although a location change sooner or later may be discovered by applications, it is hard to trigger events such as "fetch my mail before I leave work", as a change in location can not be discovered beforehand.

Lcron is similar to the standard Unix command `cron`, with the addition that Lcron users may specify a task to be executed when at a certain location and/or connected to a power source. Lcron is also able to save user-defined logical location that is based on the IP address of the gateway of the network, as many tasks examined in the Lcron document are based on the network connectivity.

Experiments with Lcron showed the program to be easy-configured and that users could benefit from the automation of day-to-day tasks that were difficult.

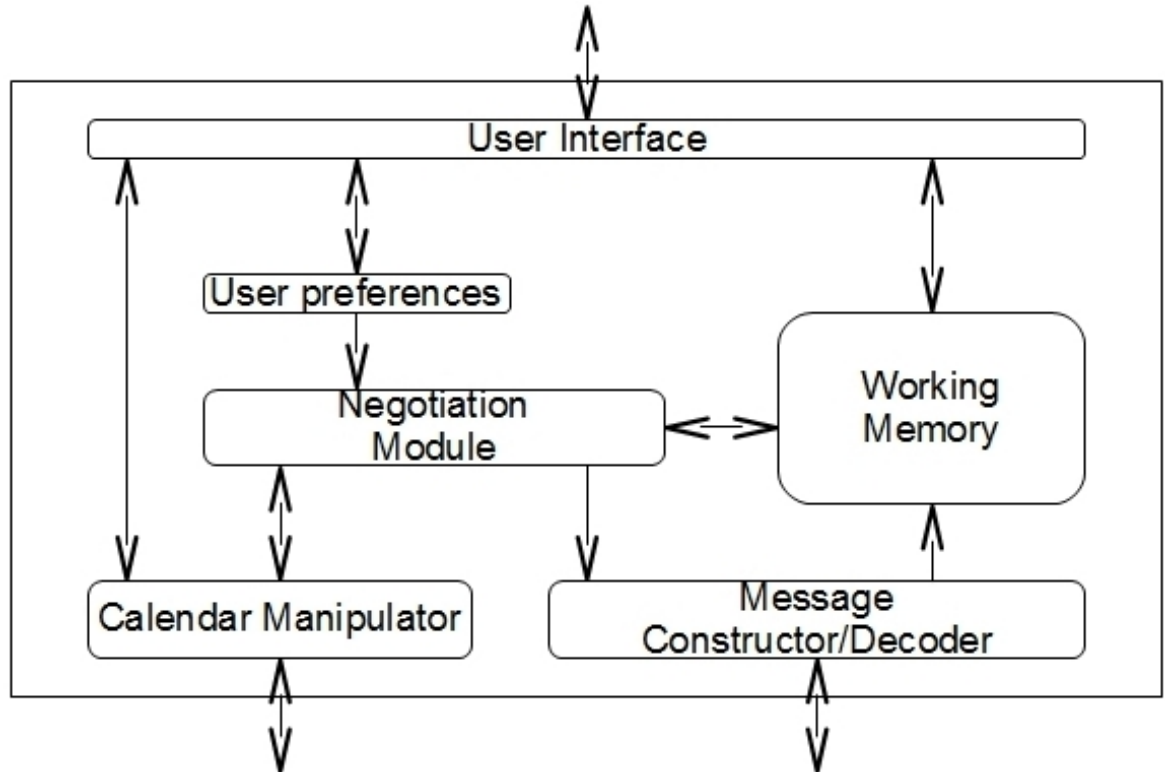


Figure 3.5: The architecture of the automated meeting scheduler, adapted from figure 1 in [32]. The user interacts with the scheduling system via the **User Interface**, the **User Preferences** module stores the preferences of the user, the **Working Memory** contains the traces of meeting negotiations, the **Negotiation Module** contains the scheduling logic, the **Calendar Manipulator** allows the user to access and modify the user’s schedule through a calendaring program, and the **Message constructor/decoder** component is the interface to send and receive e-mail messages.

3.3 Automated meeting scheduling system

This section studies an automated scheduling system proposed in [32]. The work of the authors of the following program is going to be important for this thesis project, because their scheduling program is a good example of a scheduling system. The program is using stand alone agents that communicate via e-mail messages. The program’s high-level architecture can be seen in figure 3.5.

One important contribution of the paper is that the program can schedule meeting automatically, while protecting user data because of the inter-agent communication. An automatic meeting scheduler at that time was a break-through, because none of the existing calendar sharing systems were able to autonomously schedule a meeting. The paper focuses on a scheduling mechanism constrained by

user preferences and presents a heuristic algorithm to agree on a meeting time. The simplified scheduling algorithm goes like described in the following:

1. The organizing agent tries to find time intervals in its schedule that satisfies the scheduling constraints and user preferences. The found time intervals are proposed to the attending agents.
2. Upon receiving the intervals from the organizing agent, all interested attending agents try to find time intervals in their schedule, respectively that match all the scheduling constraints and local preferences. Attending agents then send “bids” (a bid is either a subset of the intervals received or a counter proposal). Bids are selected according to user preferences.
3. If the organizing agent, after that all bids have been collected, finds a commonly available time, then the meeting is scheduled and this agent sends “awards” to the bidders. If no commonly available time is found, new proposals are generated depending on the received bids, and then sent out. Rejections are also sent in this step.
4. When the bidders receive new proposals, they will reply as in step 2. When an award was received for a given time interval, and this time interval is still available, then the calendar will record the scheduling of the meeting. Rejections are sent otherwise.

Besides a set of attendees, a meeting in this scheduling system has the following parameters: length, priority, a set of possible starting times, scheduling deadline, a constraint for scheduling a meeting only if another meeting is scheduled, a constraint to schedule a meeting before or after another meeting, etc. User preferences determine the priority of a meeting, the meeting priority in turn determines whether a user is interested in the meeting or not. The priority depends on a variety of parameters (such as meeting length and topic, invitees and organizer to the meeting, etc) and preferences (such as preferences for the length, topic, hours of the day and days of the week). Each parameter is given a threshold by a user and the users must specify a personal weightage scheme so that preferences can be dimensioned (it may be more important for a user to attend a meeting in the afternoon than to it is to know about the organizer). Users also may rate each other.

4. Realizing of the Scheduling Application

This chapter describes the design of the scheduling application. Every decision made in this chapter is based on the assumptions made during the study in chapter 2.

4.1 Application overview

In this section, all the steps that the application needs to perform before a meeting can be scheduled are considered. A fundamental description is given of what makes up a step and how and where an interaction with the user is expected/necessary. Each step will be discussed in more detail in the subsequent sections.

- 1) **Meeting setup.** A meeting organizer starts by adding a meeting on his client and invites participants for the meeting. Every participant will be sent an invitation. Each participant needs to accept/decline and the meeting could be scheduled with the participants who have accepted, or be cancelled when a required invitee has declined. Accepting means also agreeing on sharing of free/busy information.
- 2) **Pre-scheduling.** The application tries to find a time slot and suggest a meeting place. The results should be presented to the users in a convenient way.
- 3a) **Relaxing constraints.** If no time spot could be found the organizer or some other participant probably should be given a choice of further actions such as, postponing or splitting up the meeting.
- 3b) **Agreeing upon meeting occasion(s).** If the participants did not choose automatic scheduling and there are several possible alternatives, then they should agree on a time and location in this step. Once this is done, ask the users if they want to be reminded of the meeting in a later phase. The event should be added to their calendars.
- 4) **Smart notifying.** If desired, users can be informed about the state of other participants. A possible reminder notification might look like this: “You have an appointment in x minutes and you should be on your way there in y minutes” (this smart notification is tightly coupled to the user’s current location and the type of transport available), or “You are late to a meeting” (at the same time at the organizer’s device: “user A is late”).

Note that all these steps may need to be executed several times, as a meeting may be rescheduled. There are many reasons why rescheduling might be necessary, but these are reflected upon in section 4.7. Before rescheduling, all the steps but the 4th step must have been completed. Also note that there might be other types of notifications beyond the ones covered by the 4th step, such as notifying the user about the scheduling progress.

4.2 Step #1: Meeting setup

On some devices, in particular those that are hand-held, the installed calendaring clients differ a lot. Few clients have a synchronization interface to a calendar server, some clients do not even allow the user to add meetings (although meetings technically are like every other event with the additional aspect of being able to invite participants). In fact, many clients do not support meetings and often work with a local calendar or solely with a read-only version of a remotely located calendar. In either case, our demands concern an even higher level of details. For example, the organizer might want to specify the first possible date/time whenever he/she is ready with some preliminary material, if any. A deadline might be another desirable constraint, because most tasks have a due date and often, meetings are associated with tasks. Another constraint might be an approximate meeting duration and whether a meeting should be scheduled automatically (after all participants have accepted the meeting request), for example as early as possible.

Once the organizer has entered the details, the meeting request should be saved and given an identifier*. The request (with the identifier attached) is then sent to every participant and everyone should reply either by accepting or rejecting the request. When a user accepts, he/she also agrees on sharing his/her calendar in order for other participants to get this user's free/busy information. Once everyone has accepted, the application is ready for the next step.

4.3 Step #2: Pre-scheduling

Once everyone has replied, relevant calendar data should be sent out in this or the previous step among all participants who have accepted the invitation. Relevant data is the calendaring data necessary for suggesting a meeting place and deciding upon a meeting time. These data could be attached in a message or as a reference (a URI) to a calendar server. How could our application learn from this data and how would it use this knowledge to decide/suggest a meeting place?

4.3.1 Meeting place priorities

The following equation will be useful in order to realize how the application should suggest a meeting place. Assume that $\{P\}$ is the set of all participants that want to meet and the set of all available times before the deadline is $\{T\}$. Then there are three different types of time periods in $\{T\}$ that the scheduling process could

*Because the identifier will be created on the organizer's device, identifier collisions are not excluded when this identifier later arrives at any other participant. However, for each iCalendar component, its creator must create a globally unique identifier, as was shown in section 2.2.1.

encounter. For a given time period $t \in \{T\}$ that is closest to a planned event of participant p :

1. $\exists p \in \{P\}$ such that p has at least one planned event for such a t .
2. $\neg \exists p \in \{P\}$ such that p has at least one planned event for such a t **and** $\exists p \in \{P\}$ such that p has historic information in his/her calendar.
3. None of the above.

Why is this important? Because the application should prioritize users that have planned events when suggesting a meeting place over users characterized by alternatives 2 or 3 above. This prioritization occurs because the location of planned events is more certain than the location information estimated based upon historic events. Similarly, the application gives priority to users having some information in their calendar - as these are users that the application can learn from. Although users that do not have any calendar information at all might not always be available for a meeting for various reasons, such users will be assumed to be free at any time, as stated in their calendars or because they do not have any calendars. Note that this does not mean that users without calendars will be forgotten, they still might be able to receive calls, accept invitations, and agree to attend a meeting. There is always the possibility of negotiating for a common time for a meeting with such users using a non-automatic method.

4.3.2 Learning from past events

If no user has any planned events before the deadline, then the application could try to learn from past events (having a location value) and build a model for an agenda. The agenda could then be applied on a day where a meeting could be scheduled. To construct a model for one day's agenda, all events might be important, not only past events in one particular day of the week. For example, even if a meeting might take place each Friday 14.10-14.40 for continuous discussions, other (meeting) events might not directly depend on the time of the day they occur. Thus, some events could occur at any time, others only on certain days. Recurring events are straight-forward to find in the iCalendar file (as is discussed in section 4.3.2.2).

4.3.2.1 Constructing a model according to event type

The scheduling system prefers to learn from (regularly) recurring events that have a high predictability and high accuracy for a user's location. For recurring events, a sufficiently long period of a user's calendar is needed in order to calculate a frequency histogram. This histogram could be used in order to learn about a number of different types of events in a given period of time. A meeting should be scheduled close to the location of the most frequent event.

If events are not part of a single iCalendar recurrence set, a central question needs to be answered: What makes two events "the same"? Most probably these two events are the same if they occurred approximately on the same location and time, but if the start time/event duration are variables in this model, an event's location information is the only tie-breaker. A location-centric approach is in fact more valuable for the application, because all potential meeting places that are close

relative to each person's current location would not change as long as the location of the persons who want to meet are (roughly) the same.

4.3.2.2 Recurring events

A way of finding recurring (and planned) events is to look for events that contain the `RRULE` property. This property (often used together with an `RDATE` that specifies a list of date/time values and the `EXDATE` property that specifies exceptions in the recurrence set) is used to define how often and in what time span an event will occur (see [7, p. 122]). In order to find all instances of a recurring event, the complete recurrence set, as specified by the `RRULE`, `RDATE`, and `EXDATE` properties has to be analyzed. iCalendar defines the calendar file to contain a single `VEVENT` object whose `DTSTART` and `DTEND` properties specify the time span of the first instance in a recurrence set. Once the recurrence set is created, the application would *know* all the exact date/time periods when the event will occur in the future, if any.

4.3.2.3 Learning about traveling

Typically, an individual has recurring events, whether they are added in the calendar or not. When such events are found and their location is determined, the application could generate a knowledge database where actual traveling times between two events can be stored for later use. Because the destination location is approached many times due to this recurring event, the application is likely to schedule a meeting at that place. Because the traveling time is (well-)known, the application can make a better estimate of the traveling time to that meeting, thus, the application has a better estimate of the time available for a meeting scheduled at that location.

The application could also learn about the maximum distance a user could travel within the time available. But for such estimates to be accurate, additional knowledge of what transport mode the user prefers could be needed (this may require direct input by the user). However, solely knowing the transport mode will not be sufficient to calculate the traveling time without knowing the local city's infrastructure (so that a path can be found between two points of travel). Even if the most reliable way of getting from one point to another might be walking, different persons walk at different speeds. There might not be a significant difference in walking speeds, but walking for a significant amount of time may determine if a person is going to make it to the meeting or not. If the person utilizes a bicycle, his/her average speed may also vary a lot from person to person. Thus, to have a good travel time estimate, additional context information is required.

4.3.2.4 Recording location in dummy events

If no (planned) future events exist in a user's calendar, the application could try to learn from the past events by observing several weeks of historic calendar entries, in order to estimate the user's likely location in future. However, such historic calendar information might or might not exist in a user's calendar. A simple approach to ensure that every user (after $\Delta t = \text{deadline-current_time}$) has enough historic events having location information, so that the scheduling application would know about the location of that user at any time before a meeting's deadline, is to automatically

create dummy events to record the user's location. Given this rich source of location information, scheduling a meeting with such user would enable the application to suggest a suitable meeting place.

In this approach, a user's current location information is added in their calendar in a dummy event. Note that users probably do not wish their calendar to be "polluted" with dummy events, thus a different calendar should be used for recording this location information. To add a dummy event, some criteria for determining the length of such "generic" events are needed. For example, if users typically schedule their day in 15 minutes increments, then a 15 minute duration could be chosen for these "generic events".

When someone constantly changes his/hers location, for example by traveling somewhere, it would be inappropriate to create thousands of mini-events that last only some seconds (or similar amount of time, depending on the update interval of the location provider used). Assuming that a meeting will take place at one fixed location, continuous location updates are not essential for the application, as long as the location stays within a bounded area (for simplicity a circular area with some radius could be used) the user is considered to be "in the same" location.

However, the period of time when a user is traveling between two points could be represented as an event too, in order to reduce overhead, but the application needs to recognize that the user is traveling between two or more points in the first place. For example, Matt needs to travel from home to a shopping center to buy shoes. He needs to take one bus for about 10 minutes, wait 15 minutes, then take another buss. Depending on the minimum time that we think a (usual) event would last, the time spent waiting for the second bus could be entered into Matt's calendar as a dummy event on its own, even though it is part of one trip. As traveling times can vary widely, "traveling events" should be allowed to be of arbitrary length. If a short meeting for Matt is 15 minutes, then we would add a dummy event of 15 minutes, while Matt is waiting and not moving somewhere. If Matt often has some business to do in that shopping center and also travels the same way, then the probability of this 'dummy waiting event' occurring is correspondingly high. In this case, while Matt is waiting for the bus, the scheduling system might schedule a phone meeting for him.

4.3.3 Example: Choosing the closest meeting place

The following example demonstrates how the closest meeting place could be found for two participants both having reoccurring events. Assume that Lars works in A and lives in B and Bengt works in C and lives in D. Assume that both of them start working at the same time, work nine hours a day each (including lunch), and that we consider a day where neither of them is going to a place other than their respective living place, after their work is finished. Both Lars and Bengt stay home once they get there. The distance between Lars and Bengt during the working hours is $d(A, C)$ and $d(B, D)$ after that they arrived at their homes. If $d(A, C) < d(B, D)$, then a meeting should be conducted at either A or C (in contrast to B or D) immediately before or after, or even during the working hours, depending on the availability of Lars and Bengt.

4.3.4 Meeting place candidates

Assume for now that we know roughly where each participant is before and after each previously computed commonly available time slot. The easiest meeting location for all participants is a known location of any participant(s). That potentially leads to two times the number of participants potential meeting places in theory, while some of these meeting places might be the same. Common sense suggests that each of the participants needs to be able both to travel to and from the meeting place, *and* attend a meeting within this participant's available time.

A participant might have more available time than computed by intersecting with other participants' available time slots. For example, if a participant has more time prior a meeting, then that means that he/she could travel for a longer period of time in order to arrive at a potential meeting place. It is important for the application to know how much time each participant has left over for traveling to a meeting place, therefore an example considering this will be given in section 4.3.5.

Note that a meeting place can be limited to a specific location because of specific resources available there (such as a meeting room equipped with a projector and microphones), in which case there may be a smaller number of potential meeting location candidates. The application has no idea what kind of collaboration task that the meeting people are planning, thus it may not always suggest an appropriate meeting location despite all the participants' locations that the application learns about from user calendars. This suggests that in the future the application may also need to consider the requirement for specific resources as a constraint that has to be satisfied when selecting a meeting location.

4.3.5 Example: Eliminating meeting place candidates

Assume that today is Friday and Alice wants to meet Bob. She creates a meeting with the following parameters:

Title: Meeting with Bob
Duration: 30 min
Deadline: Wed, 20.00
Invitees: bob@company.com
Note: Discuss future recruitment for our company

Further assume that Bob has received and accepted the meeting invitation. Alice's available/busy scheduled along with location information from now to the deadline is (the notation is a day in the week: a triple consisting of the last location known before the free time, the available time, the location of the following event):

Mo: a_loc1, 13-15, a_loc2; a_loc3, 17-21, a_loc4
Wed: a_loc5, 8-9, a_loc6, 18-20, a_loc7

Bob's available/busy schedule is:

Mo: b_loc1, 14:00-14:45, b_loc2
Tue: b_loc3, 12-13, b_loc4
Wed: b_loc3, 12-13, b_loc4, b_loc5, 19-20, b_loc6

The intersecting times are:

Mo: 14:00-14:45 Wed: 19-20

As discussed previously, for each time slot we have 4 possible meeting places. That is, either Alice stays at a_loc1 after she is done with the event ending at 13.00 and Bob travels to this location, or both Alice and Bob travel to a_loc2. Similarly, b_loc1 and b_loc2 are two more candidate locations. We are of course not limited to only considering these locations and could potentially add more.

We denote the function $att(user)$ as the *allowed traveling time* both from and to the meeting place for the user $user$. Given this definition, we can state that, no matter what meeting place, we have to satisfy

$$\begin{aligned} att(Alice) &\leq 90min \\ att(Bob) &\leq 15min \end{aligned}$$

for the meeting time on Monday and

$$\begin{aligned} att(Alice) &\leq 90min \\ att(Bob) &\leq 30min \end{aligned}$$

for the meeting time on Wednesday.

That is because, for each user, both traveling times and the meeting's duration need to fit into the available time window. More generally, assume that the list of all participants is P and that the date/time stamp of the meeting deadline is D . Further assume that S denotes a specific time interval during which a meeting could be scheduled. Thus S is an intersection of all of the participants' available time on a particular day. In the above example we have two possible meeting possibilities: $S_1 = 14.00-14.45$ on Monday and $S_2 = 19-20$ on Wednesday. The functions

$$\begin{aligned} start_ftime(timeinterval, user) \\ end_ftime(timeinterval, user) \end{aligned}$$

give the start of the available time and the end of the available time for a user $user$ in a given *time interval*, respectively. For example

$$\begin{aligned} start_ftime(S_1, Alice) &= 13.00 \\ end_ftime(S_1, Alice) &= 15.00 \\ start_ftime(S_1, Bob) &= 14.00 \\ end_ftime(S_1, Bob) &= 14.45 \end{aligned}$$

Thus, for a given potential meeting occasion S we have

$$\forall p \in P : att(p) \leq end_ftime(S, p) - start_ftime(S, p) - D \quad (4.1)$$

The reader may note that the above relation is meeting place agnostic and does not depend on when (within time interval S) the meeting is going to be scheduled.

However, this relation may help the application to remove one or more potential meeting places from the candidates list of all meeting places.

When no information about the metropolitan infrastructure is available, the application could compute the air distance between the meeting place and the origin/destination location of each user and make a good estimate of when it would be possible for the user to walk, or whether some faster way of traveling, like a vehicle, will be required. For example, assuming a speed of 5 km/h when walking, a distance greater than or equal to 1.5 km requires more than 15 minutes of walking. Similarly, the minimum time to cover a distance of 10 km is 7-12 minutes assuming a car driving at a speed of 50-90 km/h.

When the application has learned about a user's traveling route and time to a meeting place, that is, either by user input or some third party service giving this information, the allowed traveling time constraint from equation 4.1 can be applied on the actual traveling time to see if traveling so still makes sense.

4.4 Step #3a: Relaxing constraints

In this step, no common time slots could be found that are suitable for every attendee. While everyone could be informed about this, only the organizer should do something about it (such as splitting up or extending the deadline for the meeting). Theoretically, this change of meeting parameters might lead to a feasible schedule for a meeting. For example, there is no point in splitting up a meeting when there are no common time slots at all. If the meeting can not be scheduled because of its duration, a shorter meeting could be scheduled. The application should suggest shorter meeting durations that are feasible. The application also could inform the organizer about common time slots that are available. If it is possible to specify a preferred meeting duration and scheduling this meeting fails because of insufficient time being available, then the application could suggest extending the deadline.

There are many parameters that can be changed in order for the scheduling to succeed, but the person who is rescheduling should know what change would make it possible to schedule this meeting. Most of the work should be done by the application, rather than the person who, if all else fails, will eventually be forced to manually view all of the participants' calendars him/herself. Thus, the scheduling application may require methods for suggesting both a parameter to be changed and a "smart value" for this parameter. In addition, the user in charge of changing the meeting details should be aware of the consequence of changing one or more meeting parameter(s), so that he/she can intelligently make this change.

4.5 Step #3b: Agreeing upon meeting occasion(s)

It is important for the participants to agree on the same set of meeting occasions. However, it should not be required for all users to actively select the meeting occasion(s); some users do not care exactly when to meet, or simply do not have the time to carefully select a meeting place, while others prefer to organize all these details. Also, these details might be meeting dependent - for example, consider meeting a friend somewhere outside his/her house, or having a company meeting in a conference room at the company's head office.

4.5.1 Filtering of meeting occasions

If there are common time slots available, then the users (or the application) will have to agree on which time slot they want to book. Because there are, according to section 4.3.5, potentially two meeting places per user per time slot, the users might be overwhelmed by the list of choices, if they care about where and when to meet. Therefore, rather than displaying the whole list, for each occasion, the application should choose one “good” meeting place according to some method. In the example presented in section 4.3.5, a good idea is to present a meeting place near one of the locations Bob is (going to be) in his available time. On the other hand, since Alice has more available time, she could travel more in order to meet to Bob. Generally, a meeting place that every attendee has enough time to get to is a good choice, but the user should also be able to see other meeting places if desired.

4.5.1.1 Smart occasion listing

The visual list of possible occasions should be limited to less than some predefined maximum size (that we define to be n). Which items are chosen to be in the list depends on the meeting’s deadline. The user should get a good overview of choices that are possible just by looking at these n items from which he/she easily could pick one.

For example, if a meeting’s deadline is three weeks from now, and there are more than n possibilities for the first week and another n possibilities for each of the next two weeks, listing the first n items would not represent the whole set. The users should get to see any of $n/3$ items found together with their amount for each of the weeks. If the deadline is in 6 days from now, $n/6$ items could be presented from each day. More formally, the time left until the deadline (d) should be divided into m periods of equal length, namely $m = d/n$. For each period, at most n/m of the first occasions are shown together with the total number of all occasions for that period. To make agreement easier, the users should not get too many occasions too choose from; but these occasions need to be representative of the range of alternatives. How many items should be presented might in the end depend on each user’s individual preferences.

4.5.1.2 Setting priorities

How should the participants agree upon these n occasions? If it matters for the participants, they could proceed by setting priorities from 1 to n for chosen occasions. They could set priorities only for the items most important to them, whereas the first possible occurrence would be of highest priority by default (allowing for automatic scheduling). If users do not care about exact occasions, but still want to set priorities, the users could do that for bigger periods of times. If a good algorithm for determining the number ($= m$) of such periods is chosen, then it would be easier for users to set priorities on the period level, because there will never be too many of alternatives to choose from due to the algorithm. The highest priority a period can get should be equal to m .

When every participant has set their priorities (or chosen to let the application set the highest priority for the first possible occurrence), the item with the highest

priority wins and will be used as the meeting's occasion. If two or more items have the same priority sum, the general rule "to strive for getting done with the meeting as soon as possible" would lead the application to choose the earliest occasion.

Note that in contrast to calendaring applications like Tungle (section 3.1.3), an application working as described above would find a solution if a solution exists. Thus, automatic scheduling is guaranteed to find all common time slots, if they exist, and select the one with the highest priority, although it might fail because of insufficient time being available for traveling (i.e., equation 4.1 can not be satisfied), but that is by design.

Before going to the next step, the users will be notified about the resulting time slot and the meeting will be added to their calendars. The users should also be prompted asking if they want to be notified (to enable the next step).

4.6 Step #4: Smart notifying

This "step" is optional, because not all users want to receive notifications (users should be able to configure the activation for each type of notification). There are different types of notifications: For some of these types of notification to work, users might need to agree on both sending and receiving context information. For example, if user A wants to know when user B has arrived at the meeting place, either user A has to know about the current location of user B, or user B once they arrive has to notify user A. For notifications reminding a user of a meeting all context information is locally available or could be acquired from the user by asking. In the following subsections two different notifications are discussed.

4.6.1 Participant's arrival status notifications

To be able to generate notifications such as "All participants except you have arrived", the current location of the participants needs to be compared to the location of the meeting place. The users individually can/should calculate the distance between the meeting place and their current location (since that will not require sharing their current location information). If the distance is within a bound of X meters, the application can assume that the user has the same location, thus has arrived at the meeting place (X depends mostly on the accuracy/precision of the location determination service and the definition of someone "being near" someone/something). This user could send a message to all other participants stating that he/she has arrived. Any other participant interested in the message could configure his/her mobile phone to vibrate/play a sound when such a message is received. Note that because all participants want/need to meet, it is in everyone's interest to know about the arrival status. Therefore, each client's default behavior should be to sent out arrival status notifications to other participants (assuming that everyone is interested). A small extension to the application would be to keep a dynamic set of all participants that have arrived (denoted A). Because the set of all invited participants (denoted P) is known, a user could be offered the possibility to view the intersection of A and P , thus learning the absent participants' names (or other identifier). In addition, a user could see the current names (or other identifier) of the participants who have not yet arrived each time a new arrival

status notification is received or when the current time becomes equal to the planned start time of the meeting. In the last case the organizer might call the missing participant(s), if any.

4.6.2 Reminders of a meeting

The application could remind a user to “start to get ready for the meeting starting in xx [time unit]”. It is important to note that the scheduling application will try to generate “best effort” notifications for users in order to achieve collaboration by appropriately scheduling a meeting. Notifications that include the approximate direction, a full point-to-point description of the path, and/or the traveling time to a meeting place, are not part of the scheduling application itself, although rough estimates of the traveling time could be computed. The estimates are only advisory and should not be relied on. However, location-aware notifications might be of greater value to a user than solely time-based notifications.

How can the application estimate the time needed to finish a journey? The application can compute the distance between the user’s current location and the location of the meeting that the user is invited to. The user could be asked about his preferences for travel, e.g. about the vehicle type that the user has access to and the maximal preferred walking and cycling distances. When the computed distance exceeds the maximal preferred walking distance, the application would not consider walking as a “traveling alternative”. The same reasoning could be done to determine if taking a bicycle would be an option or not (but only if the user has a bicycle available to them and wants to use it).

4.6.2.1 Traveling route alternatives

There are many web services that estimate the traveling time when driving in a car between two points. If a user does not have a car or has selected that he/she prefers not to take a car, the local transport agency could be consulted, if that is an option for this user. All traveling times gathered/calculated could then be sorted in descending order and as soon as the current time (according to the device’s clock) plus the traveling time of the first item in the list together start approaching ΔT (which is known parameter input by the user for this particular meeting), a notification could be triggered. In this notification, all (by the user selected) types of transport together with their corresponding estimated/acquired traveling times could be presented in a list sorted in descending order (in order to show the alternative that is going to be removed first if the user is not going to take this alternative).

ΔT is basically the amount of time the user personally has allocated for (1) getting ready before the upcoming journey, (2) arriving at the nearest bus stop/train station or bicycle/car (whichever applicable for triggering the notification with the parameter ΔT), and (3) arriving at the meeting place, e.g. by walking to/inside the building from a train station or parking garage, (3) is the time needed after the journey has finished and the user only needs to walk in order to arrive at the meeting place. ΔT is user and meeting specific (it probably also depends on the transport type used, how busy a user is before he/she can start traveling, etc., but for simplicity it should only be a single parameter).

Reminders of a meeting only offer additional information and could be ignored by the user. However, the information presented in such reminders should be up to date; both with respect to the user's location and the time. For example, once the user has changed his/her location, the starting point of the journey probably also has changed and because of that, the path needs to be recalculated. Also, the traveling route should be adjusted when the user has chosen to take the subway/bus/train. Even though the user did not change his/her location, the traveling route (and the duration of the journey) might have changed.

A completely different (and common) situation is when a user *knows* where he/she is going to be before the meeting, e.g. the user has a planned event in their calendar. If the location is known beforehand, no continuous estimates for the traveling time need to be made - the user could plan the journey once, either when (1) a meeting is scheduled or after some event or (2) when a new event is scheduled prior to a scheduled meeting. In both cases, the application could try to encourage the user to plan his/her journey between the events and save the duration of each traveling alternative (if there are several alternatives) in order to remind the user (once) in the same way as discussed above (except that the traveling times are not going to be recalculated). The application would need to know, if according to the user, there is "too much time" between two events and the user, instead of going from the first event's location directly to the second, first desires to travel somewhere else. In that case, the "full" reminder procedure would apply.

4.7 Moving a meeting to a different time

Sometimes it might be necessary to reschedule a meeting. For example, when one of the required participants gets sick, or can not attend because of a similar reason. When people are not delayed, it is hard for the application to know why they can not make it to a meeting and it may even be inappropriate to autonomously make a decision for the user about the rescheduling.

However, there might be other reasons why rescheduling should preferably be done automatically. One such "trigger" might be a change in a user's calendar, such that this user becomes available for a longer period of time. A meeting moved to a newly freed period of time may in fact be desired if this period has been given higher priority according to user preferences, than the period where the meeting currently is scheduled. However, users may or may not contribute to such "positive" calendar changes, as not all of them may modify their calendars so that available time is induced. On the other hand, if a user's calendar to some extent is managed by a scheduling program, such a program should contribute to such calendar changes. For example, a cancelled meeting should not block available time, although the user might choose to keep the meeting object in his calendar for record-keeping reasons.

Another reason for automatic rescheduling might be a user changing his/her location. As was discussed in sections 1.3.4 and 2.3.1, a change in location may result in a participant leaving his/her usual whereabouts and entering the city area where another participant might live. The application should be able to discover such or similar location changes that could bring people sufficiently close to each other, thus enabling *ad hoc* meetings to occur "just in time".

4.8 The communication approach

There exists different ways for participants to communicate with each other. Should the attendees talk to each other or only to the organizer? If the attendees can communicate with each other, what information should they exchange and what information should be sent only to the organizer? Figure 4.1 shows the “communication end points”.

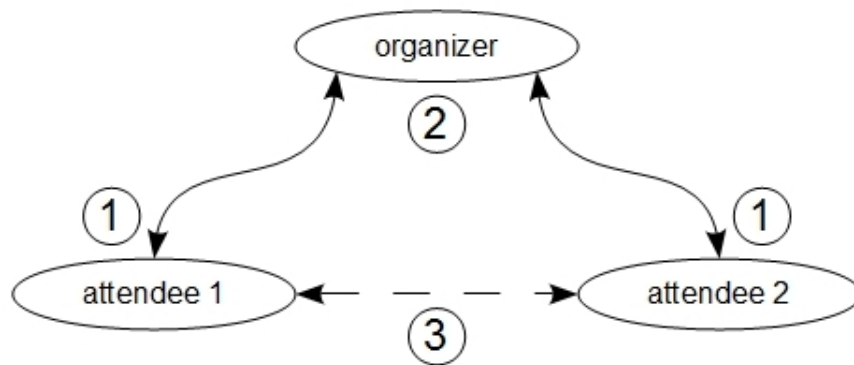


Figure 4.1: The “communication end points” between meeting participants. The dashed arrow between the attendees indicates that may exchange only a subset of the scheduling information, or no scheduling information at all, between each other, compared to the information they might exchange with the organizer. There might be 3 different types of message pools to choose from, because of the 3 different combinations of end points: 1) organizer → attendee, 2) attendee → organizer, and 3) attendee → attendee.

Are attendees interested in knowing who of them has replied to a meeting invitation? They might be, even though they might not be authorized to invite meeting participants. When the organizer has broadcasted an invite, the attendees could answer to the organizer only. Thereafter, if the invitees are interested in participating, they could send their free/busy information (to the organizer), the organizer proposes a time, and, finally, receive a yes or no answer from each of the attendees in order to accept or decline the time proposal, respectively. If every attendee has answered with a yes, the organizer must confirm the booked meeting by sending a confirmation message to all the attendees. This approach will be referred to as “the first approach” in the following.

If everyone sends the last reply (to a proposed time) to everyone else, then the confirmation message would not be necessary, as every peer would be able to autonomously learn about the booked meeting. Having more than one proposer is not appropriate since that would require an additional mechanism for choosing “the best” proposal. There is no point in choosing the best proposal since every proposer would propose the same time slots because they use the same algorithm and input parameters. This suggests an alternative approach without having any proposer at all; in this approach each peer calculates a meeting’s start time individually based on the calendar view it has received. Hence, the peers need to have the same view of each other’s calendars, so that they can “agree” on the same meeting time. In addition, the peers must know everyone’s personal priorities to be applied on the scheduling.

This “no-proposer approach” will be referred to as “the second approach” in the following.

4.8.1 Communication approach comparison

Assume that we want to compare the minimum number of messages produced in total by each communication approach. Further we assume that the number of peers is $n + 1$. In the first approach where the organizer is the only peer receiving and sending messages to everyone else, $6n$ messages are sent in the best case (when the meeting gets scheduled in the first slot proposed). That is, n invites, n accepts, n free/busy messages, n proposals, n yes-answers, and n confirmation messages. In the second approach, where every peer autonomously learns about a meeting start time, $2n^2 + n$ message are sent in the best case. That is, n invites from the organizer to each attendee; n^2 accepts and n^2 free/busy messages from each attendee to every other peer.

$$\begin{aligned}
 6n &= 2n^2 + n \\
 0 &= 2n^2 - 5n \\
 n_1 &= 0 \\
 0 &= 2n - 5 \\
 n_2 &= 2.5
 \end{aligned}$$

From the above calculation we can see that as soon as the number of participants exceeds 2, the number of messages sent in the first approach is less than the number of messages sent in the second approach. Note that this is true only for the best case.

If the meeting does not get scheduled at the first proposed time, then the number of messages in the first approach increases by $3n$ for each additional proposal made; that is, n messages for each of free/busy, proposal, and yes-answer. In the second approach, each additional proposal costs n^2 messages. We assume that p is the number of proposals necessary before a successful schedule can be made. In the first and second approaches, the invites and the accepts are sent only once; in the first approach the confirmation message is also sent only once (at the end). These observations yield the following equations (m_x is the number of messages in the x -th approach): $m_1 = 3np + 3n$ and $m_2 = pn^2 + n^2 + n$.

$$\begin{aligned}
 3np + 3n &= pn^2 + n^2 + n \\
 3p + 2 &= pn + n \\
 n &= \frac{3p+2}{p+1}
 \end{aligned}$$

The above equation says that $2.5 \leq n \leq 3$ for all values of p . Thus, the number of messages sent during the second communication approach is always greater than the number of messages in the first approach when the number of participants is at least 3. In conclusion we see that the first approach is generally preferable - hence this will be the approach that we use in our application.

4.8.2 Calendar view synchronization

So far we have analyzed the performances of both communication approaches in terms of messages exchanged. In this section, we re-exam the requirement of the second approach - that peers must have a synchronized calendar view - and examine the cost of this requirement.

First of all, every peer must send the free/busy information when it has received all the other peers' accepts, because otherwise it can not know if the meeting is going to be scheduled or not. Secondly, sending free/busy information to a peer immediately after receiving its accept would lead to changed free/busy information being generated and sent to subsequent peers, in the event of a calendar change. It is important to send the same free/busy information to all peers, which is why free/busy generation should be done only after a peer has learned about each peers participation status (which must be accepted for every participating peer).

However, the problem of unsynchronized calendar views between the peers is persistent even then. This is because peers need to wait for everyone's free/busy before they can compute the common available times, and while waiting, the calendar of the waiting peer might have changed, although this peer already has sent its free/busy information. The waiting peer can not freeze its calendar because the calendar might be modified by the user in the meantime. Thus, if a single peer's calendar has changed, this would enable this peer to book a meeting at a different time slot than another peer. In such a situation, the second approach would fail.

4.8.2.1 Possible solution

Clearly, peers need to have an additional message type, such as "abort" or "invalid" to signal a change in a calendar. The signaling peer must also recreate and resend its free/busy information. However, in order to discover a change in a calendar that possibly could lead to a different meeting schedule, peers may need to save their own free/busy information that they previously had sent to others, so that this information can be compared against their current calendar.

This suggests a two-phase-commit protocol. In the first phase, the participating peers send their free/busy information to each other, thus implicitly requesting to (re)schedule a meeting. In the second phase, peers that discover their calendars being changed, should send an abort message to the other peers followed by a freshly generated free/busy information. This free/busy information could either be recreated for the time span of the whole period until the meeting deadline, or could solely contain the changes that were discovered by the aborting peer. If a peer's calendar has not changed, or if a change in calendar would not compromise agreement, then peers with such calendars could send an "acknowledgment" message to every other peer.

Peers that receive an abort message, wait for an updated free/busy information from the originator(s) of the abort messages. Having this information, peers can proceed with the scheduling protocol as previously was discussed.

5. Implementation

In this chapter, a description will be given about the application's implementation details and the design choices that were made. This chapter also describes details of the communication (application) protocol used, the meeting scheduling states that have arisen because of the protocol, and some difficulties that emerged during the implementation process.

5.1 Design choices and their effect

A key design choice was not to use any server for scheduling - as there was no need for a server. Each user is equipped with a device powerful enough to run a minimal scheduling algorithm, periodically send and receive e-mail messages, **and** talk to a calendaring server.

5.1.1 Choose of the Android platform

The Android platform was chosen for development. Besides that Android powered phones are getting more and more popular and Android has a well-defined application programming interface and allows applications that to run inside such devices, to interact with each other (see section 5.1.2). One of the ideas for the scheduling application was that the application could benefit from using an already existing contact list. A major portion of all contacts of an individual are probably stored in that person's mobile phone. Because of this we want to allow the users of our application to select who they want to meet directly from their contact list (the contacts must of course have a mail address specified in order for the application to be able to fetch it). Another motivation for selecting an Android phone is that the scheduling application could utilize both the calendar and the location manager API (in order to get the device's location). The scheduling application might even benefit from the Android notification API.

However, the choice of the Android platform also has some drawbacks. Even though an experienced Java programmer might be happy about that the language used in the Android (as it is a Java-derivative), not all Java packages are supported or included in the Android Software Development Kit (SDK). The Android SDK is a rich library, almost like the native java SDK, because the Android SDK offers more functionality than a java ME capable device. Unfortunately, many packages are not included in the Android SDK, thus it is not guaranteed that arbitrary Java files will be able to run on that platform. For example, a jar file implementing the CalDAV functionality, as a high abstraction level API, could not be verified when we tried to build it for the Android (see figure 5.1). Thus, we were not able to utilize the CalDAV protocol in our application. Another disadvantage of the Android platform

is that it at this point has only been around for one year, therefore many bugs remain.

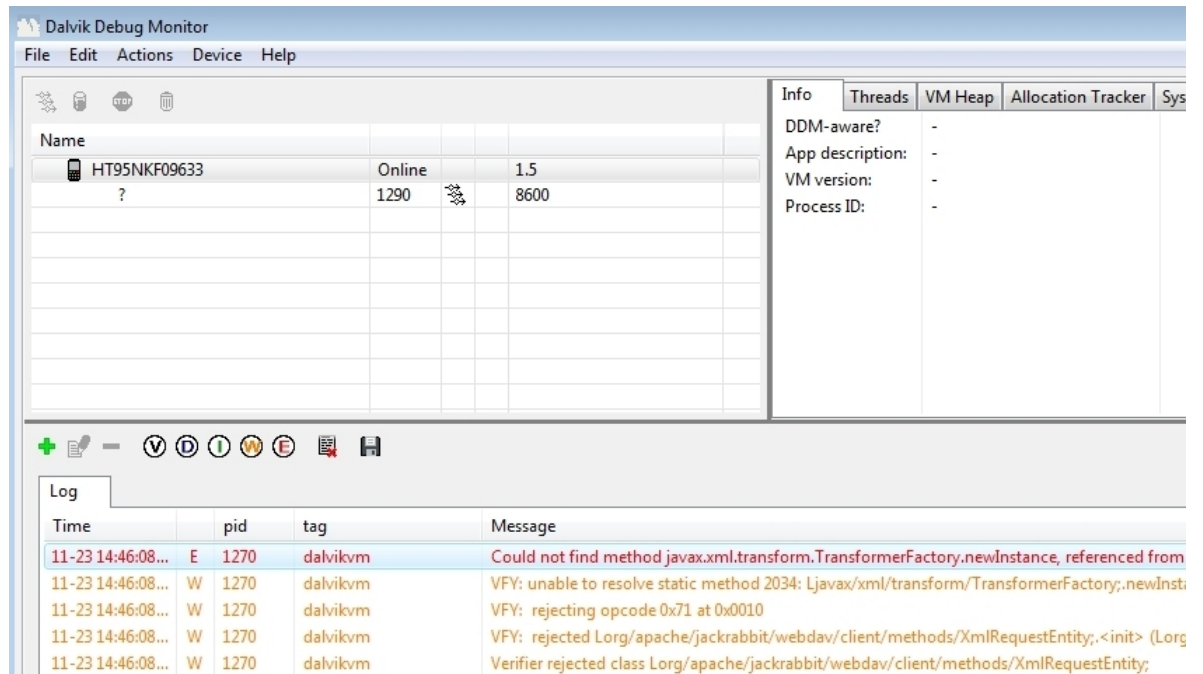


Figure 5.1: The Android SDK debug monitor showing that the virtual machine verifier rejects a class that references a non-existing method. This class is part of the CalDAV library that we were not able to use because of the broken reference.

By choosing Android we indirectly also are bound to use either the Google Calendar or the Microsoft Outlook Calendar, as the CalDAV library could not be used. Fortunately for the application, there was not any need to write code that would synchronize the device’s calendar database with the calendar database on either the Exchange or Google servers. That is, whenever a change was made in the devices database, that data could be synchronized back to Google/Outlook Calendar (and vice versa) within a short period of time, as this functionality is already available in the Android platform. Figure 5.2 shows how SCAP interacts with other “components” on an Android device.

5.1.2 Developing on Android

In this section we briefly review all the necessary terminology in order to understand the basic concepts of an Android phone. The next sections will use the terminology described in this section.

5.1.2.1 Android fundamentals

Applications designed to run on an Android platform are written in the Java language. When an Android application is compiled an *.apk* file is created. This file is uploaded and installed on the mobile device.

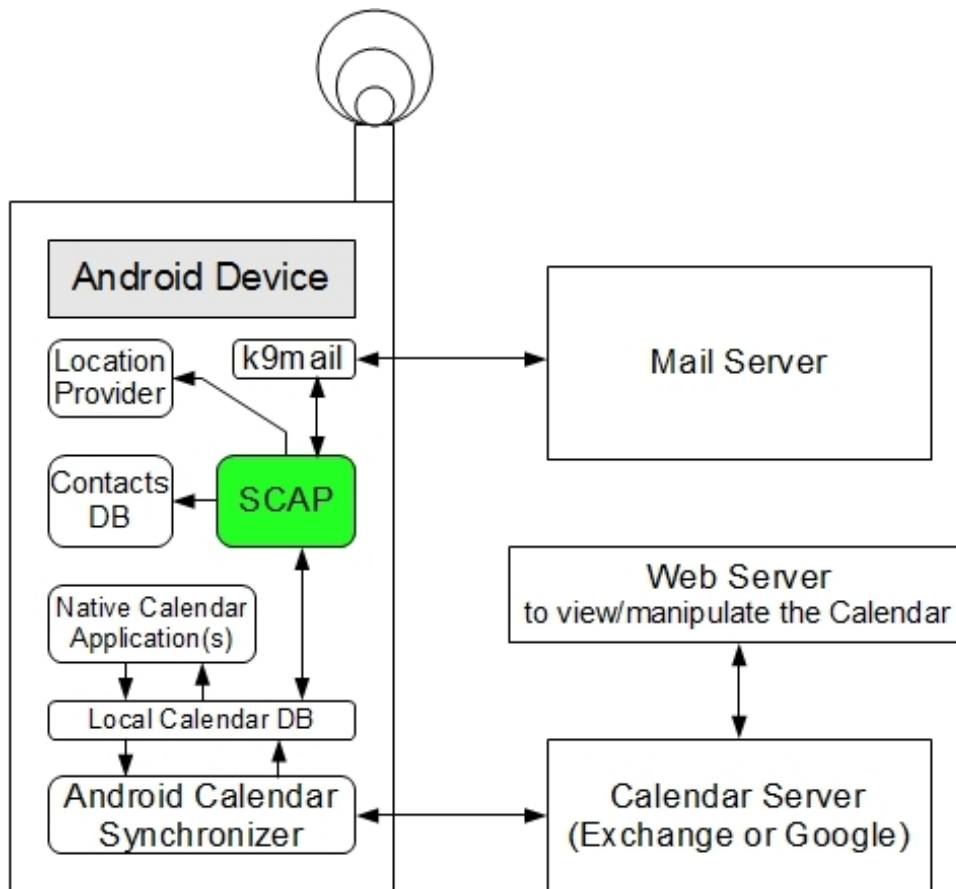


Figure 5.2: The interaction between the scheduling application (SCAP—marked in green) and other “components” involved. SCAP reads the contacts database, manipulates the calendar database, asks the Location Provider for the current location of the device, and interacts with the k9mail application in order to learn about incoming e-mail messages and in order to send messages. The calendar database on the Android device is synchronized “in the background” with a calendar server with the aid of the calendar synchronizer.

By default, each application is run in its own Linux process and each process in turn has its own Java virtual machine, in order to isolate one application from other applications. Also, each process can only access files that it has created by itself, although there are other ways to share files between applications (section 5.1.2.2). When an application’s code is to be executed, the Android Runtime will check if this part of the application (this component) already is running and will only create a new process if necessary. Because Android applications consist of different components (components are discussed in 5.1.2.2) that are independent from each other, it is possible to start/destroy these components as needed. Thus, components can be reused by other applications, but only if the requested components have the appropriate permissions allowing them to be used by a given application [30]. As an example, consider a user surfing on the web and clicking on a link representing an audio file that the user wants to hear. The host operating system on the device

where the file should be played, downloads the file, and when the download has completed, the default audio playback program is started in order to play the file. On an Android device, the component to play this audio file would be instantiated (if not already running) and a message containing the details of the audio file would be passed to this application, so that this application can open the file and start the playback.

In order for components to be called from different applications (on the same device), components must have many different entry points. Additionally, the Android developer does not have full control of a component's life cycle, because the component may be destroyed by the Android Runtime when the need for available memory arise, or (re)instantiated, when brought to the foreground. However, the component is always notified when it is about to be destroyed, so that the programmer can save the component's current state, and restore it when the component is alive again. A component is represented by a single Java object. The type of component determines which and how many different entry points it has and its life cycle.

5.1.2.2 Application components

Android applications can consist of any combination of the following types of components: activities, services, broadcast receivers, and content providers [30]. An activity is a single "window" on the screen typically dedicated for one focused endeavor. An activity can for example display the details of one contact, or a list of contacts to browse, or display the latest calls made from a mobile phone.

A service runs in the background without any graphical user interface. An example of a service is a process playing music in the background, so that the music continues to play while the user navigates between different activities. Activities can connect to a service and communicate with it through the service interface.

A broadcast receiver is a component that reacts to broadcast messages (section 5.1.2.3). Many broadcast messages are system specific (originating in system code), such as notifying that the time has changed, the system has finished booting up, or that the battery is low; other messages may be broadcast by user-created applications. A broadcast receiver does not have any graphical interface, but it can start an activity upon receiving a message.

A content provider is a component providing applications access to specific data. The data can be saved in files or in an SQLite database, and must be of a well-defined type so that requesting applications and the content provider exchange data that is to be interpreted in a consistent way. An example of a content provider is the calendar provider that allows applications to manipulate the local calendar database on the device.

5.1.2.3 Component messages

Components can be activated through messages, so called intents. Intents are objects holding a description of an operation to be performed, or a description of something to be announced. In order for a component to be activated by an intent, the component must specify an intent filter which must match the description in the activating intent. If a component wants to activate a component within the same

application, no filter is needed (other than a canonical name of the class representing the target component) as both components are part of the same application.

Intents containing a class name are called explicit intents. Implicit intents do not specify a matching component literally, but rather the Android Runtime finds all available components that match the given description in the intent. For example, when the user has taken a picture with his mobile device, the picture is displayed on the screen so the user can decide whether to save the picture or not. When the picture is taken, a different program could be started by the Android runtime that is capable of displaying the picture. The intent for opening the picture could describe the format of the file, the file URI and the action. In this case, the action tells the receiver to open the picture for viewing, although the action also could tell the receiver to open the picture for editing. For editing a different application might be launched, if such an application is installed on the device. If no such application is found, a warning is displayed telling that the desired image file can not be opened for editing (in this case). The rules for matching intents against the filters can be found at <http://developer.android.com/guide/topics/intents/intents-filters.html>.

5.1.2.4 The application manifest file and resources

Each Android application must have a manifest file where each of the application's components and the component filter(s), if any, are specified. The Android runtime uses the information in the application's manifest file in order to run the application's components. There are a number of different Android versions, therefore an Android program can set a minimum version number (called "API-level") that is required for the program to run. The required API-level can also be specified in the manifest file. The exact details about the manifest file can be found in the Android developer guide (<http://developer.android.com/>).

An application's resource in Android terminology is an element that is included in and referenced by that application. Examples of resources are images, audio and video files, text strings, layouts, etc. These resources are saved in `/src` in the route directory* of an application. Each resource when placed in its respective directory and compiled, is given a unique identification number so that it can be referenced from within the program code. Locale dependent resources, such as text strings, can be placed into a string resource subfolder that identifies the language and locale in order for Android to fetch the correct strings when the system language is changed.

5.2 Communication between peers

E-mail messages were chosen as a communication mechanism, because the application exploits the store-and-forward delivery protocol of e-mail servers in order be able to be run (simultaneously) on different devices. Thus, a user has the possibility to have multiple different "agents", for example an agent on his personal computer, and another agent on his mobile device. The user can freely switch between these

*A route directory of an Android application is the main directory where all application's files are. For example, the manifest file is found directly inside this directory, while all Java source code files are in the `src` subdirectory. The resource files are found in the `res` subdirectory.

devices. For example, a user might manage meetings from a personal computer (PC) when in the office, switch to the mobile phone when traveling home/to a meeting, or switching to his PC while at home. Because e-mail servers already store e-mail messages, no further “context-server” is necessary in order for all devices to be in a consistent state.

Of course, this requires some mechanism, so that the agents do not work against each other when reading/sending e-mail messages. IMAP should be used for e-mail retrieval enabling several agents to receive their own e-mail copies.

Today, every (mobile) device is able to poll an e-mail server for new e-mail messages. There is already an application on Android phones called “Mail” that is capable of doing this. As soon as a new e-mail is received by this application, the user (if configured this way) and all applications on the device that are interested in new mail, are notified. Not surprisingly, each such application is notified by receiving a broadcast intent that it can extract a subject and the sender of the newly received message from. The applications can even see which account (by looking up the account’s e-mail address) that the message was sent to. All of this information could be useful for the scheduling application, but it is not sufficient. First of all, there is no way to access the contents of the message (for security reasons) and secondly, other relevant header information, such as the iTIP method, are not revealed in the broadcast intent. We certainly want to send iMIP messages by putting them inside an attachment, in which case we additionally must know how many attachments an e-mail message has and of what content type, as the scheduling application is only interested in calendar attachments.

5.2.1 Receiving messages

One part of the solution to the problem addressed is to utilize an e-mail program that ships with a content provider, so that the content provider could be queried for content in order to retrieve the attachments. Another part of the solution was to program an extension that would broadcast an intent in order to inform our application about the content-type and iTIP method of the received message. The e-mail program used is called “k9mail” (<http://code.google.com/p/k9mail/>) and it is based on the native Mail application (on an Android phone). Fortunately, k9mail is open source, thus it was possible to extend it to provide the functionality that was needed.

The scheduling application (SCAP) (package `llypa.dev.scap.mua.android`) and the extension of the k9mail application (package `llypa.dev.scap.service`) communicate by using broadcast receivers. Note that this may not be the best solution since any application on the device could receive these intents. Here the focus was to implement a simple communication protocol. Once the SCAP *MailListener* has learned that an e-mail with subject “SCAPA” has been received, the *MailListener* broadcasts a message with the action to download any attachments in this message. This is necessary because attachments by default are not downloaded. The *DownloadListener* (of the k9mail extension) receives the intent and invokes a method to download the attachments. Once this is done, the *DownloadListener* checks for attachments having `content-type: text/calendar`. If such an attachment is available, then the file containing the attachment is granted permission to be read and the attachment ID together with the e-mail ID are put in an intent with the

action “attachment_downloaded” and broadcast. If no such attachments exist, an error message is generated and directed to a log file on the device (because the subject matched, thus calendar data was expected to be attached).

A component named *AttachmentListener* in the package `l1lypa.dev.scap.mua` is responsible for handling intents indicating that all attachments, if any, have been downloaded for a specific e-mail. This component can now look up the reference to the file that contains the contents of the attachment, by asking the k9mail content provider for the attachment reference with the specific ID found in the received intent. The file reference, together with the sender’s e-mail, is passed to the Mail User Agent, where the file is parsed. Depending on the component type contained in the file, a corresponding Java object representing the specific calendar component(s) is created and a specific handler for that component(s) is called.

5.2.2 Sending messages

Clearly, when no e-mail messages are sent, then nothing will be received. SCAP needs to tell k9mail when there is something to send and k9mail should construct the e-mail message with the same content disposition and subject, so that this message can later be recognized by the receiving module. By default, SCAP will try to parse all incoming messages no matter what the destination account is (as long as they have the string “SCAPA” in the subject line, as described in section 5.2.1), but when sending an e-mail, an account needs to be defined. In fact, a unique address is needed because this address will be used subsequently to identify the originating party.

In SCAP, the primary sending e-mail address can be set in the SCAP settings menu (see figure 5.3). Because SCAP does not implement any of the sending/receiving e-mail protocol functionality, the corresponding account must be configured in k9mail. Account specific settings, such as the protocol used for sending/receiving messages, the e-mail server, sending/receiving port, the account’s password, etc., are managed using k9mail.

When SCAP needs to send a message, it saves the attachment (containing the iMIP message) in a file, just as k9mail did upon receiving an attachment. The file reference and the meeting’s unique identification (UID) are saved in an SQLite database; then the Mail User Agent explicitly invokes the *ScapService* of the k9mail extension module by sending it an intent indicating that an iMIP message needs to be sent. This intent also contains a reference to the file to be attached. It also contains the related meeting’s (UID), the method, and the sender (the primary e-mail address specified in SCAP settings of the device). Once the *ScapService* has started, it can make use of the *ScapAttachmentProvider* (package `l1lypa.dev.scap.provider`) that implements the content provider interface, in order to fetch the contents of the file, in order to construct and send the e-mail message.

5.2.3 Types of messages and their meaning

How does SCAP know about meetings? When a meeting object is created, it is given a unique identification, thus each message passed to SCAP can be uniquely identified and associated with a meeting. If the receiving peer does not have a meeting with this UID in its local calendar database, then an incoming request message must

be a meeting invitation for this peer. As previously discussed in section 2.2.2, a request message in general can be interpreted as a rescheduling request; in that case, this message must have a higher sequence number than any previous requests for this meeting and the receiving peer must know about this meeting. A request can also indicate that the organizer wants to update the values of one or more properties of a meeting. If the sequence number of the received meeting object is the same. SCAP uses this type of request message to confirm that a meeting has been scheduled (implying that everyone has agreed on the scheduling time). A request can also be used together with a **VFREEBUSY** component in order to request free/busy information, which in turn consists of at least three **VFREEBUSY** components (the meaning of these components is discussed in section 5.3.4). In the following, free/busy relates to all such components.

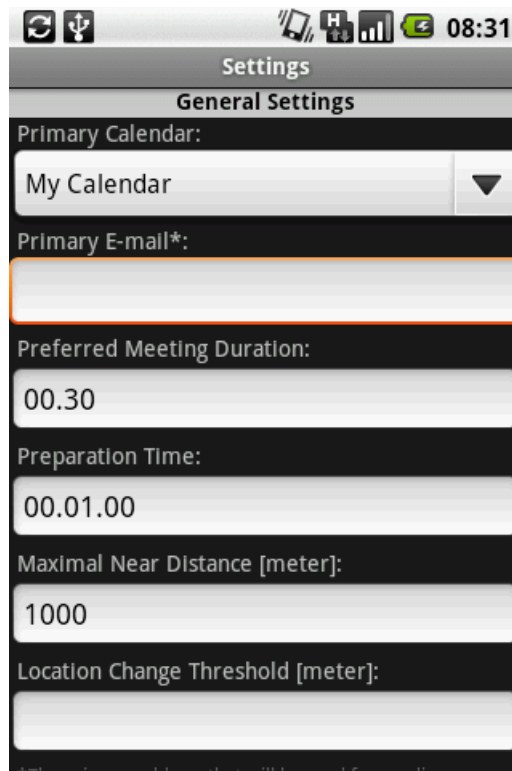


Figure 5.3: The settings menu of the SCAP application.

When an organizer decides to cancel a meeting that has not yet occurred, he/she sends a cancel message to all attendees. This is the only message sent from or to any peer(s) participating when cancelling. However, the organizer might decide to initiate a rescheduling attempt (section 5.3 gives a description about what a rescheduling attempt is). An attendee can request rescheduling by sending a counter proposal. The organizer “accepts” this request by initiating a new time negotiation process. Currently the organizer can not explicitly decline a counter proposal. However, if the organizer does not initiate any action, then the counter proposal is considered to be declined. To summarize, SCAP utilizes the following iTIP methods:

Method	Meaning
REQUEST	<ul style="list-style-type: none">• invite attendees• propose a new time for a meeting• confirm a newly proposed time• request free busy (and location) information
REPLY	<ul style="list-style-type: none">• reply to meeting invitations• reply to free busy requests
CANCEL	used by organizer to cancel meetings
COUNTER	used by an attendee to request rescheduling

5.3 The protocol for scheduling a meeting

Because the organizer books the meeting, he/she is the person responsible for the meeting actually occurring. In SCAP, this means that the organizer peer is the only peer that eventually learns everyone’s calendar and location information, as this is the only peer receiving such information. Thus, the attendee peers do not know about each others accepts or declines, unless informed by the organizing peer. Note that the only time that all attendees are informed (implicitly) is when the meeting has been scheduled[†]. In the following text in this thesis, the meaning of the word “postpone” is the same as the meaning of “rescheduling” - to move a scheduled meeting to a different time, often in the future.

5.3.1 Choosing the communication approach

Section 4.8 discussed two possible communication approaches: one where everyone shares his/her calendar with the organizer, and one where everyone shares his/her calendar with everyone else. In the evaluation part of the same section, a problem with synchronizing a calendar was discovered in the later approach. To solve this problem, peers would need to send additional messages to indicate that their calendar is out of synchronization (with the calendar view previously sent to others) so that the other peers would have the same scheduling parameters. However, because SCAP is going to run on a mobile device, peers can not respond to messages all the time. Thus, when the peer who’s calendar has changed, dies or is unable to inform everyone of its new calendar in time, then no peer know of a potential time collusion. In addition, this communication approach sent too many messages in

[†]For an organizer, a meeting is considered to be scheduled when each peer has accepted both the meeting invitation and the proposed time. Note that in SCAP, a meeting invitation does not contain any time proposal; its purpose is simply to inform the participants about other meeting details, to ask if they want to attend, and to ask their permission to share calendar information with all the people involved. An attendee learns about a scheduled meeting from the organizer’s confirmation message, that the organizing peer sends once it has learned about the successful scheduling of a meeting.

total. Therefore, SCAP will utilize the first communication approach, i.e., where the organizer is the proposer.

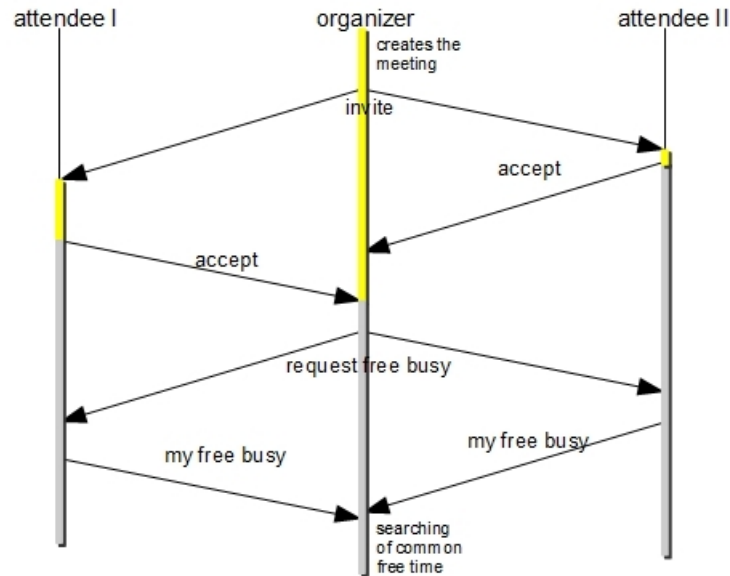


Figure 5.4: A communication example of the scheduling application, where the organizer requests for free busy information after that both attendees have accepted the meeting invitation. The colors of the vertical bars indicate what state the meeting is in for that user at the time. The meaning of meeting states is explained in section 5.4.3.1.

5.3.2 The time negotiation

Before a time negotiation can start, every attendee needs to accept the meeting invitation and the organizer needs to request and receive everyone’s free/busy information (see figures 5.4 and 5.5). Once the organizing peer has received everyone’s free/busy information, the search for common free time can start.

Note that, because a calendar server is used as a storage medium, each event must have a start and end time before being added to calendar. Initially, the organizer upon creating a meeting and for an attendee upon receiving an invitation for a meeting, this meeting is inserted right before the deadline even though no actual time proposals have yet been made. Also note that a meeting added to a calendar in this way may cause conflicts with existing events at the same time. Overlapping events are treated in a special way when free/busy information is created. More about this can be found in section 5.3.4.3. Because a meeting will be added right before the deadline in everyone’s calendar it eventually must be moved to a “better” time by moving it towards the past, if unplanned time is available. A meeting can be moved more than once; whether or not the meeting is to be scheduled the first time. For example, the organizer can actively reschedule a meeting to a later occasion, when he/she finds, that someone who must attend the meeting is sick.

A negotiation for a meeting time starts when the organizer peer has found a common free time. The first period found (in a sorted list with periods of free time,

starting with the earliest period) of at least the same length as the proposed meeting is a potential candidate, if the participants are “close” (see the definition of “close” in section 5.3.4.1). If the participants are not close, then only periods that start after the preparation time are considered, and the next available time with the highest priority is proposed. Section 5.3.4.2 describes in more detail what prioritized periods are. However, note that a meeting is always scheduled to start after the preparation time when postponed. When a peer receives a proposed new time request, it will automatically accept it, if this peer finds that no events in its calendar, visible for free/busy requests, overlap with the proposed time. By accepting such a request, the peer guarantees that it has reserved the proposed time period for this meeting - in the calendar this means that the meeting object previously added right before the deadline, is moved to the newly proposed time and that the user of this peer is considered busy for any future free/busy requests for the same time period.

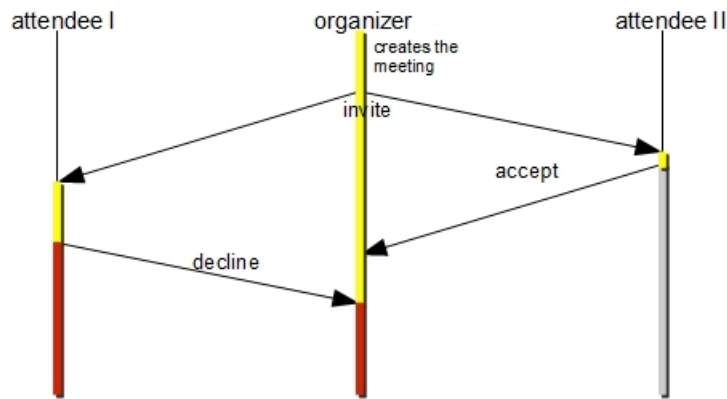


Figure 5.5: A communication example of the scheduling application, where one of the attendees declines a meeting invitation. The colors of the vertical bars indicate what state the meeting is in for that user at the time being. The meaning of meeting states is explained in section 5.4.3.1.

If a peer’s calendar returns events visible for free/busy requests that overlap with the proposed time period, then this peer replies with a “decline this proposal” message. Such a decline means that the organizing peer asks the declining peer to regenerate its free/busy information, in order to get the most recent view of its calendar. Note that because SCAP handles the proposals in the background, the proposal messages are handled directly when received. In the most typical case, the time passed between when the organizing peer has received everyone’s free/busy information the first time and that the organizing peer has found a new proposal after that it recollected the declining peer(s) free/busy information, is rather short. During such a short time it is unlikely, that any user’s calendar or location changes, hence it is not necessary to re-collect everyone’s free/busy information after one decline proposal message. However, as time passes, the probability of a change increases; at some point necessitating a re/collection of free/busy information from all the participants.

When the organizing peer has received the updated version, a new time negotiation is started. Figure 5.6 shows a possible negotiation scenario. The figure

also shows that the meeting is scheduled for the organizer, when every peer has accepted the proposal. The case that at least one accept message (to the current time proposal made) arrives at a time greater than the starting time of the meeting is not handled by the scheduling application. Once every peer has accepted the proposal, then the organizer may inform everyone in a request message about the confirmed and scheduled meeting.

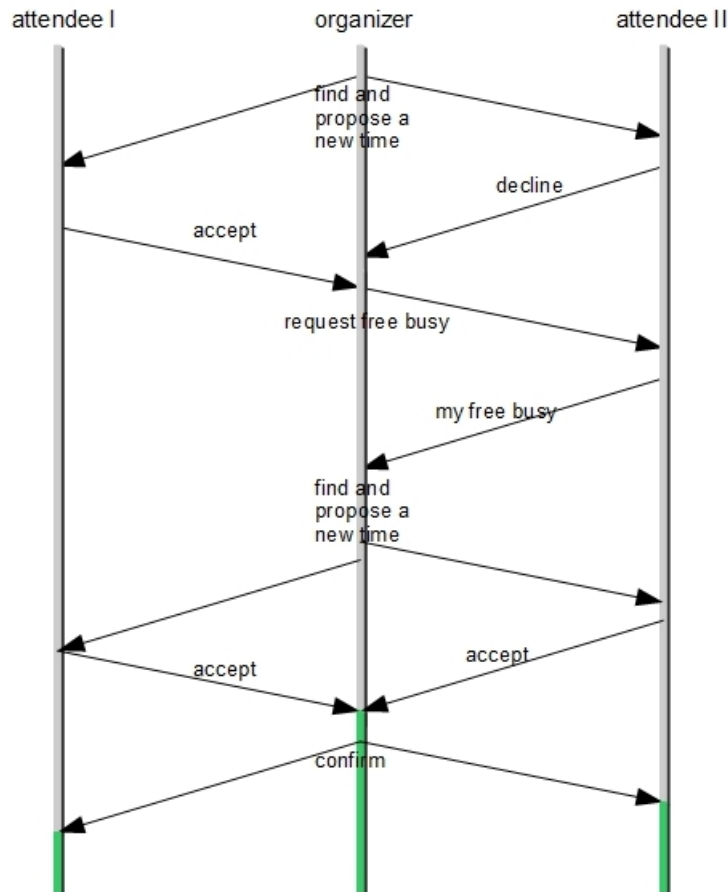


Figure 5.6: The figure shows how a time negotiation may look like between 3 participants after that every attendee has accepted the meeting invitation. The colors of the vertical bars indicate what state the meeting is in for that user at the time being. The meaning of meeting states is explained in section 5.4.3.1.

5.3.3 Rescheduling a meeting

As mentioned earlier in this section, a meeting can be rescheduled to a new time. Only the organizer can issue a rescheduling attempt (a new time negotiation), but attendees may propose a new time, once they have learned that the meeting in question has been scheduled. Figure 5.7 shows the activity for proposing a “new time”. This activity is identical both for the organizer and for an attendee. The functional difference is that a new time negotiation is issued directly, when the organizer submits the form. When an attendee submits the same form, a

rescheduling request is sent to the organizer. When the organizer accepts a rescheduling proposal or issues a new negotiation by himself/herself, all attendee proposals and their previous participation status are cleared, and the meeting goes to the pending state. From this moment on, every communication step made by any peer is equivalent to the initial communication steps happening right after the organizer has learned that everyone has accepted a meeting invitation. Technically, rescheduling means setting a meeting's preparation time to a proposed value and performing a new time negotiation instance with this new parameter.

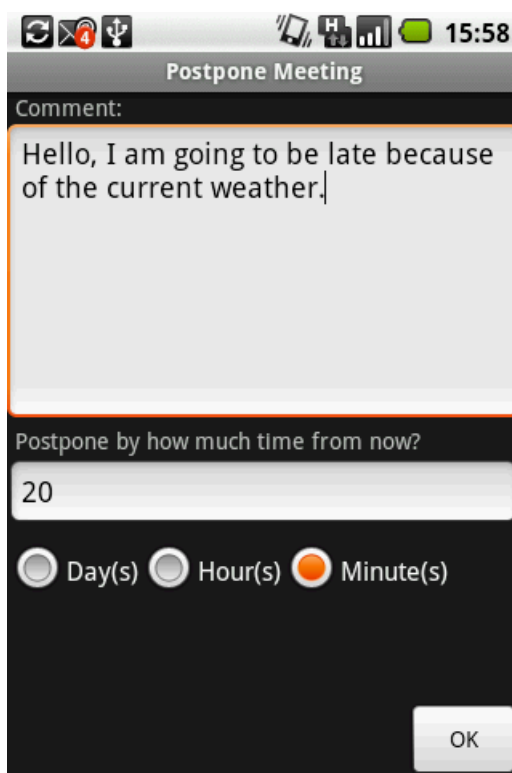


Figure 5.7: The postpone meeting view.

5.3.4 The elements of free/busy information

In SCAP there was a need to extend iCalendar's free/busy information mainly because the application should also propose a meeting place. Although no meeting place candidates are proposed in this implementation of SCAP, by scheduling a meeting "now", the users are given a "hint" relating to "here" or to a meeting place somewhere "close". To be able to give that "hint", the application needs to know about the current position of the participants. The positions will be propagated via the *Location Component*. The *Main component* contains free/busy information extracted from calendar events and the *Priority Component* contains user priorities for constrained scheduling. Section 7.1.1 describes how the existing free/busy supplementary information could be augmented in order to carry location information for events, and, using this information, how a more specific meeting place could be suggested.

5.3.4.1 Location component

The location component will help determine if it is possible for everyone to meet right away, i.e. if all participants are close to each other. Close means “within the radius of accuracy”, see figure 5.8. In the figure, each radius corresponds to the accuracy values in meters as returned by that user’s location generator. If the location generator returns a location with an accuracy lower than the maximal near distance set by the organizer (see the settings view shown in figure 5.3), then the radius of the circle for that user is set to the maximal near distance. The reason for this is to make it more likely for the application to discover a user being close.

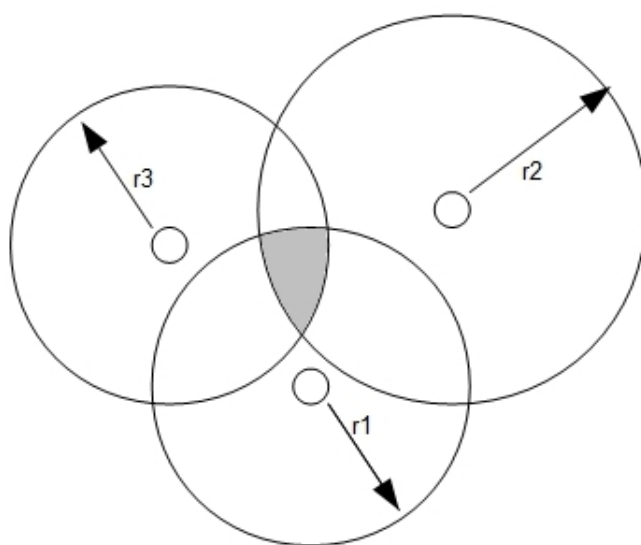


Figure 5.8: Each circle represents an area where one user’s true position could be. Here all users are considered to be “close” since each circle shares a common area with all other circles (filled in gray).

Besides the usual free/busy information (see section 5.3.4.3), another **VFREEBUSY** component was created to carry the location information. The location “stamp” is specified as a **GEO** property (latitude and longitude value pair) within such a component, together with the time stamp when this location object was created and the accuracy, in meters. Figure 5.9 shows an example of a location component. When such a component is created, a cached location stamp will be put into the **GEO** property, if this stamp is no older than ten minutes. Older values are updated by asking the mobile device to record a new location stamp. If a user’s device is capable of interacting/equipped with more than one location provider, all providers are asked for the current location and the value of the first returned location is taken. This may or may not be the best solution and needs to be examined further in the future.

Note that a **VFREEBUSY** component per definition is required to have a **DTSTART**, a **DTEND**, and a **FREEBUSY** property - thus they are all specified. Their values are chosen randomly, to pass the validation process prior to sending or upon receiving, and they are ignored at the receiver[‡]. Also note that the **ORGANIZER** property in

[‡]Adding extra data in the messages sent is a waste of both computational power and the Internet

a VFREEBUSY relates to the person whose free/busy information is requested, or, alternatively, to the person replying to a request for free/busy information.

```
BEGIN:VFREEBUSY
UID:20100114T073204Z-20100114@10.209.106.22
DTSTAMP:20100118T131612Z
RELATED-TO:20100118T131459Z@98.179.56.8
ORGANIZER:mailto:user1@host.com
METHOD:PREPLY
GEO;X-SCAP-TIME-STAMP=20100115T192742Z;X-SCAP-ACCURACY=
400.0:59.3800435791015625;17.99995528293609619140625
DTSTART:20100118T131459Z
DTEND:20100120T225959Z
FREEBUSY;FBTYPE=FREE:20100118T131459Z/20100120T225959Z
END:VFREEBUSY
```

Figure 5.9: An example of a free busy component with a GEO property. Additionally, the GEO property has an accuracy and a time stamp telling when this location information was measured.

5.3.4.2 Priority component

Another component of the free/busy information sent between SCAP peers is the component containing user priorities for scheduling a meeting. In section 4.5.1.2, an approach was introduced for how periods of time could be prioritized in order to filter meeting occasions. SCAP does not force the user to choose from among hundreds of possible meeting occasions, but the application supports scheduling priorities (even if there is not yet a GUI for setting priorities). For simplicity a globally known (by all peers) number of periods that can be given priorities to, is assumed (currently, set to a hardcoded value of 5). The first period always starts at the meeting's creating time (specified by the DTSTAMP property), while the last period always ends at the time stamp of the deadline. Each period is of equal length and all periods cover all the time from the meeting creation time to the deadline.

Although SCAP does not provide a GUI for setting priorities, a default list of priorities is always created and sent to the organizer. The organizer computes the ranked priority list (and associated periods) among all users and he/she then schedules the meeting in a time slot during the period with highest priority, if possible. The meeting is always scheduled as close to the start of the selected period as possible. If there is not sufficient free time available in the period with the highest priority, then the free time of the period with the second highest priority is examined, and so on. An example of a component containing prioritized periods is given in figure 5.10.

traffic generated by the battery-powered device. This issue is addressed in section 7.2.

```
BEGIN:VFREEBUSY
UID:20100114T073204Z-20100114@10.209.106.22
METHOD:REPLY
DTSTAMP:20100118T131612Z
DTSTART:20100118T131459Z
DTEND:20100120T225959Z
RELATED-TO:20100118T131459Z@98.179.56.8
COMMENT:This component contains periods that has been given
priorities either by the organizer directly or a calendar user agent
acting on behalf of the organizer.
FREEBUSY;X-SCAP-PRIORITY=1:20100118T131459Z/20100119T004759Z
FREEBUSY;X-SCAP-PRIORITY=2:20100119T004759Z/20100119T122059Z
FREEBUSY;X-SCAP-PRIORITY=4:20100119T122059Z/20100119T235359Z
FREEBUSY;X-SCAP-PRIORITY=1:20100119T235359Z/20100120T112659Z
FREEBUSY;X-SCAP-PRIORITY=5:20100120T112659Z/20100120T225959Z
ORGANIZER:mailto:user1@host.com
END:VFREEBUSY
```

Figure 5.10: An example of a free busy component containing prioritized periods. There are 5 periods given with their start and end times, respectively, to make sure that the receiving peer knows what periods are meant. The sender of this component (user1@host.com) has chosen to give highest priority (5) to the latest period (that will be ending when the related meeting's deadline applies)

5.3.4.3 The main component

This component contains free/busy information extracted from calendar events that are opaque and conforms to the iCalendar standard. iCalendar allows many different ways of specifying periods of time when the user is free or busy, but in SCAP, each FREEBUSY property contains exactly one period, thus either representing the booked time of an instance of a calendar event, or a free time between two such events or free time close to the start or end time of a free/busy request. Basically, the main component is similar to that shown in figure 5.10, with the exception that no priorities are specified and that each FREEBUSY property defines a free/busy type. The free/busy type value is either free or busy (note that BUSY_TENTATIVE and BUSY_UNAVAILABLE are two more precise busy values).

When free/busy information is generated, the value of the free/busy type is derived directly from the event that will be represented by the period of time in this free/busy property, unless this event overlaps with another event. In that case, the period start and end times represent the longest overlapping period between any subsequent events and the free/busy type is set to BUSY. If an event is visible for free/busy requests and has the status TENTATIVE, then the derived free/busy type value is BUSY_TENTATIVE. If an event is visible for free/busy requests and has the status CONFIRMED, then the derived free/busy type value is BUSY_UNAVAILABLE. Note that how SCAP determines the value of the free/busy type, is an implementation choice and there is not necessarily any convention for this choice.

5.4 Application overview

In this section, the reader will become acquainted with the application's fundamentals from the user's point of view. Some screenshots of the scheduling application have been attached in this section to guide this introduction.

A meeting object is always tightly coupled to its parameters, since not all of these parameters can be changed for an existing meeting. The meeting duration, summary, description, deadline, preparation time, and the list of attendees are parameters that can be entered. Figure 5.11 shows the time tab when creating a new meeting.

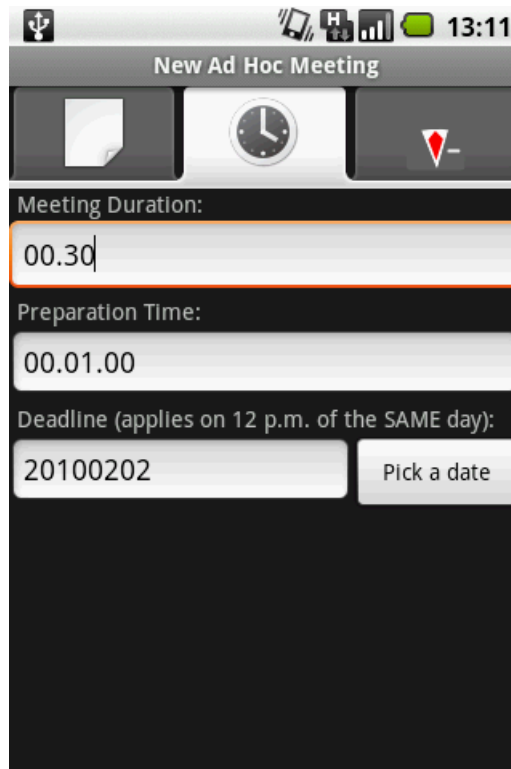


Figure 5.11: The time tab of a meeting editor in SCAP.

5.4.1 Meeting parameters

The meeting deadline will set an upper bound on the calendar end date that will be used in free busy requests; the meeting duration determines the exact length of a time period for which the meeting should be scheduled; the preparation time (together with the time stamp recorded when the meeting object is added in the calendar) determines the earliest starting time for the meeting. All time parameters in the time tab are mandatory, but they will be filled in automatically when a new meeting object is created. The default meeting duration and preparation time values are configurable in the settings (the SCAP settings were shown in figure 5.3). The deadline is by default set to the current date.

When viewing meeting details, there is an additional field with the start time of this meeting. Users should know if this time has been confirmed or not, hence

there will be an additional label for that purpose. `CONFIRMED` will appear in the label when the meeting is scheduled, otherwise it will say `TENTATIVE`.

Most meeting parameters are set once and can not be changed. However, users can alter the preparation time of a meeting when postponing (thus also changing the scheduled time). Section 5.3.3 describes how a meeting is rescheduled.

5.4.1.1 Attendee parameters

Attendees can either be added by selecting a contact (that contains a valid e-mail address) or entering the details of a new attendee. Only the e-mail address must be entered. Figure 5.12 shows the attendee view. When viewing the details of an attendee in an existing meeting object, the attendee view obtains an additional button in order to view this attendee's latest comment, if specified by him/her. This is a simple and useful feature to communicate with the organizer when declining a meeting invitation or requesting rescheduling.

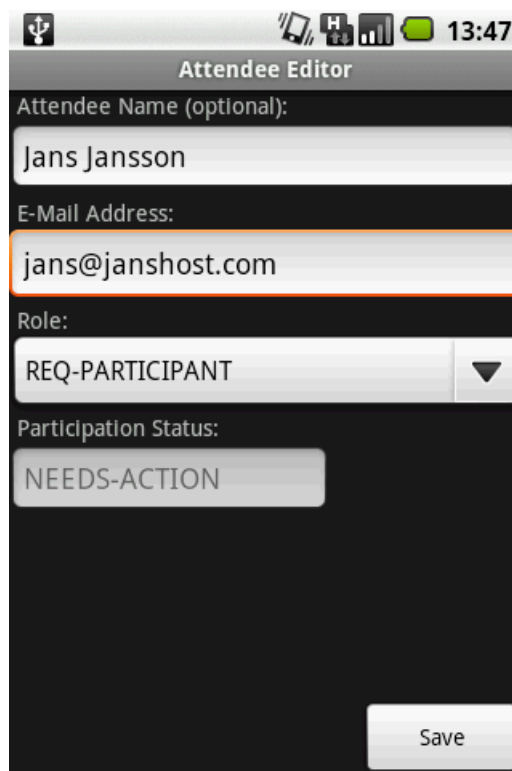


Figure 5.12: The attendee editor, when adding a new attendee. The value of the role of an attendee does not have any effect on the scheduling of the meeting with that attendee. The value of the participation status can not be set, as it is controlled by SCAP and will either be set to `ACCEPTED` or `DECLINED` when this attendee has replied.

The list of all attendees for a given meeting object looks similar to that shown in figure 5.13. The view for showing an attendee's participation status will only show an attendee's participation status if this attendee did not request to postpone the meeting. If an attendee has sent a rescheduling request to the organizer, both

this attendee and the organizer will be able to see the time stamp in the view on their devices. The time stamp indicates the earliest possible start time for a meeting when rescheduling.

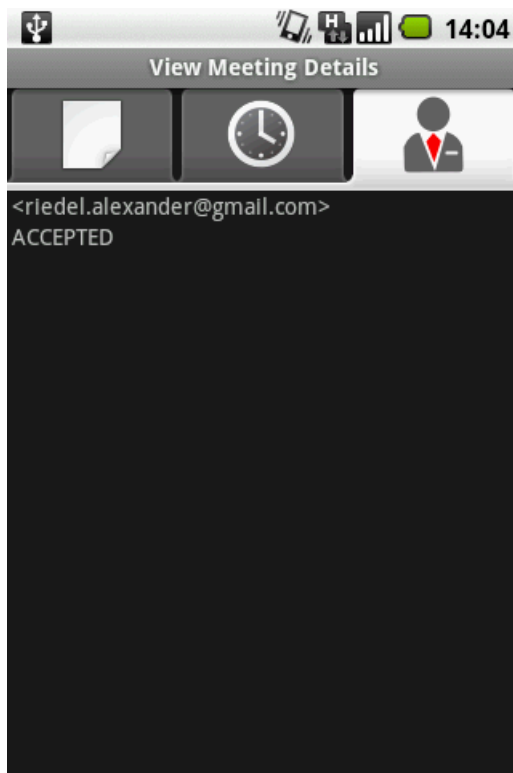


Figure 5.13: The list of all attendees in the current meeting showing their display name (= e-mail address and their name, if it exists) in the upper list item view and their participation status in the lower view. The participation status has the initial value `NEEDS_ACTION` and will be set to either `ACCEPTED` or `DECLINED` when this attendee has replied.

All meetings that have been added in the calendar using the scheduling application can be seen in the meeting list. The meeting list is also the “main” SCAP window since most important actions can be issued from here. Figure 5.14 shows the options menu of this main window. From this option menu users can navigate to SCAP settings, create a new meeting object, etc. The “Remove ALL” menu item actually was added as a “debugging” button, as in the process of developing SCAP, the application crashed many times when a user clicked on a meeting to view its details. Pressing this button clears from the current calendar all meetings in two clicks (the second click is needed to confirm the deletion).

5.4.2 Meeting list

SCAP does not offer a calendar-like view where events are rendered as rectangles occupying an area proportional to their duration. However, SCAP allows the user to change the view of their meeting list. Pressing the “change view” button displays a menu from which the user can set a status filter that all meeting must match in

order to be shown in the meeting list. The filter can either be “none” or one of the meeting scheduling states (the scheduling states are discussed in section 5.4.3.1). When all meetings are shown, the user can distinguish them by names (although they are not sortable by names in the current implementation) and the user can also recognize a meeting’s scheduling state by the colored vertical bar displayed to the left of each meeting. Figure 5.15 shows an example of a meeting list with a filter set to “none”.

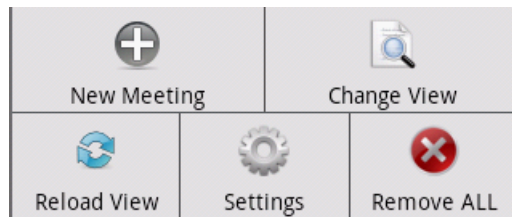


Figure 5.14: The different option menu choices, when pressing the menu button on the Android device, while viewing the meeting list.

5.4.3 Meeting scheduling states

A meeting can be in different states, see figure 5.16. Although all states are visible for all participants, not all transitions are seen by attendees. Moreover, the attendees do not necessarily have the same view of a meeting all the time.

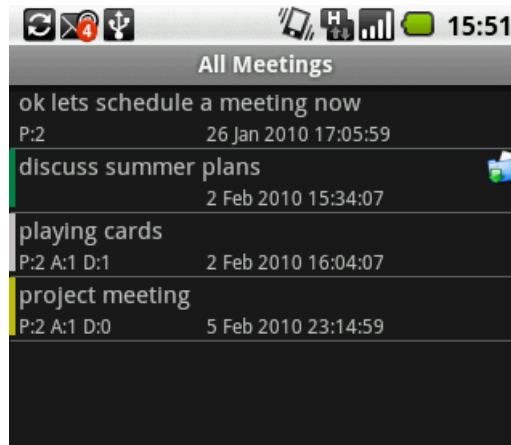


Figure 5.15: The meeting list view. Users can see the most important meeting details, like meeting summary and scheduled time, in each list item. The meeting state can be identified by the color of the vertical bar immediately to the left of each meeting, respectively. See section 5.4.3.1 for a detailed description of the scheduling states. The blue-green folder-like icon to the right indicates that the user of this device is not the organizer of this meeting. Organizers can also see a summary of the current participation statuses of all attendees: P stands for the number of participants, A for accepted and B for declined. Each list item also displays the meeting start time.

5.4.3.1 The scheduling state transitions

This section describes the meeting scheduling state transitions in the figure 5.16. If a transition's condition contains the words organizer and attendee in parenthesis, then this transition condition is different for the organizer and attendee. Transitions starting with "if not *conditional transition*" can only take place if the assertion taken from that *conditional transition* is false.

To stalled:

- S1 Organizer cancels the meeting
- S2 (Organizer) Someone declines the meeting invitation/(Attendee) I decline the meeting invitation
- S3 No free common time (of sufficient length) available
- S4 Current time greater than time stamp of the deadline

To pending:

- P1 (Organizer) Everyone accepts the meeting invitation/(Attendee) I accept the meeting invitation
- P2 The organizer issues a new rescheduling attempt

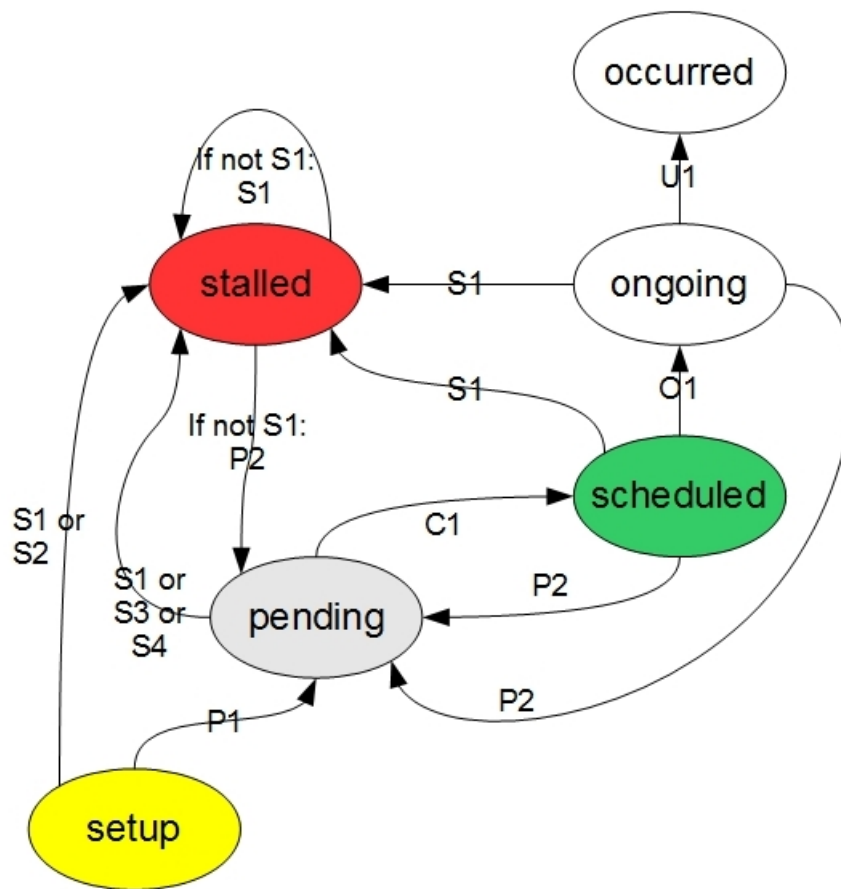


Figure 5.16: The meeting scheduling states and their possible transitions. For a detailed description of the transition conditions, see section 5.4.3.1. Once the meeting is created, it is in the setup state, while the final meeting state is occurred.

To scheduled:

C1 There is at least one common time slot available

To ongoing:

O1 Current time is greater than the meeting's start time and less than the meeting's end time

To occurred:

U1 Current time is greater than the meeting's end time

For an attendee, the view is rather simple: A new meeting invitation is shown in the state setup and the state changes to pending as soon as this attendee has accepted. The meeting will remain in pending state until this attendee has received a confirmation or a cancel message. A confirmation message changes the state to scheduled, while a cancel stalls the meeting. Note that a cancel message does not need to be sent to everyone, because the organizer can choose to kick out a participant by sending a cancel message to him/her. However, this is not yet implemented in SCAP, but is one of the desired features addressed in section 7.1.2. If an attendee declines an invitation, this basically means that he/she kicks

himself/herself out of the list of attendees, thus the meeting will be stalled for that attendee (and for the organizer getting this decline message). An attendee requesting rescheduling does not affect his/her or the organizer's current meeting state. If any future state transitions occur or not after a rescheduling request, depends on whether the organizer accepts this rescheduling request or not.

The organizer is the only person who cares about accepts/declines and the only person who knows everyone's free/busy information. Therefore, transitions S2, S3, and S4 (in figure 5.16 on page 78) are only seen by the organizing peer, so that this peer can react to such events. For example, when there is no common free time available, the organizer will see a meeting in stalled state. Meanwhile, all of the attendees see the same meeting in pending state, because all of them have accepted the previously sent invitation. While in stalled state, the organizer has the choice of cancelling the meeting or creating a new meeting object with different parameters. Section 7.1.2 describes an extension to this set of alternatives.

If an attendee declines an invitation, the organizer and the declining attendee will see the meeting in stalled state, no matter what the other attendees, if any, reply. In this situation, the organizer can cancel the meeting, or reschedule it for a later time. This also explains why the transitions stalled to stalled and stalled to pending are possible for the organizer only if the meeting has not yet been cancelled.

5.4.4 Notifications

In SCAP, users can not be reminded of meetings starting at a predefined time because time alarms are not managed by SCAP - this is up to the calendar system that is used by the user. However, SCAP can notify a user when the application has received an invitation message, when a meeting has been (re)scheduled, or when this user is an organizer receiving a request for rescheduling. All notifications will display a specific icon on the action bar of the Android device and the users can navigate to the SCAP meeting list view, alternatively, to the meeting details view, by opening the notifications menu and then clicking on the notification icon. Opening the details of a meeting represented by a notification (if such a notification is available) will "acknowledge", i.e clear, this notification.

For example, a user might have two invitation notifications (seen as a single invitation notification icon with the number "2"). When this user clicks on the notification icon, a list of all invitations will be opened and these notifications will be displayed. After the user views the details of the meeting that is associated with one of these notification icons, the notification number decreases by one. Subsequently, clicking the notification icon again will open the details of a new (unviewed) meeting invitation.

In SCAP each notification type has its own vibration pattern, so that users can recognize what the application is trying to "tell" them upon receiving a message. However, it is not possible to configure which notifications are indicated with a vibration. That is, all notifications are active by default and the user can not choose if he/she, in addition to a specific icon to be appeared, would also like an audio file to be played or would like to disable the phone's vibration. The limited time period for this thesis project has meant that notifications are not configurable, but this should be possible to do in the future.

6. Experiments with the application

This chapter describes how the application developed in this thesis project was evaluated. Tullis & Albert's book [33] was a good help in choosing the most appropriate method for carrying out the evaluation task.

6.1 Evaluation method

To select test users, both Telepo employees, student colleagues and friends of mine were asked to participate in the survey. Telepo employees were assumed to represent "business" users. Note that it is not in scope of this thesis project to investigate user groups*, to select representatives from groups, or to see how group affiliation might affect the evaluation results. The results of the survey do not consider potential user groups. However, each test user was asked to provide information about himself/herself in order to determine potential group affiliation. For details of the information that was gathered, see appendix A or the evaluation results section (section 6.3).

Although there might be a correlation between a user group and the outcome of the survey, this is not investigated further in this thesis. This report simply assembles the test results and the test user distribution to gain some simple statistical knowledge from the survey. Everyone referring to this survey, for example, to discuss the survey's value and validity, might be interested of this additional information.

To run the tests, at least 10 users are needed to give reasonable results concerning most of the evaluation questions. However, no upper limit on the number of test users was set, as the actual number of users depended mostly on the time available in this thesis project. However, more survey participants would produce higher accuracy in the statements and conclusions that could be drawn from the survey.

6.1.1 Evaluation questions

Almost all questions[†] are answered by selecting at most one of the 5 options (in a Likert scale) for that question. Each option represents a rating of what users think about something. The options the user has appear as a scale immediately after the

*Relevant user groups might consist of (1) users who frequently use calendaring software in order to book meetings, or (2) particular users who frequently use their mobile device in order to book meetings. Other groups might include users that view/edit their calendars frequently, but do not book meetings at all or often.

[†]The evaluation questionnaire can be found in the appendix.

question. If a question did not have a Likert scale after it, the user was free to a textual answer to this question.

There are many different kinds of usability evaluations, but in this thesis the main focus was on user satisfaction. However, other usability areas, such as comparing this prototype application with similar products are also addressed in the survey.

The highest satisfaction usability scenarios (of this thesis project) can be generalized by saying which of the following application goals were met:

- Reduce the time/effort needed to plan a collaborative task
- Reduce unnecessary user interaction by customizing automatic actions

Not all questions can strictly be categorized to test either of the application goals (given in the above definition). Additionally, there are many ways that these goals could be achieved. The questions asked focus on different topics and therefore, have their own degree of uniqueness. Some questions ask the users if and how the user would like to affect the scheduling process, while other questions concern location-specific application extensions. Below is an overview of the evaluation that was made in order to test how the application meets the goals:

- Affecting the scheduling by additional personal constraints
- Affecting the scheduling by additional global constraints, which can be seen as similar to including additional meeting parameters
- Decreasing the number of mandatory meeting parameters
- Making the scheduling succeed

Furthermore, the reader might have noticed, that the evaluation does not only focus on the application “as is”, but rather some questions refer to potential application extensions, in addition to the features that were implemented. The application demonstrates a proof-of concept that enables users to evaluate its advantages and disadvantages. The application should also serve as a point of comparison both to similar existing applications and to applications that could be derived from this or a similar application.

Note that the test users probably would have answered the questions about application extensions in a different manner if they would be given more time (than planned) to use the application. Also note that the allocated time for each test user is sufficient to answer the questions about application extensions and can still produce credible results. This is because each test user was made aware of the test scenarios’ outcome in case there was not sufficient time to complete a particular test scenario and that all test scenarios demonstrated all the desired SCAP’s functionality to answer such questions. If a question required the test users to know something not covered by SCAP’s introduction or not shown by the demonstration of SCAP, then the required information was included purposely into the context of the question.

Some questions might be interpreted as asking about the application’s appearance. However, the scope of this thesis (evaluation) was not to capture the user’s perception of the application’s graphical design, the number formatting, layout-specific, or other usability issues. The evaluation does not cover the performance of the application nor how fast users could complete a specific task.

6.1.2 The evaluation step-by-step

- A) Most of the test users (about 90 %) were not able to test the application beforehand. However, all users were given a short description[‡] to introduce the application features some days before the planned test date.
- B) The users were invited to a meeting with me that was estimated to take approximately 1 hour per user.
 - a) During the meeting's first part, the users were asked if they would like to use their own (Google or Exchange) calendar during the testing. If so, then the necessary settings were applied to the Android phone. If they did not want to use their own calendar, then a test account was created utilizing Google Calendar.
 - b) During the meeting's second part, the application was shown and an introduction to the Android phone was given. The features of the Android phone utilized by SCAP were explained. Depending on how carefully a user has read the SCAP introduction, this part took 15-25 minutes.
 - c) After the introduction, the users were given instructions to run the test scenarios described in section 6.1.3. If there was not sufficient time to run all the scenarios, their outcome would be explained.
- C) The users were asked if they preferred to fill in the questionnaire on paper, or answer the questions on a computer. After that, the questionnaire was handed out, or sent to the user by e-mail, and the answers registered by myself into a simple database.

6.1.3 Testing scenarios

In the following test scenarios, a meeting should always be booked with the same two participants, between the test user and myself. For all scenarios, the organizer's role was taken by the test user and all the actions made by the attendee (me) were explained.

6.1.3.1 Scenario I: Book a meeting

One of the attendees should decline the invitation. When the organizer receives the "decline", he/she should try to schedule the meeting anyway.

6.1.3.2 Scenario II: Book a meeting and ensure that it is scheduled

In the following two different sub-scenarios, an attendee should submit a "postpone request" to the organizer, then the organizer can accept or decline/ignore the proposal. In the first sub-scenario, an attendee should submit a request indicating

[‡]The SCAP introduction was about 5 pages long and it contained many screenshots (see appendix C). Note that this "manual's" purpose was not to explain every single detail of the application, but rather to demonstrate what the application looks like and how to navigate between the application's views. Although test users might have guessed the results after a certain action accomplished by the user/the application, no direct hints were given in the manual for how the scheduling process actually works from an interaction point of view.

that he/she is going to be late, thereby asking for permission to postpone the meeting so that this attendee arrives in time (according to this attendee's estimated time of arrival). In the second sub-scenario, an attendee should submit a request indicating that he/she can not attend this meeting at the scheduled time, but would like to participate in the meeting at any time starting from an arbitrary chosen time before the deadline.

6.1.3.3 Scenario III: Scheduling a meeting when all participants are “at the same place”

Make sure that all participants are “close” to each other (i.e. in the same building, the same room, same street, etc.) and also have enough free time so that a meeting with duration x minutes can be scheduled. Schedule this meeting. The participants should get a notification that the meeting could start right now (implying that the meeting place must be close to all participants).

6.2 Expectations

It is hard to expect specific results before conducting any tests, but some initial rough estimates will be given in this section.

How users will answer the evaluation questions depends on their background and experience. For example, people who never have used a scheduling program before might answer differently from those who have. The outcome of the survey might also depend on what scheduling program a user has experience with, how often the user plans meetings using electronic calendars, what tasks they do on their mobile devices (how frequently they used this device), and last but not least, the user's personal style.

However, as the application developer, I had some expectations about how users would react. These expectations were due to the objectives that were set at the beginning of this thesis project (see section 1.4). Optimistically, I expected the application to achieve its goals, thus answers to some questions were expected to be more or less obvious.

For example, given the material presented in the background chapter and because the scheduling application is based on the iCalendar format, SCAP should not differ (much) from other similar applications. Likewise, I expected most of the users would not want to have mandatory meeting parameters, if any at all, provided that the application could provide values for these parameters. Although, some users might provide all the parameters manually, because a meeting needs to be planned exactly as they want. For these more picky users is there any point in automatic scheduling?

How “automatic” is scheduling allowed to be? That probably depends on the scheduler's capability and the parameters that constrain the scheduling. Generally, no user minds setting additional global scheduling parameters, as long as there are some default values for the mandatory parameters. I expected that most users would like to have many parameters, so that as much (context) information as possible could be input and “attached” to a meeting object. This would lead to a more feasible schedule, whether scheduling automatically or not. My thought was: Who does not want to have a personally tailored schedule?

Do you often use some kind of calendaring software
in order to view or edit your calendar?

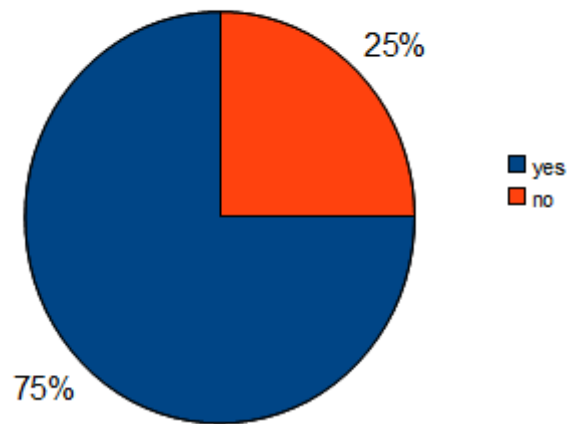


Figure 6.1: The first question for determining group affiliation of test users.

6.3 Evaluation results

In total 15 users have participated in the survey. According to the two last evaluation questions shown in appendix A, the users were asked to provide information about themselves in order to determine their group affiliation. Figures 6.1 and 6.2 show the results of these questions.

The results presented in the following are not obvious, hence an explanation will follow. As previously was mentioned in this chapter, the evaluation questions are of different nature and their nature basically determines how the results will be presented. Questions 1-4, 10 and 12 (group A) refer to the current application implementation, while questions 5-9 and 11 (group B) ask for usability of application improvements. The collected results to questions having a score value (the following paragraphs explain what a score value is), i.e. questions 1-6 and questions 10-12 can be found in the tables in appendix B. The results to the remaining questions, i.e. questions 7-9, are presented in section 6.3.1.

6.3.1 Group A questions

To present the results to questions in group A, each question was defined to have a maximum score, which is calculated in the following manner. Because all such questions have a Likert scale, there is always an alternative that should give the most score and that is considered to be of “highest value” to this project, if chosen by test users. The highest value was defined to be 5, because of the 5 alternatives. Thus, the maximum score for each question in group A is 5 times the number of users: $5 * 15 = 75$. The second “best” alternative gave 4 points, the third best 3, and so on.

The score calculation pattern applies to all questions in group A, unless otherwise specified. Because question 12 gave 3 answers instead of 1, its maximum score was the mean value of all these answers. The first answer’s maximum score is 75, as previously defined, while the other answers each have a maximum score of 45 each.

This is because the best answer is the option in the middle (which was given a value of 3, hence $3 * 15 = 45$ points), while the left- and right-most alternatives gave 1 point, and the remaining alternatives gave 2 points for each of them and for each user.

Do you often use some kind of calendar software in order to book meetings?

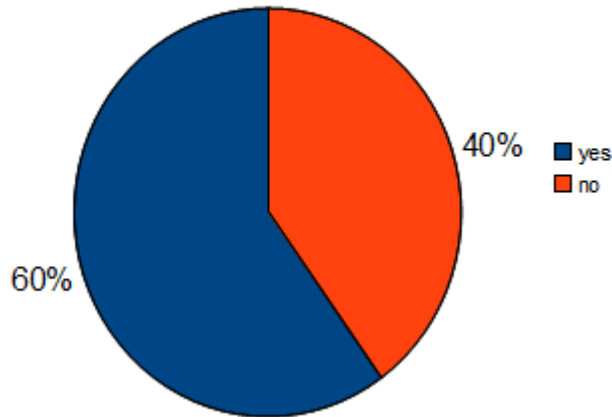


Figure 6.2: The second question for determining group affiliation of test users.

Questions 3 and 4 contributed to the achieved score of the first question. This is because all these questions are related and their mean value gave a better estimate in order to answer the first question. The value of question 2 was not so obvious to determine since it was unclear which of the alternatives was “best”. The definition of “best” depends on what the test users think about the SCAP features different from features of other calendaring software that the users have experience with. However, questions 11 and 13 were really useful and their mean value was a fundamental to answer the question whether it is good or bad in case SCAP would be perceived (much) differently from other scheduling applications. Figure 6.3 shows all the achieved score of the questions from group A, while the tables found in appendix B show the “raw” data collected from users.

Overview of all the questions' achieved scores, sorted in descending order.

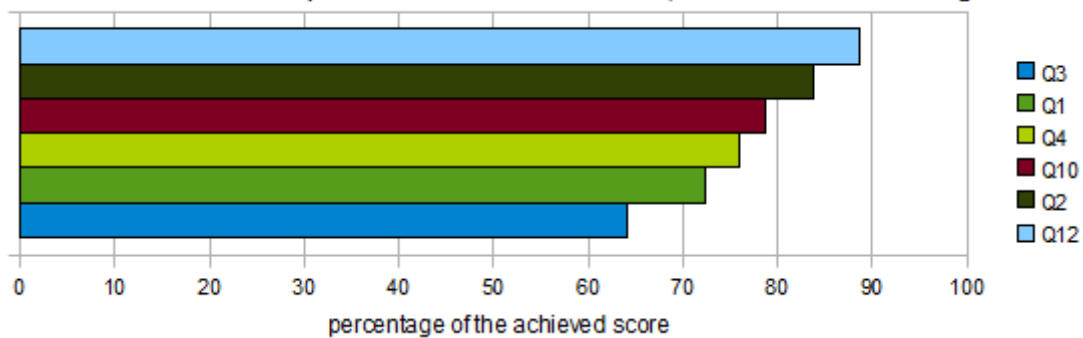


Figure 6.3: The achieved score.

6.3.2 Group B questions

Some of the questions in group B had Likert scales, while others could be answered freely. Figure 6.4 shows the results from question 5. How useful the improvements proposed in questions 6 and 11 are can be seen in appendix B.

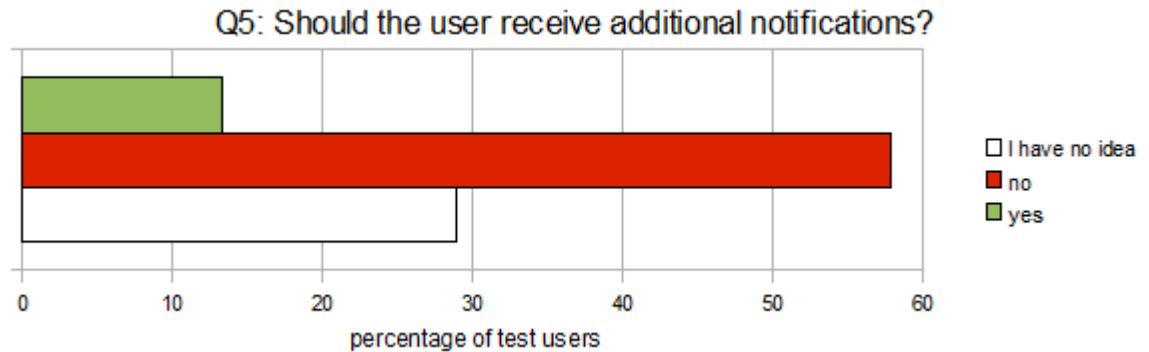


Figure 6.4: This figure presents the results acquired from question 5 in the survey.

In order to reduce the 5-point Likert scale into the three alternatives shown in figure 6.4, the following method was applied. Each user that completely disagreed (to receive additional notifications), was considered giving one point to the no-answer. Each user that marked the point immediately following the “completely-disagree-point”, gave 2/3 point to the no-answer *and* 1/3 point to the “I have no idea answer”. The point in the middle of the scale gave exactly one point to the neutral answer. The contributions to yes-answers are calculated in the same manner as for no-answers using the opposite site of the Likert scale.

Note that this calculation is slightly biased towards the two extremes (yes- and no-answers). However, this does not make the evaluation invalid - the grade of biasing is equal for each direction. The motivation to this method is elimination of the neutral answer because few conclusions can be drawn based on such answers.

In the following subsections, the most important results for questions 7-9 are presented. The first item in list is most significant for that question and the following items are at most of the same importance to users.

6.3.2.1 Results from question 7

- (1) let the users modify arbitrary meeting parameters
- (2) try rescheduling the meeting with a subset of participants
- (3) suggest a parameter that should be changed to increase the likelihood for the meeting to be scheduled
- (4) try rescheduling the meeting at a later time
- (5) like (3) but with the addition to see if it really is possible to reschedule by rescheduling the meeting

6.3.2.2 Results from question 8

- (1) personal off-hours
- (2) personal preparation time
- (3) constraints that are based on who will attend the meeting

6.3.2.3 Results from question 9

- (1) possibility to select required or optional participants for a meeting
- (2) meeting duration margin by defining a minimum time required for a meeting and a an upper bound for a preferred meeting duration
- (3) allow meetings to be recurring

7. Conclusions and Future Work

Embedding collaboration through the contextual environment of a device is a useful feature and it suggested some work that unfortunately did not fit into the scope of this thesis. This chapter summarizes some ideas and extensions that have arisen in the course of this thesis project. To see what features have shown to be (most) important for test users, see section 6.3.2.

7.1 Application improvements

7.1.1 Suggesting a meeting place

Although the scheduling application is able to schedule a meeting when all participants are close to each other, that is, at a meeting place somewhere “close”, this “meeting place” may be too ambiguous. Therefore, the scheduling application could make use of events already present in the calendar to suggest a more specific meeting place*. Several improvements may be necessary in order to cover the most general case: a meeting place might be suggested based upon planned or historical events, while a meeting’s participants might or might not have such information in their calendars. Section 4.3 gave an extensive introduction related to this improvement.

7.1.1.1 The “how to get to X” - feature

In addition to suggesting a meeting place, SCAP could be extended so that a description for how to actually get to a meeting place is given. To facilitate collaboration, SCAP should propose different alternatives of how to get to a location specified in the next *ad hoc* meeting in the meeting list. There can be several ways to travel to a pre-defined location, e.g. by going, a car, bicycle, or by public transport.

- Going
- By car
- Bicycle
- Local traffic

One might even think further and enhance the “how to get to X”-feature to be context-aware of the average speed with which a user is moving, for example when

*In order to suggest a meeting place, the participants need to send their event location together with the free/busy information for the same event. SCAP already implements the necessary method for generating free/busy information from a calendar. Should there be events having their geographical locations specified, this location information (the `GEO` property and value pair) will be copied into a separate free/busy component for each event.

traveling to a meeting. Knowledge of the user's speed could make it possible for the application to guess the type of transport currently being used. Clearly, that would help the application to provide a better service to the user, because the application could itself select the appropriate type of transport as input. However, a simpler approach is to let the user choose the transport type available/preferred.

7.1.2 Enabling changes to the meeting details

Currently, when a meeting gets stalled in one or the other way, the organizer can not change the meeting's details. If the scheduling fails, it might be time-consuming to manually re-enter most of the details from the stalled meeting in order to create a new meeting object with different details. There should be an easy way to at least change the deadline, meeting duration, and kick out or add participants.

All of these features are rather easy to implement in the existing SCAP application. If a meeting is stalled because the automatic scheduling has failed, the organizer should be allowed to change this meeting's details. Depending on what the organizer changes, the following could happen (in the same order as mentioned): participants who are kicked out should be sent a cancel message; old participants should receive an updated version of the meeting object; the organizer should ask new attendees to attend and await their acceptance messages; and a time negotiation should start.

7.1.3 Customizing notifications for meetings

In the current implementation of SCAP, the user can not customize meeting notifications (see section 5.4.4). That is, every type of notification is always active and a user is notified by one specific icon appearing on the device. Not all users want to receive all notifications, while some users want to change the way that they are notified. For example, playing a customized audio file, phone vibration, or other indication of a particular type of notification is desirable.

7.1.4 Additional meeting parameters

7.1.4.1 Meeting location and *ad hoc* meetings

During the evaluation it was observed that a meeting object could contain more and more parameters. The meeting location is one of the most desired parameters, but SCAP would need to be modified to take advantage of this parameter. Users could be asked, if they want the meeting to be *ad hoc*, if possible. Alternatively, a meeting could take place at a location defined by a user, if it is not possible to schedule the meeting at the "current place". SCAP should be able to handle both cases.

The survey has also shown that test users would like to know when all the meeting participants are close. It was a bad idea to rely on SCAP's indication of that the participants are close (in this case, SCAP would try to schedule a meeting at the current time). However, in the worst case, if the participants are close to each other and if the earliest possible time to schedule a meeting is at least as big as this meeting's preparation time (because of unavailable time of some of the participants), then with the current indication of closeness, SCAP users would not be able to know that they are close.

If the users would like the meeting to be *ad hoc* and all the participants are close to each other, then SCAP should provide meeting place coordinates. In addition, there should be a way for users to see how close the other participants are to the meeting place. The meeting place coordinates of an *ad hoc* meeting could by default represent a geographically centered location between all the participants.

7.1.4.2 Scheduling period priorities

In section 4.5.1, an approach was presented as to how meeting occasions could be presented to users so that they could choose suitable occasion(s), if desired. According to the subsections of 4.5.1, an alternative approach was to let users affect the scheduling by setting priorities on the “period level”. In this simple approach, users can not “pick” individual meeting occasions and SCAP sets priorities on a constant amount of periods for a user, each time this user generates his/her free/busy information. Currently this is the only way that meetings are automatically scheduled. The following improvements seem to be desired:

- A GUI is needed in order to display the number of meeting occasions.
- A GUI is needed in order for the users to set priorities.
- A user should be able to select if he/she wants the meeting to be scheduled automatically, for each meeting created or invitation received.
- The user should be able to set his/her own default priorities, if automatic scheduling is enabled for that user. SCAP already implements a method for sending priorities in a free/busy component and determining the period with the highest priority.

7.1.4.3 Reminding the users to answer invitations

In order to speed up the scheduling, invitees must quickly reply to meeting invitations. Furthermore, it is sometimes necessary to remind a user to reply, particularly when the deadline of a meeting is close. If people want to meet, they should accept the meeting invitation as soon as possible, because that enables *ad hoc* meetings to more readily take place. Additionally, a person who wants to meet as soon as possible may find it annoying if the other participant(s) do not quickly answer a meeting invitation.

SCAP could be programmed to remind users of meeting invitations that were received a long time ago, if desired by the user. Compared to other invitations, older invitations and invitations with a closer deadline should get higher priority for reminders.

Another way of reminding a user is by re-sending the meeting invitation. This way, the organizer either manually or by configuring automatic meeting invitation retransmissions could remind the participants to reply.

7.1.4.4 Introducing scheduling costs for meetings

A meeting scheduling cost represented the ease of (re)scheduling a meeting. For example, the more participants a meeting has and the longer its duration, the harder

it can be to find a commonly available time. A meeting's location also affects the ability for a meeting to be scheduled, because the participants need to travel to that place; some of them might be able walk, while others might need to take a plane. Thus, a meeting place implies a travel cost for some or all the people invited to the meeting. In addition, there may be other additional costs for a meeting, such as renting a conference room, food and beverages, administrative and technical support, or other costs. The scheduling cost of a meeting might also depend on how close to its deadline the meeting is scheduled and the penalty if the deadline is not met.

When all meetings in a calendar have a scheduling cost associated with their scheduled time, then the scheduling costs could be compared. The ability to compare meetings is useful both for a person participating and for this user's scheduler. An advantage is that people with a busy schedule could attend the "more important" meetings because other meetings could be rescheduled more easily. However, individual people might also have individual preferences as to which meetings they want/need to attend.

When SCAP is able to propose a meeting place, or perhaps has an additional meeting parameter (the meeting place) that the the users might fill in manually, the scheduling cost could be calculated. With this feature enabled, SCAP could even be programmed to rearrange meetings in order to produce a more feasible schedule for one or more individuals.

7.1.4.5 User defined alarms

When a meeting is scheduled, SCAP users will be notified. However, this notification might not be sufficient because meetings can be scheduled at a different time than the current time and users might want to receive an additional notification at some pre-defined time *before* the meeting starts. Users could for example check a checkbox when creating a meeting if they want to be notified before the meeting starts.

Notifying users of a meeting's starting should be straight-forward to implement, as most alarm functionality is already implemented in the device. To notify a user, a VALARM object has to be created and added to the current calendar - the triggering of the alarm is provided by the native calendar application. In addition, users can "snooze" alarms.

7.1.5 Supporting multi-device users

One motivation for using e-mail messages as a communication mechanism was because the important information about the scheduling state would be forwarded via e-mail servers. Thus, multiple SCAP agents serving the same user can read this information enabling the agents to maintain a consistent view of the calendar of events. SCAP should be modified so that the scheduling application could be running on a PC in conjunction with another SCAP agent residing in an Android mobile device.

A big advantage of running the scheduling application on a PC is its capability of running a more complex scheduling algorithm. If SCAP is run on a server machine, then the SCAP agent on a mobile device could instruct the PC agent to execute a

specific task, thus hopefully saving battery power on the mobile device. Furthermore, the agent on the server machine could be available 24/7.

7.1.6 A better meeting list

All meetings that were booked with the scheduling application are displayed in a list view (see figure 5.15 on page 77). The meetings shown should be sorted in some fashion, as it is confusing for users to see many meetings without any particular order. In particular, it is hard to recognize impending scheduled meetings.

More general, the application should display the most important meetings at the top of the meeting list. What is most important might be up to the user himself/herself, but meetings that start soon should be shown first. Thus, delayed users could easily see and select the most important meeting, for example, in order to postpone it, skip a less important meeting to attend the important meeting, take a taxi rather than walking, etc.

The meeting list could be further customized. The evaluation showed, that SCAP users organizing a meeting had trouble finding the same meeting object when they viewed it after a rescheduling request has arrived. Of course, a potential cause for this problem is that meetings are not shown in any particular order in the meeting list. Moreover, once the notification signaling a new time proposal is cleared, there is no additional information in the meeting list to show such meetings. One simple solution to this problem would be to display the icon for a “propose new time notification” in the menu item for the associated meeting. This icon could be kept displayed until the organizer has postponed/rescheduled this meeting.

7.1.7 A better proposal manager

When the organizer of a meeting has received one or more meeting requests, he/she is able to select which proposal is to be accepted by selecting the proposing attendee. Once this is done, all proposals are deleted from this meeting, and they can no longer be selected. During the evaluation, some users found it inconvenient to “discard” all the remaining proposals by default. The example below shows why this is not always desirable.

Consider Alice meeting with Bob and Carol; Alice was the organizer. On her way to the selected meeting place, Alice receives a message from Bob indicating that he will be delayed 10 minutes. Shortly after that, Carol sends a message to Alice that she will be 20 minutes late. Based upon these two messages, Alice decides to postpone the meeting by (at least) 10 minutes, thus to start without Carol. Case I: the rescheduling succeeds, but the meeting is postponed by 30 minutes due to Carol’s busy schedule. Case II: the rescheduling succeeds and the meeting is postponed by 10 minutes.

In the first case, it is appropriate to delete Carol’s proposal as, according to her estimated time of arrival, she will make it to the meeting in time. In the second case, Carol’s proposal should not be deleted because: (a) Alice might decide to reschedule the meeting to a later occasion, thus enabling Carol to attend from start, and (b) Alice might need the information about which participants are late (and by how much) in order to make her decision about what to do.

Moreover, the organizer should be provided with additional information when he/she selects a proposal to accept. Currently, when the organizer has received many time proposals and wants to meet everyone he/she needs to figure out what time proposal would postpone the meeting the most and who has proposed it. It is hard to see such a time proposal without the proposal list being sorted. It may be necessary to sort the list of participants and their proposals, or mark such a proposal so that it can be recognized quickly.

7.1.8 Optimize the “schedule now” feature

When SCAP schedules a meeting as early as possible, in most of the cases, the meeting will already have being started when users receive the notification of the scheduled meeting. During the survey, some users got a bit confused about receiving such notifications - as the devices clock indicated that such users are late to the meeting. However, practically the meeting had not even started yet.

The problem of scheduling an “already started meeting” exists because of the communication delay between the peers and all peers relating to the creation time of the meeting object measured at the organizer’s device. While it may be hard to estimate the average delay of the e-mail servers, the problem of late scheduling can easily be mitigated by introducing another scheduling constraint. This time constraint could be hardcoded to have a value of for example 10 minutes and SCAP should not schedule a meeting earlier than the meeting’s creation time plus 10 minutes. One might go further and each time a communication takes place between SCAP peers takes place, measure the communication delay between all the peers and readjust this global time constraint to the average of all of the measured delays.

Another aspect related to this issue is that even though user’s might be “close”, their minimal traveling time might be large, depending of the definition of being close. Moreover, some users might simple be too busy despite that they are shown to be available according to their calendars. It may even be important to consider the differences between the organizer’s and the attendees’ clocks. Therefore, users should be able to provide SCAP with a personal preparation time constraint.

7.1.9 Remove-from-calendar issue

Currently, if an invitee deletes a meeting invitation, this invitation is removed from his/her calendar. But does that mean that this user has declined the invitation? Unfortunately for SCAP, the organizer of this meeting will not be able to discover that an invitee has removed the meeting invitation **and** that this invitee can not accept or decline this meeting invitation in the future. Because the organizer has no way of reminding this invitee to reply, for example by re-sending the meeting invitation, the scheduling protocol will be live-locked.

The easiest way to mitigate this live-lock is to disallow users removing the meeting object without that they decline to attend this meeting.

7.1.10 The issue about late proposals

Section 5.3.2 stated that SCAP does not handle late proposal and proposal accepts properly. If the current time proposal proposes a time greater than the current time

(of the device's clock), then such a meeting should not be scheduled - it should become stalled. Moreover, the reason why the meeting stalled should be given to users.

An additional problem with this issue is that user's devices do not have synchronized clocks by default. In the current communication approach, this problem means, while one peer might stall this meeting, another peer might schedule it. This situation must not happen in a scheduling application.

7.2 Interesting future project

Because SCAP uses e-mail messages to communicate, it may be interesting to investigate how the device's battery life time is affected by SCAP messages. If SCAP could be reprogrammed to poll e-mail messages by itself, the application could be optimized to adjust the poll interval depending on what messages are received and what messages are expected because of the communication protocol.

For example, the organizer peer once a meeting time has been proposed is expecting replies from all attendees. Because the attending peers reply without any kind of user interaction in between, the organizing peer expects the replies almost directly. Thus, the organizer's device could maintain the connection to the e-mail server while waiting. For how long the device should maintain the connection depends on the average processing and communication delays.

The masters thesis by Saha and Saqibuddin [34] proposes an interesting approach where an e-mail server is set up on the mobile device that should receive e-mail messages. The advantage with this approach is that the device can poll for new messages locally, thus almost without any cost in terms of the battery power consumed compared to polling a remote e-mail server. Additionally the e-mail latency is reduced greatly. SCAP should utilize this functionality.

7.3 Conclusion

A calendar-like application was programmed that is capable of manipulating a user's calendar through the interface on an Android device. The application communicates by sending e-mail messages and runs a simplified scheduling algorithm allowing it to schedule meetings between several users. A GUI was programmed in order to enable people to use (and control) the application.

When meetings are booked, the participants' current location information along with their calendars is used to determine if the meeting can be booked at the current time. This can occur when the participants are close to each other.

7.3.1 Meeting the goals of the project

Based on the implementation and the evaluation of the scheduling application, objectives 1-4, as defined in section 1.4 on page 6, have been met, some aspects have been quite successful. We will examine each of these objectives in more detail in the following paragraphs.

7.3.1.1 First objective

When a meeting is booked, meeting details are entered either manually or default parameters are used where it makes sense. A meeting object is created and added to the user's personal calendar, as configured through the application settings menu. Thus, the first objective is met successfully.

7.3.1.2 Second objective

The scheduling application utilizes the iCalendar format, which enables cooperation with other applications using the same standard. The communication between SCAP peers utilizes e-mail messages, which is a major advantage for SCAP's communication due to the incredibly large number of e-mail servers available.

Unfortunately, manipulating a remotely located calendar is limited because of the Android platform, currently only Google Calendar and Outlook Calendar are supported. Additionally, calendar synchronization is not managed by SCAP. This is a disadvantage that the application can not directly upload calendar changes to a calendar server. However, this is not a limitation of SCAP itself, but rather the synchronization functions of the Android platform. SCAP functions perfectly well even without a calendar server by using the phone's calendar - however, this can result in inconsistency between the device's view of the user's schedule and the user's calendaring server's view.

CalDAV should be used to allow SCAP to have "direct" calendar access and SCAP could be optimized to discover calendar changes by utilizing ETags. Unfortunately, not all calendar servers support CalDAV, thus SCAP's functionality depends on the calendar server and technology used. Thus, while the second objective has been met, the goals of the second objectives could have been achieved in a different way and it is not clear if there exists "better" ways to implement this functionality.

7.3.1.3 Third objective

In order to reason about whether the third objective has been or not, and if it has been met with success, several question results might be helpful. One could reason that if users find that SCAP reduced the time and effort needed to plan collaborative tasks (as stated in question one), then this also means that the application did not demand much attention from the user, while scheduling a meeting. This reasoning is also valid the other way around.

Figure 6.4 on page 86 shows how users responded to question 5 in the survey. These results suggest that most of the users did not want to receive any additional notifications, but it is hard to tell whether too many or not notifications were triggered. Remember that the current implementation of SCAP did not allow users to customize or turn off the notifications, thus users were not asked how they perceived the quality of SCAP's notifications. Nevertheless, the results from questions 1 and 5, which includes the results from questions 3 and 4, suggest that users were satisfied with the amount of interaction required between the them and the application. In fact, question 1 has achieved more than 2/3 of its total points (see the diagram in figure 6.3 on page 85).

These results are not surprising - most users have found the UI of the application being intuitive and easy-followed (question 3). When a user creates a meeting, the only notification that the organizing user gets, in case it is possible to schedule, is that the meeting has been scheduled. The invitees get two notifications: one for the invitation and one for scheduling, after that a meeting invitation has been accepted. This kind of UI and these notifications should not demand too much attention from users.

Another aspect of the third objective that should be investigated is the user's perception about how easy a meeting is created. This might depend on the UI of the application, thus on user's learnability to complete a task, and how different the application might be compared to other similar calendaring software. From the diagram in figure 6.1 we can see that 3/4 of all test users often use some kind of calendaring software in order to view or edit their calendars. From the following diagram in figure 6.2, we can see that approximately 2/3 of all users often use some kind of calendaring software in order to book meetings. From both of these diagrams, we can judge that if SCAP would be perceived differently from other calendaring applications, then most of the test users probably would notice that **and** the test results would be affected greatly.

The results of question 2 is exactly what is needed to show the application's differences. Remember that this question is answered by answering on questions 10 and 12. One can see that question 2 has achieved over 80 % of its maximum points, thus this question is one of the top rated ones.

In summary, the third objective might be one of the most investigated (by the survey), because questions 1-5, 10 and 12 are relevant. From these questions we can say that the third objective has been met with great success.

7.3.1.4 Fourth objective

SCAP utilizes location information in order to see if a meeting's participants are close to each other. If they are close, then SCAP will schedule the meeting at the current time. This feature has been favored by users very well (see the achieved score of question 10).

Unfortunately, the time of this project did not allow the implementation of any more interesting features that utilize location information. The user is referred to section 7.1 on page 88 where some of these features are listed. The goals of the fourth objective are many and it was hard at the start of this project to estimate the time needed in order to accomplish this objective. Probably, the goals were set to high. Therefore, the fourth objective has nearly been met, if at all.

7.3.1.5 Fifth objective

Because there are dozens of ways how location information could be utilized, one might say that the fifth objective is a subset of the fourth objective. However, the fifth objective takes up a slightly different but specific purpose for location information - suggest a co-located meeting place based on the meeting participant's current/prior position(s) *and* compute a meeting's cost to be rescheduled and make it location dependent.

As already mentioned in the previous section, SCAP has a good potential to propose a meeting place somewhere near the participants. However, SCAP does not provide users with the information that the other participants are close, nor does the program decide a precise meeting location. Therefore, the goals of the fifth objectives have hardly been met.

7.3.2 Security and privacy

The messages sent by the scheduling application can be intercepted and read, because they are sent in clear text. However, users should configure their e-mail application (used by SCAP) to use a secure socket layer protocol, so that application messages (containing free/busy and location information of users) would be protected between the SCAP application and the e-mail server.

Section 2.2.1.6 on page 13 stated that a problem in the iCalendar format is its inability to know about and trust other “calendar users”. This problem also applies to SCAP, because user identification is based simply on the user’s mail address. Section 2.2.3 addresses this issue and suggests security be implemented by using multipart for MIME and using public/private key authentication (i.e. S/MIME functionality).

Although SCAP relies on the security of the underlying transport protocol, a certain level of security can be implemented by the users themselves. This includes enforcing use of a secure mail transfer protocol, as already mentioned, and declining meeting invitations from “suspicious”[†] e-mail addresses. The application protocol of SCAP will not distribute private user information to every participant, but does provide this information to the organizer.

7.3.2.1 SCAP-k9mail protocol issue

Section 5.2.1 described how SCAP communicates with k9mail in order to talk to an e-mail server. Because the programs communicate through broadcast receivers, any other program could implement the interface of a broadcast receiver in order to receive messages sent between SCAP and k9mail. Thus, the intruder program could learn about the attachments available in a received e-mail message. As SCAP, a such program could instruct k9mail to download attachments. k9mail broadcasts a message to SCAP in order to inform it about a successfully downloaded attachment. Any program receiving such broadcasts could access the downloaded attachments.

7.3.2.2 Issue with the e-mail recognition

SCAP needs to know what e-mail messages are designated to be SCAP messages. SCAP learns this from a subject of an incoming e-mail message (the subject must contain the string “SCAPA”).

Sending e-mail messages with this subject enforces SCAP to download the header of this message, in order to see if this message contained a calendar attachment. The content type header information could be used in order fake a calendar attachment, in truth being any kind of program or file. k9mail will download this file and by

[†]Note that various denial of service attacks and calendar spam are possible despite a secure protocol.

broadcasting reveal where this file will be saved, thus enabling other programs to access the file.

7.3.3 General conclusions

Generally, this project evolved quite well, although not without complications. From the start it was a bit hard to define clear and specific goals and to limit the scope of this thesis project. It was hard to find some of the information that would have been very useful for a pre-study. While the Android developer's site contributed significantly to this project, it had some weaknesses; specifically it was not always obvious how to find relevant information, as developer's guides tended to be rather detailed - but not always clear. Sometimes, the information found was too theoretic and there was a need to either learning by following the guide's tutorials, or searching for and understanding of additional examples.

7.3.4 The "iCalendar standards"

Implementation-specific issues have lead to a longer time than expected being required for the development. A major workaround needed to be made in order to make it possible to cooperate with the Android device's calendar database and to keep the back-end synchronization from invalidating a meeting object, because of the object's properties and values.

The iCalendar format itself might be a crystal-clear definition of what a calendar application has to support, and there might be dozens of ways to actually implement it, disregarding iCalendar extensions. Thus, saving an iCalendar object in a database required additional knowledge of the database architecture. Besides that, individual calendar servers have certain rules about what additional component properties an iCalendar object may or may not have, besides the mandatory properties. The same applies to limitations on their values. Additional values may be defined for iCalendar properties and/or string values represented by numbers instead, because of the database architecture chosen.

The iCalendar format itself is ambiguous. For example, consider all the different ways that are possible to represent a value of a date/time period, or the diversity of representing a free/busy period inside a free/busy component. All these possibilities together sum up to dozens of different formats, although formally defined to have one meaning. From a developer's point of view, there is no need for this variety, as a single representation is sufficient and saves both time and effort when dealing with such values.

7.3.5 Conclusions related to the evaluation

Also the evaluation could have been done better. This concerns both the questionnaire that was print on paper (instead of having an online questionnaire) and the time spent while supervising test users. Some users postponed answering the questions until some days after that they have done the "test", which might have affected their answers. Also the need for an additional Android phone limited the time when users could come to test the application, because this device not always was available.

A different approach to carry out the evaluation task would be to have an online survey, without that the users would need to interact with the application. The users could answer the evaluation questions directly after that they have understood how the application works and how much interaction the application demands. Probably the evaluation questions would all have been answered without a large delay. However, such an evaluation approach would require either for users to read additional material describing the application when it schedules a meeting, or the application to be run on an emulator. The Android emulator would not be an option, since the emulator does not have any calendar library necessary for SCAP to run. Despite that users might have been able to run the test without any/as much supervision as needed in the evaluation that has been performed, their perception might have been different because a physical Android device would not be used. Due to the same reason, a larger amount of test users could have found such a survey less attractive and a different recruiting strategy might be needed.

Maybe the testing users could be arranged to run the tests with relatively little supervision, for example, by providing them with clear instructions of what they should do and when they should do it. At least, this would save the time required for supervision considering a large number of test users. The large number of test users in turn would enable for better test results from a statistical point of view. Additionally, the test scenarios could have been written for more than than two participants.

7.3.6 Following up the use cases

How much of the defined functionality that the application was supposed to do according to the use cases in section 1.3 was implemented? This is not obvious and this section will reflect upon these use cases.

7.3.6.1 Eating lunch at your favorite restaurant

The first use case is about people who want to jointly go out eating, but do not want to wait for each other unnecessarily before going out. The proposed program as a solution to the problem should enable users to click on a button to indicate that a user is ready (to go to eat).

The scheduling application developed in this thesis project has become something different than desired in this use case. However, SCAP could be used for this purpose too, even though some effort might be required. In SCAP, a meeting has always a (preliminarily) scheduled time and this time could indicate the meeting time outside a restaurant, for example. If someone is not ready to go by the time indicated by a booked meeting, he/she could postpone the meeting. However, this requires that everyone has available time so that the meeting can be (re-)scheduled. Moreover, if the person who wants to postpone the meeting is not the meeting organizer, the organizer must accept this new time proposal.

7.3.6.2 Discussion with your teacher

This use case is about two students who have decided to meet their teacher at an allocated meeting room. One student is late and this student does not call the other

student (or the teacher) in time. The proposed program as a solution should have an interface to manage traveling routes to predefined locations, as to reduce the chance for being late. This program also should track the user's current location so that deviations from the originally planned location can be registered, new traveling route(s) can be planned and estimates about user's delay can be made.

Unfortunately for SCAP, there is major work left to do in order for this application to function as described in the above paragraph. First of all, SCAP can not be used to plan a meeting at a selected meeting place. Secondly, SCAP does not implement the functionalities of a traveling route manager, nor does SCAP track the user's location in order to learn about a user's delay.

7.3.6.3 Just another day at work

The problem addressed in this use case is about a software manager that needs to get a meeting started without the participants to wait for each other unnecessarily. The proposed solution is similar to the program introduced in the solution to the first use case, with the addition to learn about the user's current location. Users' locations might be of help in order to determine if a meeting has been started or not and to propagate this information about a meeting's status to (absent) attendees.

SCAP can be used to plan meetings, but this is not sufficient to provide users with information about whether a meeting has started or not. SCAP does not implement any logic in order to learn if participants are inside or outside a meeting room (i.e. are sufficiently close to the meeting location or not). Although users can set their personal closeness values (i.e. a range in meter to learn if participants are to be considered close by the application to something), no conference calls are set up. SCAP provides most of the functionality to plan a meeting, but it has to be extended before the program can learn about a started meeting.

7.3.6.4 Meet your friend

This use case is about two friends living in two different cities far apart. These friends are not visiting each other often because of the large distance, but sometimes they really want to meet. One day, some of them might have some business in the town of the other friend and that is when they could be meeting each other. In the proposed solution, the application should discover such "good opportunities" and book a meeting at a commonly available time.

SCAP can be used to set up a meeting, but currently SCAP does not learn from users' locations. Thus very little can be done by the scheduling application in order to solve the problem of this use case.

Bibliography

- [1] Statistiska Centralbyrån,
The Use of computers and the Internet by private persons,
published: 2008-10-17
- [2] Statistiska Centralbyrån,
The Use of computers and the Internet by private persons,
published: 2007-12-17
- [3] Sun Yu,
Context-aware applications for a Pocket PC,
Masters thesis,
Department of Communication Systems, Royal Institute of Technology (KTH),
COS/CCS,
2007-28, December 2007.
http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/071220-Sun_Yu-with-cover.pdf
- [4] Hans-W. Gellersen, Albrecht Schmidt and Michael Beigl
Multi-Sensor Context-Awareness in Mobile Devices and Smart Artefacts,
published Oct 2002
<http://www.comp.lancs.ac.uk/~albrecht/pubs/>
- [5] The Calendaring and Scheduling Consortium (CalConnect),
The CalDAV protocol,
<http://caldav.calconnect.org/>
- [6] F. Dawson and D. Stenerson,
Internet Calendaring and Scheduling Core Object Specification (iCalendar),
Network Working Group, Internet Engineering Task Force,
Request for Comments 2445,
November 1998
<http://www.ietf.org/rfc/rfc2445.txt>
- [7] B. Desruisseaux,
Internet Calendaring and Scheduling Core Object Specification (iCalendar),
Network Working Group, Internet Engineering Task Force,
Request for Comments 5545,
September 2009
<http://www.ietf.org/rfc/rfc5545.txt>
- [8] S. Silverberg, S. Mansour, F. Dawson, and R. Hopson,
Transport-Independent Interoperability Protocol (iTIP): Scheduling Events,

- BusyTime, To-dos and Journal Entries*,
Network Working Group, Internet Engineering Task Force,
Request for Comments 2446,
November 1998
<http://www.ietf.org/rfc/rfc2446.txt>
- [9] F. Dawson, S. Mansour, and S. Silverberg,
iCalendar Message-Based Interoperability Protocol,
Network Working Group, Internet Engineering Task Force,
Request for Comments 2447,
November 1998
<http://www.ietf.org/rfc/rfc2447.txt>
- [10] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen,
HTTP Extensions for Distributed Authoring – WEBDAV,
Network Working Group, Internet Engineering Task Force,
Request for Comments 2518,
February 1999
<http://www.ietf.org/rfc/rfc2518.txt>
- [11] Wolfram MathWorld,
Great Circle,
November 2009
<http://mathworld.wolfram.com/GreatCircle.html>
- [12] C. Daboo, B. Desruisseaux, and L. Dusseault,
Calendar Extensions to WebDAV (CalDAV),
Network Working Group, Internet Engineering Task Force,
Request for Comments 4791,
March 2007
<http://www.ietf.org/rfc/rfc4791.txt>
- [13] Geographic Location/Privacy (geopriv),
Network Working Group, Internet Engineering Task Force,
June 2009
<http://www.ietf.org/dyn/wg/charter/geopriv-charter.html>
- [14] J. Cuellar, J. Morris, D. Mulligan, J. Peterson, J. Polk
Geopriv Requirements,
Network Working Group, Internet Engineering Task Force,
Request for Comments 3693,
February 2009
<http://www.ietf.org/rfc/rfc3693.txt>
- [15] G. Clemm, J. Reschke, E. Sedlar, J. Whitehead, U.C. Santa Cruz,
Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol,
Network Working Group, Internet Engineering Task Force,
Request for Comments 3744,
May 2004
<http://www.ietf.org/rfc/rfc3744.txt>

- [16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee,
Hypertext Transfer Protocol – HTTP/1.1 Network Working Group, Internet Engineering Task Force,
Request for Comments 2616,
June 1999
<http://www.ietf.org/rfc/rfc2616.txt>
- [17] *Chandler: The Note-to-Self Organizer*,
1.0, 30 July 2009,
<http://chandlerproject.org>
- [18] *OSAF Server Bundle Administrator Documentation*,
Open Source application Foundation,
October 2009,
<http://chandlerproject.org/Developers/ServerBundleAdministrator>
- [19] *Bedework Documentation 3.5*,
Bedework,
September 2009,
<http://www.bedework.org/downloads/3.5/BedeworkManual-3.5.pdf>
- [20] *Microsoft Office Outlook 2007 - demo*
Microsoft Corporation,
September 2009
<http://office.microsoft.com/en-us/outlook/>
- [21] *Tungle Extends Its Leadership By Introducing the Industry's First Web-based Calendar Accelerator*
Tungle Press Releases, 21st of April 2009,
http://www.tungle.com/Home/press/press_2009_08_21.htm
- [22] T. Vincenty,
Survey Review, Directorate of Overseas Surveys,
Ministry of Overseas Development,
Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations,
April 1975
- [23] National Atlas,
Latitude and Longitude,
http://www.nationalatlas.gov/articles/mapping/a_latlong.html
- [24] Christopher Drane, Malcolm Macnaughtan, and Craig Scott,
Computer Systems Engineering, University of Technology,
Positioning GSM Telephones,
IEEE Communications Magazine, April 1998
- [25] Ahmed El-Rabbany,
Introduction to GPS: the Global Positioning System,

2002 ARTECH HOUSE, INC
ISBN 1-58053-183-0

- [26] Benjamin Özmen,
MONGER: A Location Dependent Information System for use in Traffic,
Master Thesis, Royal Institute of Technology, School of Information and
Communication Technology,
July 2004
- [27] Mozilla,
Location-Aware Browsing,
October 2009,
<http://en-us.www.mozilla.com/en-US/firefox/geolocation/>
- [28] Jenny Charvandeh,
Location aware web access,
Masters thesis, Royal Institute of Technology, School of Information and
Communication Technology, TRITA-ICT-EX-2009:121,
September 2009,
[http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/
090907-Jenny_Charvandeh-with-cover.pdf](http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/090907-Jenny_Charvandeh-with-cover.pdf)
- [29] Daniel Hassellöf,
Position Determination using multiple wireless interfaces,
Masters thesis, Royal Institute of Technology, School of Information and
Communication Technology, COS/CCS 2008-08,
April 2008,
[http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/
080428-Daniel_Hasselöf-with-cover.pdf](http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080428-Daniel_Hasselöf-with-cover.pdf)
- [30] Android Developers *The Android Development Guide*,
January 2010,
<http://developer.android.com/guide/index.html>
- [31] John Heidemann, Dhaval Shah
Location-Aware Scheduling With Minimal Infrastructure,
Research Paper, USC/Information Sciences Institute, USENIX. 2000,
<http://www.isi.edu/~johnh/PAPERS/Heidemann00b.pdf>
- [32] Thomas Haynes, Sandip Sen, Neeraj Arora, Rajani Nadella
Automated meeting scheduling system that utilizes user preferences,
Research Paper, Association for Computing Machinery, University of Tsula,
Department of Mathematical and Computer Sciences, 2000,
<http://sigart.acm.org/proceedings/agents97/A166/A166.PDF>
- [33] Tom Tullis & Bill Albert
*Measuring the user experience: Collecting, Analyzing, and Presenting Usability
Metrics*,
Morgan Kaufmann, 2008, ISBN 978-0-12-373558-4

- [34] Mohammad Saqibuddin and Iplu Saha,
Using Simple Mail Transfer Protocol on the Last Hop,
Masters thesis, Royal Institute of Technology, School of Information and
Communication Technology, COS/CCS 2007-11,
March 2007

A. The Questionnaire

1. The main idea of the application was to reduce the time effort needed to plan collaborative tasks. How did the application manage that?

time effort reduced greatly time effort not reduced at all

2. Is the application different from calendar and scheduling software you are used to/have seen?

much similar much different

3. Is the UI of the application rather intuitive and easy-followed?

fully agree completely disagree

4. Did the application demand much attention from the user in the scheduling process?

fully agree completely disagree

5. Should the user receive additional notifications?

fully agree completely disagree

6. One way of affecting the scheduling is by giving priorities to each of a globally known number of periods (for example 4), respectively, where each period is of the same length, the first period starts at the current time (i.e. the time when the meeting is being created), and the last period ends at the deadline. This would enable for users to select approximately when they would like a meeting to be scheduled, such as “now (in the first period)”, “later (in the second period), much later (in the third period)”, or “as late as possible (close to the deadline)”. How useful would this feature be?

very useful much worthless

7. Currently, a meeting will be stalled when it is failed to be scheduled. If the organizer needs to conduct a meeting with all/a subset of the participants, he/she has no choice but try to create a new meeting object with different parameters and hope that the scheduling will succeed. How would you improve the application so that a second scheduling attempt succeeds with a higher probability?

8. SCAP users can solely book meetings with parameters that are global scheduling constraints, i.e. the parameters entered by the organizer will apply for all the meeting participants. Thus, your preferred meeting duration and preparation time variables will only be applied if you are the organizer. Are there any **personal** scheduling constraints that you would like to be applied even if you are not the organizer? Examples of personal scheduling constraints are “schedule a meeting at least 10 minutes before and after any existing meeting/calendar event”, “If the organizer is Anders Andersson, schedule a meeting as close to the deadline as possible”, “Do not schedule a meeting later than 6 p.m.”, etc.

9. Are there any additional **global** scheduling constraints you would like to have in a scheduling application? Examples of global scheduling constraints are “preferably, this meeting should last for 2 hours, but it is ok to schedule a time period of at least one hour”, “At least these participants must attend [list_of_participants]”, etc.

10. SCAP is able to “suggest” a meeting place if all the meeting participants are close enough to each other. If they are close, the meeting is scheduled “right here” and “right now”. How useful is this feature?

very useful ○ ○ ○ ○ ○ much worthless

11. SCAP could be extended, so that the application could propose a meeting time and a meeting place that is close to an event’s location of some of the

participant(s) (if this meeting is not scheduled “right now”). The participant locations can either be derived from the calendar events of the participants that are close to the proposed meeting time, or be predicted based on the history of passed events, if no events are found to be “close” to the proposed meeting time. How useful would this feature be?

very useful much worthless

12. In the iCalendar standard, which most calendaring application are based on, an event can either be tentative, confirmed, or cancelled. Note that the meaning of these states is application specific, if they are considered at all. SCAP utilized the states (although with different names, in the same order: pending, scheduled, or cancelled) and two new states are added to capture meeting agreement (setup) and that the program has stalled while scheduling this meeting. What do you think about these states?

a) very useful much worthless

b) there are too many states there are too few states

c) the states are too general the states are too specific

Questions about you

Please provide the following information about you.

1. I often use some kind of calendaring software in order to view/edit my calendar.
true/false
2. I often use some kind of calendaring software in order to book meetings.
true/false

B. The results of the survey

The results in table B.1 have the following columns, from left to right: the question number; 5 columns each containing a number of voices, where the left most column represents the count of the left most alternative of the scale; the control sum to count the number answers to each question (= number of test users); the achieved and maximum score numbers, the percentage of the achieved score and depends on (to specify what questions contributed to this question's answer). The score value in this table is calculated solely based on the question's answers (the second question was not awarded any points), while the percentage of the achieved score represents the concatenated score for that question. For example, questions 1 and 2 are concatenated. Questions 5, 6 and 11 are extension questions, therefore they do not have any score value. See section 6.3.1 for a detailed description for how score values are calculated.

	left most				right most	control sum	score	maximum score	percentage	depends on
Q1	2	7	6	0	0	15	56	75	72	1, 3, 4
Q2	0	0	3	6	6	15	0	75	84	10, 12
Q3	1	6	4	3	1	15	48	75	64	-
Q4	0	2	3	6	4	15	57	75	76	-
Q5	2	0	3	4	6	15	0	0	0	-
Q6	6	7	1	1	0	15	0	0	0	-
Q10	6	3	5	1	0	15	59	75	79	-
Q11	4	6	5	0	0	15	0	0	0	-
Q12a	5	5	4	0	1	15	58	75	77	-
Q12b	1	1	13	0	0	15	42	45	93	-
Q12c	0	0	13	2	0	15	43	45	96	-
Q12							62	70	89	12a, 12b, 12c

Figure B.1: This table shows the answers to all evaluation questions having a Likert scale. The description of this table is given in appendix B.

Q6: How useful is it to give priorities to time slots where a meeting should be scheduled?

Please see Q6 in the appendix for a more detailed question description.

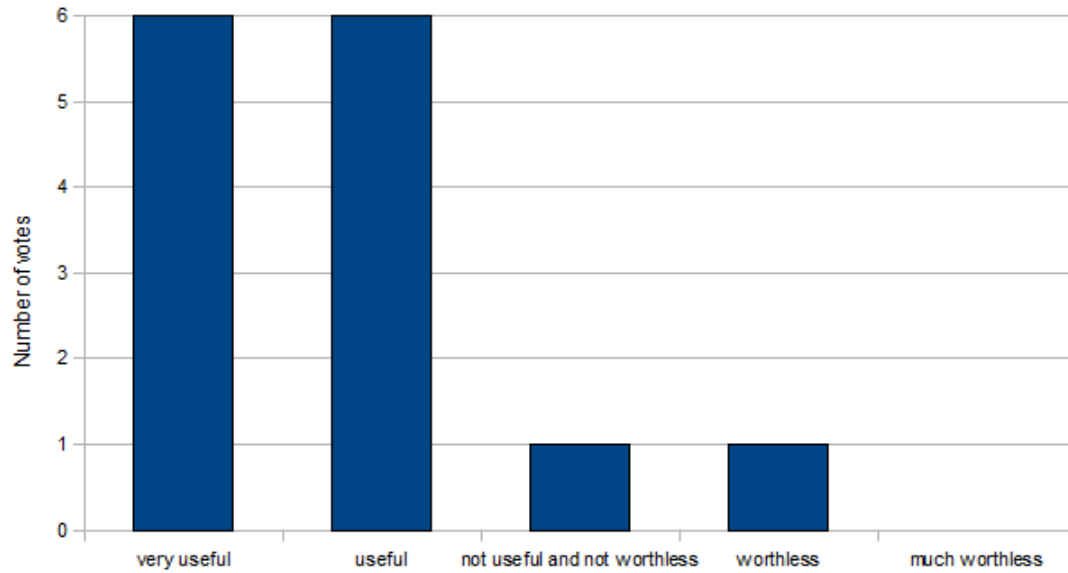


Figure B.2: This figure presents the results to question 6.

Q11: How useful is it for a scheduling application to propose a meeting place?

Please see Q11 in the appendix for a more detailed question description.

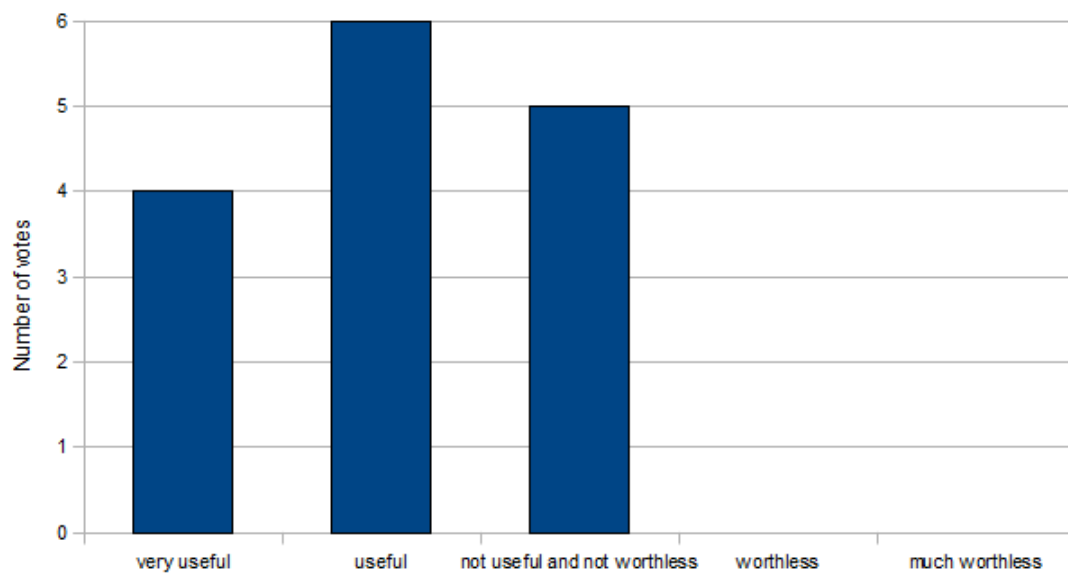


Figure B.3: This figure presents the results to question 11.

C. Introduction to the Scheduling Application (SCAP)

In this section, the reader can get acquainted with the application's fundamentals from the user's point of view. Some screenshots of the scheduling application have been attached in this section as resources to guide the introduction.

C.1 Meeting parameters

A meeting object is always tightly coupled to its parameters, since not all of these parameters can be changed in an existing meeting. The meeting duration, summary, description, deadline, preparation time and the list of attendees are parameters that can be entered. Figure C.1 shows the time tab when creating a new meeting.

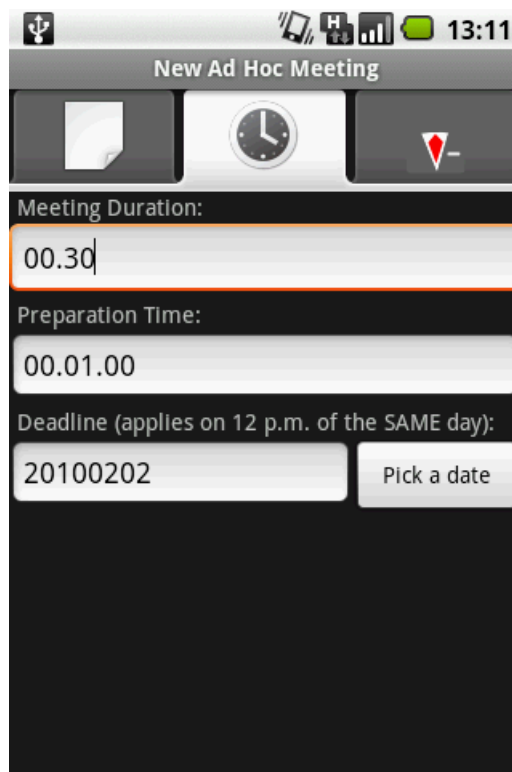


Figure C.1: The time tab of a meeting editor in SCAP.

The meeting deadline will be setting the upper bound on the calendar end date that will be used in free busy requests; the meeting duration determines the exact

length of a time period where the meeting should be scheduled; the preparation time (together with the time stamp recorded when the meeting object is added in the calendar) basically sets the earliest start time of the meeting. All time parameters in the time tab are mandatory, but they will be filled in automatically when a new meeting object is created. The default meeting duration and preparation time values are configurable into the settings. The deadline is set to the current date.

C.1.1 Attendee parameters

Attendees can either be added by selecting a contact (that contains a valid e-mail address) or entering the details of a new attendee. Only the e-mail address is required to be entered. Figure C.2 shows the attendee view. When viewing the details of an attendee in an existing meeting object, the attendee view obtains an additional button in order to view this attendee's latest comment, if specified by him/her. It is a useful and a simple feature to communicate to the organizer when declining a meeting invitation, or requesting for rescheduling.

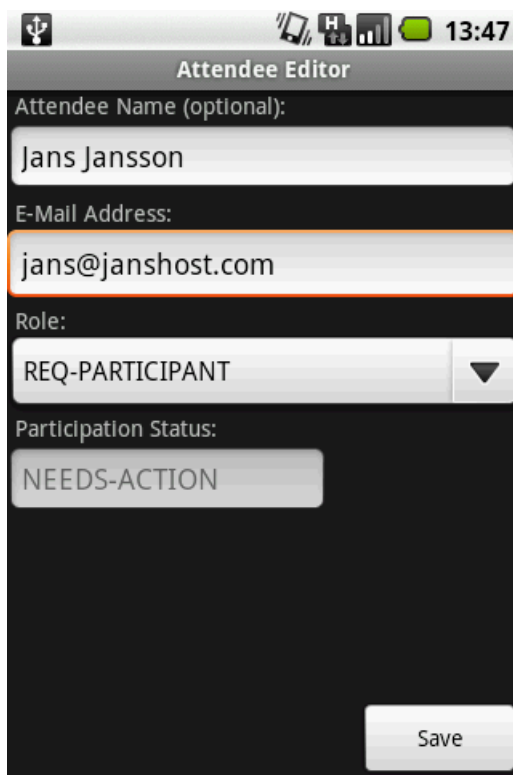


Figure C.2: The attendee editor, when adding a new attendee. The value of the role of an attendee does not have any effect on the scheduling of the meeting with that attendee. The value of the participation status can not be set, as it is controlled by SCAP and will either be set to **ACCEPTED** or **DECLINED** when this attendee has replied.

The list of all attendees for a given meeting object looks like shown in figure C.3. The view for showing an attendee's participation status will only be showing an attendee's participation status, if this attendee did not request to postpone the

meeting. If an attendee has sent a rescheduling request to the organizer, both this attendee and the organizer will be able to see the time stamp in the mentioned view on their devices. The time stamp indicates the earliest possible start time for a meeting when rescheduling.

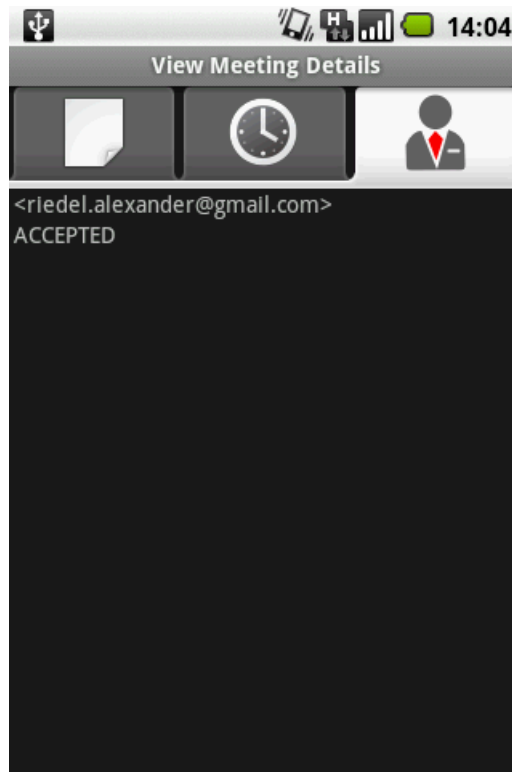


Figure C.3: The list of all attendees in the current meeting showing their display name (= e-mail address and their name, if exists) in the upper list item view and their participation status in the lower view. The participation status has the initial value `NEEDS_ACTION` and will be set to either `ACCEPTED` or `DECLINED` when this attendee has replied.

C.2 Meeting list

All meetings that have been added in the calendar using the scheduling application can be seen in the meeting list. The meeting list is also the “main” window of SCAP since many important actions can be issued from here. Figure C.4 shows the options menu of the main window. From here, the users can navigate to SCAP settings, create a new meeting object, etc. The “Remove ALL” menu item actually was added as a “debug” button, as SCAP in the process of developing crashed many times when a user clicked on a meeting to view its details. Nevertheless, pressing on it cleared the current calendar from all ad hoc meetings in two clicks (the second click is needed in order to confirm the deletion).

SCAP does not offer a calendar-like view where events are rendered as rectangles while occupying an area proportional to their duration. On the other hand, SCAP

user can change the view of a meeting list. Pressing on the “change view” button will display a menu where the user can set a status filter that all meeting must match in order to be in the meeting list. The filter can either be “none” or one of the meeting scheduling states (the scheduling states are discussed in section C.3). When all meetings are shown, the user not only can distinguish them by names (although they are not sortable by names in the current implementation), the user also can recognize a meeting scheduling state by the colored vertical bar displayed to the left of each meeting. Figure C.5 shows an example of a meeting list with a filter set to “none”.

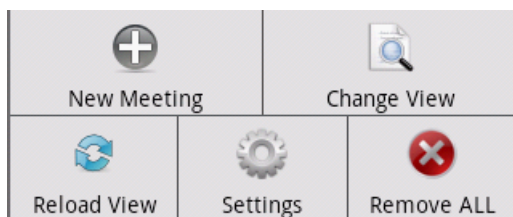


Figure C.4: The different option menu choices, when pressing the menu button on the android device, while viewing the meeting list.

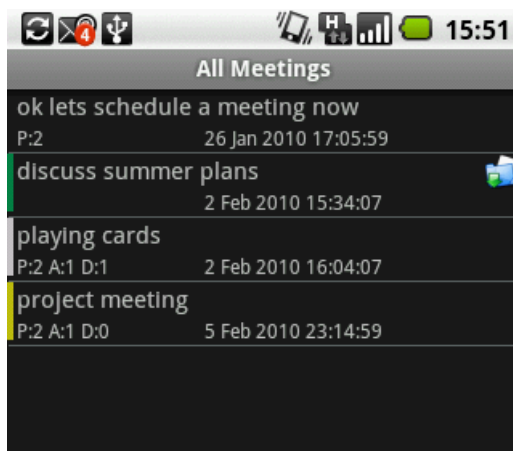


Figure C.5: The meeting list view. Users can see the most important meeting details, like meeting summary and scheduled time, in each list item. The meeting state can be identified by the color of the vertical bar immediately to the left of each meeting, respectively. See section C.3 for a detailed description of the scheduling states. The blue-green folder-like icon to the right indicates that the user of this device is not the organizer of this meeting. Organizers can also see a summary of the current participation statuses of all attendees: P stands for the number of participants, A for accepted and B for declined. Each list item also displays the meeting start time.

C.3 Meeting scheduling states

A meeting can be in different states, see figure C.6. Although all states are “visible” for all participants, not all transitions are seen for attendees. Moreover, the attendees

do not necessarily have the same view of a meeting all the time.

For an attendee, the view is rather simple: A new meeting invitation is shown in state setup and the state goes over to pending as soon as this attendee has accepted. The meeting will remain in state pending until this attendee has received a confirmation, or a cancel message. A confirmation message means to go over to scheduled, while a cancel means that the meeting becomes stalled. If an attendee declines an invitation, this basically means that he/she kicks himself/herself out, thus the meeting will be stalled for that attendee (and for the organizer getting this decline message).

The organizer is the only person who cares about accepts/declines and the only person who knows about everyone's free busy information. That is, transitions S2, S3 and S4 are only seen by the organizing peer, so that this peer can react upon such events. For example, when there is no common free time available, the organizer will see a meeting in state stalled. Meanwhile, all of the attendees see the same meeting be in state pending, because all of them have accepted the previously sent invitation. While in state stalled, the organizer has the choice of cancelling the meeting, or creating a new meeting object with different parameters.

If an attendee declines an invitation, the organizer and the declining attendee will see the meeting be in state stalled, no matter what the other attendees, if any, replied, or going to reply. In this situation, the organizer can cancel the meeting, or reschedule it to a later time. This also explains why the transitions stalled to stalled and stalled to pending are possible for the organizer only if the meeting has not been cancelled yet.

C.4 Rescheduling a meeting

A meeting can be rescheduled to a new time. Only the organizer can issue a rescheduling attempt (a new time negotiation), but attendees may propose a new time, once they have learned that the meeting in question has been scheduled. This postpone view is identical both for the organizer and for an attendee. The functional difference is that a new time negotiation is issued directly, when the organizer submits the form. When an attendee submits the same form, a rescheduling request is sent to the organizer. When the organizer accepts one proposal or issues a new negotiation by himself/herself, all attendee proposals made so far and the attendees' participation status are cleared in the meeting, and the meeting becomes pending.

C.5 Notifications

In SCAP, users can not be reminded of starting meetings because time alarms are not managed by SCAP - this is up to the calendar system that is used. However, SCAP can notify a user when the application has received an invitation message, when a meeting has been (re)scheduled, or when this user is an organizer receiving a request for rescheduling. All notifications will display a specific icon on the action bar of the Android device and the users can navigate to the SCAP meeting list view (alternatively, to the meeting details view when clicked on a request for rescheduling icon) by opening the notifications menu and then clicking on the notification icon. Clicking on the notification icon will clear all notifications of the same type. Note that notifications are also cleared when the meeting list is reloaded (and the current filter “matches” the type of notification). The meeting list can be reloaded either by the user refreshing it, when the screen orientation changes, or when the meeting list view comes to the foreground (for example, when an overlaying window has been closed).

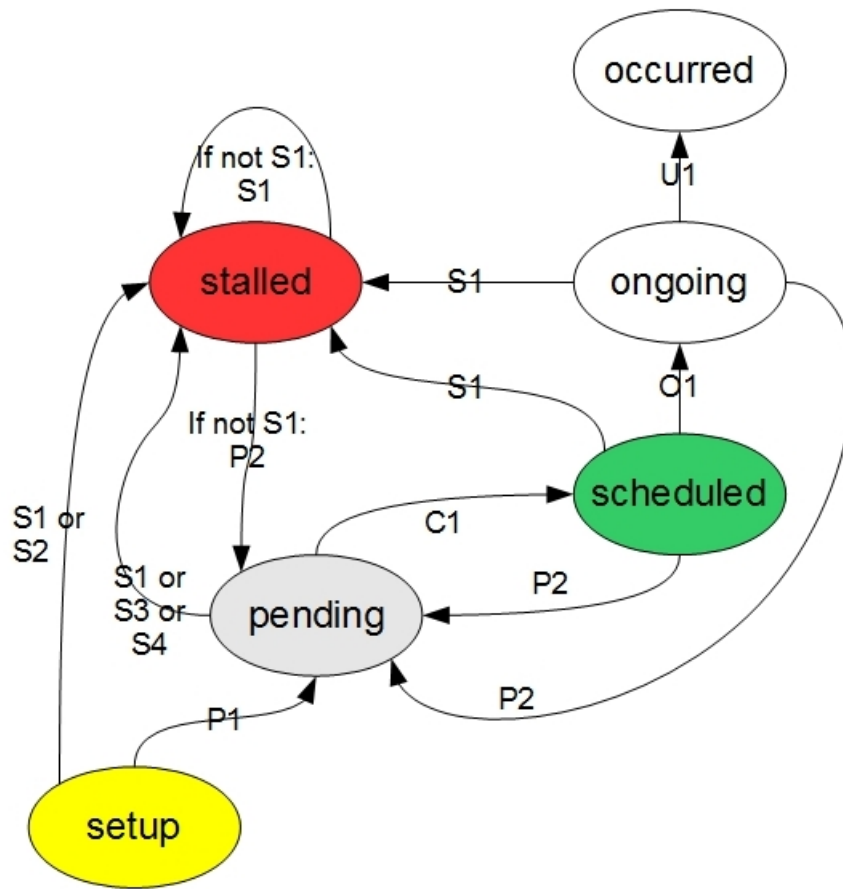


Figure C.6: The meeting scheduling states and their possible transitions. If a transition’s condition contains the words organizer and attendee in parenthesis, then this transition condition is different for the organizer and attendee. Transitions starting with “if not *conditional transition*” can only take place if the assertion taken from that *conditional transition* is false.

To stalled:

- S1 Organizer cancels the meeting
- S2 (Organizer) Someone declines the meeting invitation/(Attendee) I decline the meeting invitation
- S3 No free common time (of sufficient length) available
- S4 Current time greater than time stamp of the deadline

To pending:

- P1 (Organizer) Everyone accepts the meeting invitation/(Attendee I accept the meeting invitation)
- P2 The organizer issues a new rescheduling attempt

To scheduled:

- C1 There is at least one common time slot available

To ongoing:

- O1 Current time is greater than the meeting’s start time and less than the meeting’s end time

To occurred:

- U1 Current time is greater than the meeting’s end time

