

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Olli Korjus

Meeting Scheduling Assistant:

Automatic scheduling between heterogeneous calendar systems

Master's Thesis
Espoo, September 27, 2012

Supervisor: Professor Tomi Männistö
Instructor: Varvana Myllärniemi M.Sc. (Tech.)

Author:	Olli Korjus	
Title:	Meeting Scheduling Assistant: Automatic scheduling between heterogeneous calendar systems	
Date:	September 27, 2012	Pages: 115
Professorship:	Software Engineering	Code: T-76
Supervisor:	Professor Tomi Männistö	
Instructor:	Varvana Myllärniemi M.Sc. (Tech.)	
<p>Finding a suitable time for a meeting between multiple people is a common practical problem today. The meeting scheduling process involves sharing information about one's availability and negotiating possible meeting times. The process can often be arduous and time-consuming. There are many software applications that simplify the scheduling process but they often still require a lot of manual effort to use or cannot be used to schedule meetings across different organizations.</p> <p>In this thesis, we design a software architecture for a meeting scheduling system that allows people to schedule meetings with each other effortlessly and regardless of organizational boundaries. The meeting scheduling system finds suitable meeting times by automatically reading the calendar information of the users to determine their availability. The calendar information is read directly from whatever device or system that the user uses to maintain her calendar, such as a personal mobile phone or a company's calendar server. Support for each type of calendar system must be developed separately but the generic interfaces provided by the meeting scheduling system allow third parties to develop support for any calendar system.</p> <p>We also implement a prototype meeting scheduling system based on the architecture. The prototype successfully realizes all of the key features of the architecture. The prototype consists of a central web server and a client component that runs on a Nokia N950 mobile phone and reads calendar information from the built-in calendar of the phone.</p> <p>The architecture design and the prototype are evaluated in three ways. First, we verify that the architecture and the prototype adhere to their original specification. Second, we compare the effortlessness of the prototype with that of popular existing scheduling applications. Third, a group of experts validates that the architecture and the prototype fulfill the needs of the users.</p>		
Keywords:	meeting scheduling, heterogeneity, calendar systems, integration, automation	
Language:	English	

Tekijä:	Olli Korjus		
Työn nimi:	Aputyökalu kokousaikojen sopimiseen: Automaattinen ajan etsintä heterogeenisten kalenterijärjestelmien välillä		
Päiväys:	27. syyskuuta 2012	Sivumäärä:	115
Professuuri:	Ohjelmistotuotanto	Koodi:	T-76
Valvoja:	Professori Tomi Männistö		
Ohjaaja:	Diplomi-insinööri Varvana Myllärniemi		
<p>Sopivan ajan löytäminen usean ihmisen väliselle kokoukselle on nykyään yleinen käytännön ongelma. Ajanetsimisprosessissa osallistujat jakavat tietoa omista aikatauluistaan ja neuvottelevat mahdollisista kokousajoista. Tämä prosessi voi usein olla työläs ja aikaavievä. Monia ajanetsintäprosessia helpottavia ohjelmistosovelluksia on olemassa, mutta niiden käyttö vaatii usein paljon vaivannäköä tai niitä ei voi käyttää kokousaikojen sopimiseen eri organisaatioiden välillä.</p> <p>Tässä työssä suunnittelemme ohjelmistoarkkitehtuurin kokousaikojenetsimisjärjestelmälle, joka mahdollistaa kokousaikojen sopimisen vaivattomasti ja piittaamatta eri organisaatioiden välisistä rajoista. Kokousaikojenetsimisjärjestelmä lukee automaattisesti käyttäjien kalenteritietoja selvittääkseen heidän aikataulunsa ja löytääkseen näin kokoukselle sopivat ajat. Kalenteritiedot luetaan suoraan mistä tahansa laitteesta tai järjestelmästä jossa käyttäjä ylläpitää kalenteriaan, kuten henkilökohtaisesta matkapuhelimesta tai yrityksen kalenteripalvelimelta. Tuki jokaiselle eri tyyppiselle kalenterijärjestelmälle täytyy kehittää erikseen, mutta kolmannet osapuolet voivat kehittää tuen mille tahansa kalenterijärjestelmälle kokousaikojenetsimisjärjestelmän tarjoamien yleisten rajapintojen ansiosta.</p> <p>Kehitämme tässä työssä myös prototyypin kokousaikojenetsimisjärjestelmästä arkkitehtuurisuunnitelman pohjalta. Prototyyppi toteuttaa onnistuneesti kaikki arkkitehtuurin avainominaisuudet. Prototyyppi koostuu keskusverkkopalvelimesta ja asiakaskomponentista, joka pyörii Nokian N950-matkapuhelimessa ja lukee kalenteritietoja puhelimen sisäänrakennetusta kalenterista.</p> <p>Arkkitehtuuria ja prototyyppiä arvioidaan kolmella tavalla. Ensimmäiseksi varmennamme, että arkkitehtuuri ja prototyyppi noudattavat alkuperäistä spesifikaatiota. Toiseksi vertaamme prototyypin ja suosittujen olemassaolevien ajanetsintäohjelmien käytön vaivattomuutta. Kolmanneksi ryhmä asiantuntijoita vahvistaa, että arkkitehtuuri ja prototyyppi täyttävät käyttäjien tarpeet.</p>			
Asiasanat:	kokousaikojen etsintä, heterogeenisyys, kalenterijärjestelmät, integraatio, automaatio		
Kieli:	Englanti		

Acknowledgements

Most of all, I wish to thank my instructor Varvana Myllärniemi for all the guidance and feedback that she has consistently given me during the writing of my thesis. Her insight and enthusiasm have been an inspiration for me and the thesis could never have reached the level of quality that it has without her aid. Also, I wish to thank my supervisor Tomi Männistö for his expertise and for helping me define the topic of my thesis in the beginning.

In addition, I wish to give my heartfelt thanks to Mikko Raatikainen for providing frequent and knowledgeable feedback for my thesis. He also contributed to and was closely involved in the architecture design process.

I also wish to thank Terho Norja and others at Intelligent Precision Solutions and Services Ltd. for their assistance and dedication to the architecture design process. They provided unique perspective and invaluable comments that helped shape the architecture into a feasible and valuable solution.

I am also particularly thankful to Jari Pääkkö and Mikko Ylikangas for their help with developing the prototype implementation but especially for their joyous company during my work days. I am also grateful to everyone at SoberIT for making it the best place I can think of to write my thesis.

Finally, I of course want to give special thanks to my family for all their love and support and especially to my father and brother for setting such a fine example for me to follow.

Espoo, September 27, 2012

Olli Korjus

Abbreviations and Acronyms

API	Application Programming Interface
CRM	Customer Relationship Management
DS	Design Science
EARS	Easy Approach to Requirements Syntax
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
IIMS	Intelligent Inter-organizational Meeting Scheduler
IPSS	Intelligent Precision Solutions and Services Ltd.
MSS	Meeting Scheduling System
MVC	Model-View-Controller
ORM	Object-Relational Mapping
PDA	Personal Digital Assistant
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
UI	User Interface

Contents

Abbreviations and Acronyms	5
List of Tables	8
List of Figures	9
1 Introduction	10
1.1 Motivation	10
1.2 Research goals and questions	11
1.3 Scope	15
1.4 Research methodology	16
1.5 Structure of the thesis	20
2 Previous Work	21
2.1 Existing scheduling applications	21
2.2 Previous research	25
3 Requirements	29
3.1 Important features	29
3.2 Terminology	30
3.3 Assumptions	30
3.4 Detailed requirements	31
4 Architectural Views	37
4.1 Context diagram	37
4.2 Functional view	40
4.3 Information view	47
4.4 Process sequence view	52
4.5 Integration view	54

5	Prototype Tool Implementation	59
5.1	Prototype functionality	59
5.2	Technology choices	64
5.3	Implementation of the functional elements	66
5.3.1	Central system	66
5.3.2	Calendar system	72
5.4	Design and implementation of the interfaces	73
5.4.1	E-mail interface	73
5.4.2	REST interface	74
5.4.3	Long polling interface	76
5.4.4	Interface usage sequence	79
6	Evaluation	80
6.1	Traceability from research questions to implementation	80
6.1.1	From research questions to requirements	81
6.1.2	From requirements to architecture	81
6.1.3	From architecture to implementation	82
6.2	Comparison to other existing scheduling applications	84
6.3	Expert evaluation	87
6.3.1	Evaluation procedure	87
6.3.2	Evaluation of effortlessness	88
6.3.3	Evaluation of the scheduling results	89
6.3.4	Evaluation of integration capabilities	90
6.3.5	Evaluation of privacy protection	91
6.3.6	Other evaluation findings	92
6.3.7	Expert evaluation summary	93
7	Discussion	94
7.1	Answers to the research questions	94
7.2	Threats to the validity of the results	97
8	Conclusions	99
8.1	Conclusions	99
8.2	Future work	101
A	REST Interface Methods and Fields	109

List of Tables

1.1	Summary of the MSS success factors.	14
2.1	Comparison of existing meeting scheduling applications.	24
3.1	Terminology used in this thesis.	30
3.2	Assumptions made regarding the problem area.	31
3.3	Integration and privacy requirements.	32
3.4	Group and available times customization requirements.	32
3.5	Meeting creation requirements.	33
3.6	Meeting time finding requirements.	34
3.7	Meeting notification and response requirements.	35
4.1	Decisions regarding the context of the MSS.	38
4.2	Decisions regarding the functional elements of the MSS.	40
4.3	Decisions regarding the information in the MSS.	48
4.4	Information elements in the MSS.	51
4.5	Decision regarding the adding of the meeting event to calendars.	52
4.6	Decisions regarding the integration capabilities of the MSS.	55
5.1	Scheduler database table summary.	70
5.2	HTTP methods accepted by the REST interface.	74
5.3	Long polling interface response types.	77
6.1	Omitted requirements in the prototype.	83
6.2	Comparison of MSS architecture and MSS prototype.	84
6.3	Comparison of existing scheduling applications and MSS.	86
A.1	REST interface resource URLs.	109
A.2	REST interface methods for each resource.	110
A.3	Data fields of the Meeting REST resource.	111
A.4	Data fields of the Time Period REST resource.	113
A.5	Data fields of the Request Response REST resource.	114
A.6	Data fields of the User REST resource.	115
A.7	Data fields of the Group REST resource.	115

List of Figures

2.1	Screenshot of Doodle web service.	23
4.1	Context diagram of the MSS.	39
4.2	Functional view of the MSS.	42
4.3	Information view of the MSS.	50
4.4	Scheduling process sequence diagram.	53
4.5	Integration of different types of calendar systems to the MSS.	56
4.6	Interface usage sequence diagram.	58
5.1	Screenshot of the prototype client script output.	60
5.2	Screenshot of the prototype meeting creation form.	61
5.3	Screenshot of the prototype meeting detail view.	62
5.4	Screenshot of the prototype administrator website.	63
5.5	Solution stack of the MSS prototype.	65
5.6	Prototype functional view.	67
5.7	Database diagram of the <i>Scheduler database</i>	69
5.8	Prototype interface sequence diagram.	79

Chapter 1

Introduction

In this chapter, we introduce the problem area, define the focus of the research by setting research questions and the limits to our scope. We also describe the research methods that we have chosen for this thesis.

1.1 Motivation

Finding a suitable time for a meeting between multiple people is a common practical problem in many work places. The meeting scheduling process usually involves sending and receiving meeting time proposals back and forth between the participants until a suitable time is found. This is an arduous and time consuming task and can often lead to situations where some participants might agree to times that are not the most suitable for them just because they want to get the scheduling process over with.

The main challenge behind meeting scheduling is the fact that one participant is only familiar with her own calendar events and thus only knows which times are suitable for her and not the others. So, to be able to compare everyone's free times, and to find the time that is suitable for all, the free times need to be elicited individually from each participant, which can cause a significant amount of work depending on the method of communication (e.g. talking, phone, email etc.)

Many software applications exist which can help people share calendar information more easily. Some of them, for example Microsoft Exchange (Microsoft Corporation, 2011), enable the automatic sharing of electronic calendars between groups of people. This way, one person can quickly see which times are available for others and schedule a meeting appropriately. Other applications, such as Doodle (Doodle AG, 2012), provide a convenient way for people to inquire and share the available times that they have

checked from their calendars themselves. However, all of these applications often have some limitations to their usefulness. For example, the sharing of calendars can usually only happen between users of the same application and within a single organization. Thus, people wanting to schedule a meeting between other people in different organizations still need to resort to e-mail or telephone discussions. The limitation with the applications that share manually checked times is that checking your own available times by hand is cumbersome and it can take a long time for all participants to share their times.

Despite the large offering of scheduling tools available, there does not seem to be an optimal solution that would allow easy meeting scheduling across organizations. In this thesis, we study the problems that are involved in designing such a solution and describe a software architecture for a Meeting Scheduling System (MSS) that is effortless to use and support scheduling across organizational boundaries.

1.2 Research goals and questions

We intend the MSS to be useful to anyone and everyone who might be struggling to find suitable times for meetings. Because of this, we aim to design the MSS to be easy to take into use by a broad audience. There are many things that influence how fast a new innovation, such as the MSS, is adopted by the prospective users. Such things are, for example, the communication channels through which information about the innovation spreads or the social system within which the adoption occurs but also, of course, the characteristics of the innovation itself (Rogers, 1995). The characteristics of the innovation are the most interesting of these factors for us since in this thesis we are designing the innovation (the MSS) and its characteristics.

The characteristics of an innovation that affect its adoption are its relative advantage, compatibility, complexity, trialability and observability (Rogers, 1995). *Relative advantage* (Rogers, 1995) means that users perceive the innovation to be somehow better than previous solutions. *Compatibility* (Rogers, 1995) refers to how well the innovation matches with existing values, needs and the past experiences of the users. The adoption happens slowly if users have to change their values or learn new behavioral patterns to be able to utilize the innovation. *Complexity* (Rogers, 1995) measures how difficult it is for the users to understand and use the innovation. *Trialability* (Rogers, 1995) is the possibility of the users to test the innovation in some limited format. Adoption is likely to be faster if the users are able to evaluate the benefits of the innovation without having to commit to it too strongly. *Ob-*

servability (Rogers, 1995) means how well the results of the innovation are visible to others. If an innovation has high observability, the news of its usefulness reach prospective new users quickly.

Based on our own experience on using existing meeting scheduling applications and other software, we have identified four success factors for the MSS which address these important characteristics defined by Rogers (1995): The MSS should offer something that existing scheduling systems do not, provide satisfactory results, be easy to take into use and protect the privacy of the users.

The most important characteristic of an innovation such as a software application is to have a relative advantage to other similar applications. That is, the application should **be better than competing applications** in some way. Many applications already exist that attempt to solve the problems related to meeting scheduling but most of them have some limitations that make them sub-optimal for certain usage scenarios. Some require a lot of manual effort to use while some require that all participants use the same system, thus usually confining the scheduling capabilities to within a single organization. So, to improve upon these existing systems, the MSS should enable the easy scheduling of meetings over heterogeneous calendar systems and organizational boundaries.

Also related to providing a relative advantage, the MSS should **deliver results that the users are satisfied with** so that it is at least not any worse than the other systems in this regard. This is a fairly obvious criterion but we feel it is an important success factor for the MSS. The reason is that while meeting scheduling is a process that is a suitable target for automation due to its repetitive nature, the quality of the results from a fully automated scheduling process is more difficult to guarantee. This is because the calendar from which an automated process would determine a user's availability is not a perfect source of information. People do not always record all of their schedule in a calendar (Ehrlich, 1986). For example, someone might not mark their regular lunch break in their calendar but would still prefer not to have meetings at that time. Also, the other participants of the meeting could influence which times a person wants to make available for that meeting. For example, some people might want to meet with customers only on mornings but would be willing to meet coworkers at any time. In cases such as these, it is only the person herself who can truly know what times are suitable for her and a completely automated process would fail to provide good results. A way in which this problem could be alleviated is to allow the users to somehow provide input to the automated scheduling process in order to get better results. On the other hand, of course, requiring the users to contribute too much effort into the process would defeat the point of the automation in

the first place so a balance between automation and manual effort must be found.

To avoid too much complexity, the MSS should **be easy to learn and simple to deploy**. The amount of new things that the users have to learn is minimal if the MSS takes as much advantage as possible from other software applications or systems that the users have already learned to use. Specifically in our case, the users should not have to learn to use an entirely new calendar or scheduling system but, instead, the MSS should utilize the calendar systems that the users are already familiar with, such as the calendar on their mobile phone or other device. Allowing the users to use the same calendar systems that they have used before is also a great way to ensure that the MSS is compatible with their previous values and experiences, which should make them more readily adopt it. Utilizing existing calendar systems also has a positive impact on the trialability of the MSS because the users are able to quickly test the MSS without too many new things to learn and slow them down.

In this thesis, being simple to deploy refers to how well the application is supported on the software and hardware environments that different people might be using it on. Or, if such support for a specific environment doesn't already exist, how easily it can be developed. To facilitate this, the software application should be designed to be open and easily integrable to external systems and as free from various technological constraints as possible.

We believe that one important success factor for any system is to **protect the privacy of the users**. This is related to providing a relative advantage to other solutions because the privacy protection should be at least on the same level if not better than on competing systems. Privacy is also one aspect of compatibility with existing values and past experiences because people want their calendar data to remain private and have come to expect that from the other calendar related applications that they use. Privacy preservation is an interesting problem for us because the entire purpose of the MSS is to give users more information on the availability of others but at the same time the users should not be able to see too much private calendar information of the others. Finding the right balance between privacy and exposing enough information for the MSS to function is not trivial.

The success factors and the innovation characteristics they address are summarized in Table 1.1.

The one characteristic that our success factors do not address is observability. The results of the MSS are not very visible to those that are not already using it because the meeting times scheduled for others just are not relevant to them. However, we believe this problem is offset by the fact that users who have gotten good results with the MSS are likely to recommend

Table 1.1: Summary of the MSS success factors.

Success factor	How to realize	Related innovation characteristic
Be better than competing applications.	Enable easy scheduling across heterogeneous calendar systems and organizational boundaries.	Relative advantage.
Provide satisfying results.	Automate certain parts of the scheduling process but allow users to provide input to the process.	Relative advantage.
Be easy to learn and simple to deploy.	Utilize existing calendar systems and make it easy to develop support for new calendar systems.	Complexity, compatibility, trialability.
Protect the privacy of the users.	Prevent private calendar information of the users from being visible to other people.	Relative advantage, compatibility.

it for others because the existing users benefit from the MSS even further if they are able to schedule meetings with more of their associates.

The success factors together represent what we envision the MSS to be. They can be combined into the following vision statement for the MSS:

A system that automatically utilizes the heterogeneous calendar systems of the users to help schedule meetings effortlessly and accurately over organizational boundaries while protecting the privacy of the users' calendar data.

The research goals for this thesis are to find solutions to the problems regarding the success factors of the MSS. We have condensed the research goals into the following research questions:

- RQ1** How to enable the scheduling of meetings between people who use heterogeneous calendar systems in a way that is more effortless than with currently existing tools or methods?
- RQ2** How to ensure that the scheduled meeting times are satisfactory to the users?

RQ3 How to enable the MSS to take advantage of a wide variety of existing calendar systems?

RQ4 How to avoid exposing too much user's private calendar information to other users while allowing the MSS to function effectively?

The contribution of this thesis is the comprehensive design and documentation of a software architecture for the MSS. We will also demonstrate the feasibility of the architecture by creating a prototype implementation of it and evaluating how well the implementation fulfills the objectives set for the architecture.

1.3 Scope

One of the success factors for the MSS discussed in Section 1.2 and the idea of the research question RQ3 was making the MSS integrable to a wide variety of different calendar systems. While we will be designing the architecture so that it allows integration with as many types of calendar systems as possible, we will primarily focus on integrating the MSS with calendars on mobile phones or other mobile devices. The reason is that this thesis is made as a part of a larger research project and mobile phones are the focus area of that project as well.

To further clarify the intended scope of this thesis, in this section we highlight some possible focus points relating to the problem area that we are deliberately leaving out of our research.

We will not focus on the design of algorithms for the various tasks within the MSS. The performance of the MSS is not something that we intend to study rigorously in this thesis and the optimization of the algorithms would mostly only contribute to performance. Instead, we are interested in researching the suitable distribution of responsibility between the users of the MSS and the automated process of the MSS itself. Therefore, in a sense, it is more important to us to know *what* the MSS does rather than *how* it does it.

We are not interested in doing user interface- or user experience research in this thesis. The final implementation of the MSS would surely benefit from having a well designed and intuitive user interface that provides sufficient guidance to the users since it is intended to be used by a large audience of non-experts in their daily life. Other possible features to improve the user experience could be, for example, taking holidays automatically into account when suggesting meeting times and providing travel schedules for the users based on the location of the meeting. However, since we are only

making a prototype implementation of the MSS architecture, and only seek to demonstrate that the basic functionality of the MSS is viable, we find that focusing on the user experience is unnecessary at this point.

Note, however, that while we are not focusing on the actual ease-of-use of the MSS, we are interested in the ease-of-deployment instead, as mentioned in Section 1.2. With ease-of-deployment we refer to how well the MSS supports different platforms and calendar systems and how the users of those systems can take advantage of the MSS. Ease-of-deployment could also mean the simplicity of the process of installing the MSS to a given platform but we will not be focusing on that problem since it is not as significant architecturally.

In Section 1.2, we mentioned how a person's calendar does not necessarily accurately imply when that person is available for a meeting and we stated two reasons for this. One was that people do not always record their complete schedule in their calendar. The other one was that people might want to meet certain people only at certain times, which could be referred to as the *social context* of the meeting. However, most often meetings also have other contexts that can influence whether a person is available for that meeting, such as the location or the topic of the meeting. The location can matter because a person needs time to get to the location from wherever she is before the meeting and there might not be enough free time in that person's schedule to travel there. The topic of the meeting could be relevant if users want to dedicate a certain time for meetings with certain topics or assign different priorities to different topics and only participate in meetings that are of high priority. However, in this thesis the only context that we will be focusing on is the social context. This is because we want to limit the complexity of the problem area that we are researching and we think that the social context is the most relevant one for our research.

1.4 Research methodology

Any research strategy in software engineering is composed of three different parts: research setting, approach and validation (Shaw, 2001). Research setting defines the problem that is being solved and it is primarily determined by the types of research questions that the researchers ask. Research approach refers to the types of results that are produced by the research to answer the research questions. Finally, the purpose of validation is to demonstrate that the research results are valid and useful. All of the three parts of the research strategy should be planned carefully so that they support each other. They should also be presented clearly in the research paper to effectively communicate the results to the readers (Shaw, 2003).

A classification of different types of research questions, results and validation techniques has also been developed by Shaw (2002). The type of all of the research questions in this thesis is *design of a particular instance* (Shaw, 2002) because all of the questions are about solving problems regarding a specific application rather than software development in general. Following this, the results of this thesis are of the type *specific solution* (Shaw, 2002) because we are designing a software architecture and a prototype for a specific application that incorporates the answers to our research questions. The type of validation that we employ in this thesis is *evaluation* (Shaw, 2002) because we perform the validation by evaluating how well the architecture design and the prototype match the requirements that were elicited for them. Additionally, we allow a group of experts to evaluate the feasibility of the architecture and how well it fulfills their needs for the system.

Because the research setting and the research approach of this thesis focus on the creation of a specific solution, we have chosen Design Science (DS) as our research methodology. DS is a suitable methodology for this thesis since its goal is to "create things that serve human purposes" (March and Smith, 1995). Many different frameworks for conducting DS research have been proposed (Hevner et al., 2004; Gregor and Jones, 2007) but the framework that we chose for this study is the one proposed by Peffers et al. (2007). Their framework is a combination of common elements from several prior DS frameworks.

The framework consists of the following six activities:

1. **Problem identification and motivation:** Define what actually is the problem under research and justify how a solution to it would be valuable (Peffers et al., 2007).
2. **Define the objectives for a solution:** Elicit objectives for the solution that it has to fulfill to properly address the previously identified problems (Peffers et al., 2007).
3. **Design and development:** Based on the objectives, design and create an artifact which can be applied to solve the problem (Peffers et al., 2007).
4. **Demonstration:** Use the artifact to solve an instance of the problem to demonstrate that it is appropriate for the task (Peffers et al., 2007).
5. **Evaluation:** Measure and compare the results from the use of the artifact with the objectives defined beforehand and evaluate how well the artifact fulfills its purpose (Peffers et al., 2007).

6. **Communication:** Communicate the background, methods, and results of the study to relevant audiences to spread the new knowledge (Peppers et al., 2007).

In the text below, we explain how each of these six activities is performed in this thesis.

We **identified the problems** with meeting scheduling through personal experience and discussions with other people. Finding a suitable time for a meeting has time and time again proven to be a tedious task that requires close communication between the participating people. The vast offering of various meeting scheduling and calendar applications would indicate that there is a demand for such things. This, in turn, speaks something about the prevalence of the problem. Despite the existence of a multitude of assisting applications, meeting scheduling is still often problematic in some situations, such as in cross-organizational scheduling, due to the limitations of the applications. We believe that many of the problems in the existing applications can be overcome and that the new solution would not only reduce the amount of manual work required but also save time and provide better results. The research problem and motivation are described in more detail in Section 1.1.

Since the problem that we are addressing was identified from personal experience, we were also mostly able to determine ourselves which are the necessary **objectives for the solution**. We decided that in addition to, of course, solving the obvious problems at hand, the new solution should also have other qualities that make it appealing to more people. Such qualities are, for example, the usefulness of the results, ease-of-deployment and privacy. To help us in defining the objective for the solution, we collaborated with our research partner on the industry side, Intelligent Precision Solutions and Services Ltd. (IPSS). IPSS is a company that provides technology solutions for businesses to manage their customer relationships and to utilize their customer data to increase sales. They are interested in possibly applying the results of this thesis to actual products and were able to provide us with a better perspective to the solution from the end user's point of view. The objectives that we defined for the MSS are explained in more length in Section 1.2.

We began **designing the artifact** by eliciting a number of more detailed requirements for the MSS based on the objectives that we had defined previously. The requirements were formulated partly from our own needs for the MSS and partly from the needs that we estimate a typical user of the MSS to have. Many of the requirements also came from IPSS and they were very helpful in providing us with insight into the end user needs. IPSS also validated all of the requirements to ensure that they are realistic from an end

user's point of view. We prioritized the individual requirements on a scale of high/medium/low based on how important we felt that the requirements were to achieving the purpose of the MSS. Once the detailed requirements for the MSS began to clarify, we started designing the software architecture by identifying what kind of functional components there would have to be in the MSS and what kind of information they would be handling. We decided, for example, what was the network topology between the different components and what tasks each of them would do. IPSS, again, helped us by providing their insight into architectural challenges that didn't have a single clear answer, such as the specifics of balance between user privacy and information availability. After the most significant decisions about the architecture had been made, we began developing the prototype implementation of the MSS. We chose only the most crucial features of the MSS, that are key to demonstrating the ideas behind the architecture, to be implemented to keep the required effort reasonable in the scope of this thesis. The technologies we used were selected mostly on the basis of what was familiar to us from previous experience. All of the parts of the design, the requirements; the architecture; and the implementation, were iterated over several times during the design process as the problem area became more familiar to us. The requirements that we elicited for the MSS can be found in Chapter 3. The software architecture for the MSS is documented in Chapter 4 and details about the implementation are described in Chapter 5.

We successfully implemented all of the important features of the architecture in the prototype, which **demonstrates** that the architecture design is suitable for solving the problem. However, we have not used the prototype in an actual usage scenario so we mainly evaluate the architecture through other methods.

We **evaluated** the artifact by validating and verifying the architecture and the prototype using various methods. Validation was a constant process alongside the architectural design process. We frequently presented the progress of the architecture design and the prototype to IPSS and they provided feedback and insight for the design process. Additionally, we held a larger, slightly more formal evaluation session at the end of the design process where the experts at IPSS evaluated the complete architecture as well as the prototype. We did verification by tracing the prototype implementation and the architecture back to the requirements and research questions which are the basis of the design. In addition, we compared the effortlessness of the MSS scheduling process against that of some other meeting scheduling applications. The results of our evaluation methods are documented in Chapter 6.

This thesis itself is the main method of **communicating** the knowl-

edge that we have gained in our research. The relevant audiences that would most likely benefit from this thesis are software architects and other technologically-oriented people. Chapter 1 should also give a clear picture of the problem area and an overview of the solution to less technologically-oriented audiences. Additionally, we have presented the problem, the architecture design and the prototype in poster sessions at two quarterly review events of the Finnish Cloud Software Program. The audience of the review events consisted of representatives from academic and software industry organizations.

1.5 Structure of the thesis

The rest of the thesis is structured as follows: In Chapter 2, we study the previous work that has been done in the field of meeting scheduling. This includes analyzing existing scheduling software applications and presenting some previous research based on the literature. In Chapter 3, we introduce the key features and the requirements for the MSS. Chapter 4 documents the architecture design for the MSS and also contains some discussion about the major architectural decisions that we have made. In Chapter 5, we describe the prototype of the MSS in detail and explain how we implemented it. In Chapter 6, the validity of the architecture is evaluated in various ways. Chapter 7 contains discussion about the results of the thesis. Finally, Chapter 8 concludes the thesis.

Chapter 2

Previous Work

The finding of a suitable time for a meeting is a very common practical problem and, to solve it, numerous calendaring and scheduling software applications have been created over the years. Meeting scheduling has also been the topic of some scientific research. In this chapter, we describe some of the meeting scheduling software that already exists today and highlight examples of previous research relating to the problem area.

2.1 Existing scheduling applications

The amount of different calendar or scheduling software applications available in the internet is vast. The applications can either be web services or installable on the users' own devices. Installable applications are available for desktop computer operating systems, such as Microsoft Windows (Microsoft Corporation, 2012b); Apple OS X (Apple Inc., 2012a) and Linux, and also for mobile operating systems, such as Apple iOS (Apple Inc., 2012b) and Google Android (Google, 2012a). In addition, mobile operating systems often have a built-in calendar application.

Not all applications offer actual meeting scheduling features. Many are simply electronic calendars that help users keep track of their own appointments. However, many applications do offer the functionality to simplify meeting scheduling. We divide those applications that have scheduling features into two categories. Applications in the first category allow users to share their calendar with other people. Applications in the second category allow one user to make time slot suggestions that other users can then mark as suitable or unsuitable.

Popular calendar applications Microsoft Exchange calendar (Microsoft Corporation, 2011) and Google Calendar (Google, 2012b) fall into the first

category (calendar sharing). While Google Calendar is a web service and Microsoft Exchange can be used either with a web or a desktop client, they both offer largely the same features. They provide the user with one or more calendars where the users can record their appointments and other events. Users can choose to allow other users who are in the same domain to have access to their calendar and see when they are busy or available. Domains are set up by system administrators and are usually limited to a single organization. The sharing of calendars makes it easy to schedule meetings as one user can see at a glance when every participant is available. This saves the participants from having to manually communicate their available times to each other. Microsoft Exchange and Google Calendar also allow the organizer to send meeting invitations to people who have not shared their calendar but the applications cannot assist in actually scheduling meetings with these people. The meeting invitations are sent via e-mail and the meeting details are included in a standard iCalendar format (Dawson and Stenerson, 1998). Other calendar applications that support the iCalendar format may then be able to add the meeting automatically to the participant's calendar.

Examples of the second category (manual time slot suggestions) are applications called Doodle (Doodle AG, 2012) and Meetin.gs (Meetin.gs Ltd, 2012). Doodle is a web service that allows anyone to create scheduling "polls" where they can initially select multiple date and time options for a meeting. The poll creator can invite other people to the meeting by providing them with a link to the poll. The other participants can then use the link to see the time slots proposed by the poll creator and can mark on each proposal if that time is suitable, unsuitable, or in some cases "possible if absolutely necessary" for them. The poll creator and optionally other participants are able to see all the answers given by the participants and based on these answers the organizer can select and confirm the time slot that she feels is the most suitable for everybody. A screenshot of Doodle can be seen in Figure 2.1. This arrangement simplifies the scheduling process because it removes the participants' need to communicate their times directly with each other. On the other hand, the selection of the time suggestions by the organizer and the marking of suitable times by the participants still requires a lot of manual effort from the users. Meetin.gs is another web service that uses the same scheduling principle as Doodle but also has other features related to organizing meetings, such as a messaging board and meeting material management capabilities.

There are also some scheduling applications that are not intended for meeting scheduling but instead offer more specific or refined scheduling features for certain environments or use cases. Hello Scheduling (Hello Industries, Inc., 2010), for example, is geared towards employee scheduling and

The screenshot shows the Doodle web service interface for a poll titled "Project iteration meeting". The poll was initiated by Olli Korjus and has 2 participants, 0 comments, and was started 28 minutes ago. The interface is in "Calendar view" for the "Europe/Helsinki" time zone. It displays a calendar for September 2012 with five time slots: 10:00 AM - 12:00 PM (Thu 20), 11:00 AM - 1:00 PM (Fri 21), 2:00 PM - 3:30 PM (Fri 21), 10:00 AM - 12:00 PM (Mon 24), and 1:00 PM - 3:15 PM (Thu 27). Three participants are listed: Ernesti, Jalmari, and Hermann. Ernesti has green checkmarks for the 10:00 AM - 12:00 PM and 2:00 PM - 3:30 PM slots. Jalmari has green checkmarks for the 11:00 AM - 1:00 PM, 2:00 PM - 3:30 PM, 10:00 AM - 12:00 PM, and 1:00 PM - 3:15 PM slots. Hermann has a checked checkbox for the 11:00 AM - 1:00 PM slot and a checkbox for the 10:00 AM - 12:00 PM slot. A "Save" button is located at the bottom right.

		SEPTEMBER 2012				
		Thu 20	Fri 21	Fri 21	Mon 24	Thu 27
		10:00 AM - 12:00 PM	11:00 AM - 1:00 PM	2:00 PM - 3:30 PM	10:00 AM - 12:00 PM	1:00 PM - 3:15 PM
2 participants	Ernesti	✓		✓		
	Jalmari		✓	✓	✓	✓
	Hermann	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
		1	1	2	1	1

Figure 2.1: A screenshot of Doodle web service. In this view an invited user can mark which times are suitable and unsuitable for her and see how other users have responded.

Table 2.1: Comparison of how different steps of the scheduling process are done in some of the existing meeting scheduling applications.

Application	Selecting participants	Finding free times	Agreeing on a time	Adding the event to a calendar
Microsoft Exchange	Any e-mail address.	<p>Calendars of other Microsoft Exchange users who are in the same domain are visible to the organizer so she is able to select a suitable time. The application also suggests times.</p> <ul style="list-style-type: none"> + No effort needed from participants + Free times are found immediately - No help for scheduling with people who are not in the same domain or use a different application - Must share at least free times from the whole calendar 	Organizer selects a time. Meeting invitation is sent via e-mail in iCalendar format which the participants can accept, accept tentatively or decline. Possible to reschedule.	<p>Added automatically by Microsoft Exchange and possibly other applications based on an iCalendar file included in the e-mail.</p> <ul style="list-style-type: none"> + No manual effort required
Google Calendar	Any e-mail address.	<p>Calendars of other Google Calendar users who have shared them are visible to the organizer so she is able to select a suitable time. The application also suggests times.</p> <ul style="list-style-type: none"> + No effort needed from participants + Free times are found immediately - No help for scheduling with people who have not shared their Google calendars - Must share at least free times from the whole calendar 	Organizer selects a time. Meeting invitation is sent via e-mail in iCalendar format which the participants can accept, accept tentatively or decline. Possible to reschedule.	<p>Added automatically by Google Calendar and possibly other applications based on an iCalendar file included in the e-mail.</p> <ul style="list-style-type: none"> + No manual effort required
Doodle	<p>Share a link by any means. Also possible to send e-mail directly from the service.</p> <ul style="list-style-type: none"> + Freedom to choose how to share the link 	<p>Organizer creates date and time suggestions. Participants mark which times are suitable for them.</p> <ul style="list-style-type: none"> + Participants do not need to share calendars - Organizer must guess good suggestions - Participants have to manually check and mark their free times - Could take a long time for participants to respond while situation might change 	Organizer selects one or more of the suggested times. Possible to change selection later.	<p>Manually. Possible to also export an iCalendar file which has the meeting data. Possible to add directly to Google Calendar. Possible to add directly to Microsoft Outlook or Apple iCal via plug-ins.</p> <ul style="list-style-type: none"> - Requires manual effort or plug-ins
Meetings	Any e-mail address.	<p>Organizer creates date and time suggestions. Participants mark which times are suitable for them.</p> <ul style="list-style-type: none"> + Participants do not need to share calendars - Organizer must guess good suggestions - Participants have to manually check and mark their free times - Could take a long time for participants to respond while situation might change 	Organizer selects one of the suggested times. Possible to change selection later.	<p>Manually. Possible to also export an iCalendar file which has the meeting data. Possible to add directly to Google Calendar.</p> <ul style="list-style-type: none"> - Requires manual effort

allows managers to plan work shifts and breaks for their employees. Another example is Mimosa scheduling software (Mimosa Software Ltd., 2012), which is intended for school and university scheduling with features such as course prerequisite checking and synchronizing study terms with the calendar.

Table 2.1 summarizes how the four steps of a meeting scheduling process are done in the four meeting scheduling applications that we introduced in this section. The advantages and disadvantages of each approach are also presented with green plus- and red minus points.

2.2 Previous research

While there has been much research about electronic calendar systems in the past (Kelley and Chapanis, 1982; Kincaid et al., 1985; Payne, 1993; Mosier and Tamaro, 1997; Blandford and Green, 2001; van den Hooff, 2004), the topic of meeting scheduling especially in mobile context has received less attention.

One study regarding mobile meeting scheduling is a user study by Starner et al. (2004) about what different devices people use to help them schedule meetings while mobile. The four most used devices in order were memory, scrap paper, paper-based planners and Personal Digital Assistants (PDAs). Their study showed that 36% of the people who claimed to use a PDA and 66% of the people who claimed to use a paper-based planner actually used memory or scrap paper to schedule a meeting. They hypothesized that people are more inclined to use a device with a shorter access time. This was supported by their measurements that paper-based planners and PDAs required the most time to retrieve the device and navigate to the right time in the calendar. Overall, the lower the access time was for a device, the more it was used in the meeting scheduling. On the other hand, people who relied only on memory confessed that they forget appointments more easily than those who use paper-based planners or PDAs (Starner et al., 2004). We would also like to add that while electronic calendars are often slower and more cumbersome to use than memory or scrap paper, we believe that the possibility to automate some tasks such as meeting scheduling or synchronization make electronic calendars an appealing option to use even in a mobile context.

The usage of mobile digital calendars has been studied on several occasions. Wu and Tremaine (2004) interviewed knowledge workers about the time management tools they use. In addition to mobile calendars, the people who were interviewed also used desktop electronic calendars and paper

calendars. The results showed that the people who were most satisfied with their time management tools used a combination of mobile and desktop calendars (Wu and Tremaine, 2004). The mobile tool naturally allowed them to access their calendar at any time while the desktop tool provided them with a better view of the calendar due to a larger screen. However, there were many complaints that the synchronization of the calendar between these tools does not always work well.

The motivations to use electronic calendars rather than paper-based ones has also been studied by Sell and Walden (2006) through interviews with a group of professionals. The research results highlighted three different categories of motivations. Some people have very heavy time management requirements and they do not believe that they could handle them without a digital mobile calendar (Sell and Walden, 2006). Other people have the desire to take advantage of a group calendar, which allows them to see the calendars of other people and other people to see their calendar (Sell and Walden, 2006). Thus, they use a digital calendar because they need some type of a digital tool to access the group calendar. Finally, some people have a personal interest in the technology of a digital calendar or they feel that using one is appropriate for their status or position (Sell and Walden, 2006). The usage of both mobile and desktop electronic calendars together was found to be popular also in this research and there were also some complaints about the problems with synchronization.

However, in further research of mobile digital calendar use among knowledge workers, it was revealed that the calendar load of the users did not affect their view of the mobile calendars' efficiency (Sell, 2008). The users' personal interest in technology also did not appear to have an effect on their satisfaction of using a mobile calendar (Sell, 2008). The factors that did increase the sense of efficiency of a mobile calendar were the mobility of the user's work, the mandatory use of a group calendar and the sense of image or status that the users associated with mobile calendars.

The topic of automatic meeting scheduling has been the focus of some research. The comparison of four meeting scheduling methods, including an automated scheduler, regarding their satisfaction and perceived efficiency was conducted by Higa et al. (1996). In addition to the automatic scheduler, the methods under study were face-to-face discussion, e-mail with a specific time table formatting and free-form e-mail. The results showed that the test subjects were more satisfied with the meeting time found through face-to-face discussions than those that came from the automatic scheduler. This was despite the fact that the automatic scheduler resulted in much fewer conflicts that forced the subjects to compromise on the meeting time than face-to-face discussions. Face-to-face discussions were also perceived to be

more efficient than both of the e-mail based methods and more satisfying than the formatted e-mail method (Higa et al., 1996). There was no indication in the study as to why the face-to-face discussions resulted in higher satisfaction than the automatic scheduler even though the discussions suffered from more numerous scheduling conflicts. We estimate that the habits and conventions used in scheduling and the technological level of the scheduler at the time when the study was conducted could explain this. However, we also assume that one reason is that people enjoy the feeling of being in control of their decisions and a completely automatic scheduler takes this control away from them. For this reason, to maximize the combination of satisfaction and efficiency of the MSS, we think that it is necessary to find a balance between automation and allowing the users to control the process.

There has been a lot of research regarding automatic meeting scheduling systems. Many researchers have built their own prototype scheduling systems to study some specific feature of the system. For example, Sen and Durfee (1994) have created a system that can adapt to the environmental conditions, such as the number of invited people and the system load, to provide the best performance in scheduling. Haynes et al. (1997) expanded this same system to allow users to set constraints and preferences, such as the topic; host; invited people and length of the meeting, into the system to determine which meetings are scheduled automatically, which ones are presented to the user for confirmation and which ones are rejected outright. Bilogrevic et al. (2011) have studied the effectiveness of different privacy-preserving algorithms in their meeting scheduling system where the potentially malicious central server and other participating devices try to decipher the calendar information of the user. Jeong et al. (1999) have developed a meeting scheduling system that can resolve meeting time conflicts by automatically rescheduling previously scheduled meetings. Dent et al. (1992) as well as Kozierok and Maes (1993) have experimented on intelligent software agents that can help their users to schedule meetings by automatically learning the habits and preferences of the users which allows the agents to make suggestions regarding future meetings.

Most of the automatic meeting scheduling systems in the literature function in a homogeneous environment. That is, all the participating people are using the same calendar system. Automatic meeting scheduling in a heterogeneous environment has been researched only a little. One such work is a conceptual model for an Intelligent Inter-organizational Meeting Scheduler (IIMS) developed by Glezer (2003). The IIMS allows different organizations to share knowledge, such as calendars; organizational charts; process flows; personnel records and goals, between each other even if they are using heterogeneous data storages. The organizations can subscribe to receive some type

of information from other organizations but to do so they must also publish that type of information from themselves to others. The different pieces of information in each organizations are mapped to keywords that allow different pieces from different organizations to be recognized as certain type of knowledge (Glezer, 2003). Enabling cross-organizational scheduling is also an important goal for the MSS but the IIMS approaches this problem from a slightly different perspective than we do. The IIMS connects the different organizations to each other as a whole and the people who use the IIMS are regarded only as parts of the organization. Conversely, we seek to remove the organizational boundaries from the system altogether and connect the people to each other as individuals.

Chapter 3

Requirements

This chapter documents the user needs and requirements that form the basis of the architectural design. Assumptions that we have made regarding the problem area are also stated in this chapter. Additionally, some terminology used in this thesis is explained.

3.1 Important features

In Section 1.2, we stated that the research goals for this thesis are to find solutions to the problems regarding the success factors that we identified for the MSS. Based on the success factors we formulated four research questions for this thesis. We answer these questions by designing a scheduling system that incorporates the solutions to our problems and we have identified a set of key features that such a system should have to succeed, taking into account the scope of the thesis that we set in Section 1.3. These features are:

- Finding suitable meeting times.
- Allowing users to organize meetings through the system.
- Enabling integration with existing calendar systems.
- Allowing users to customize which times from their calendar are used in the meeting scheduling based on the other participants of the meeting.
- Protecting the privacy of the users' calendar information.

The features listed here are only the high-level requirements for the MSS. More detailed lists of requirements are specified in Section 3.4.

3.2 Terminology

There are some specific terms that are used consistently throughout this thesis to refer to certain concepts or parts of the architecture. While they have been chosen to be as descriptive as possible, some of the terms might not be instantly understandable to readers. To avoid misunderstandings, these terms are explained in Table 3.1.

Table 3.1: Terminology used in this thesis.

Term	Description
MSS	The entire system that is designed in this thesis including all of its components: hardware, software etc.
Calendar system	Any device, application or other system that contains the calendar of one or more users. For example, a mobile phone or a Customer Relationship Management (CRM) system.
Central system	A dedicated component of the MSS that manages all the scheduling activity and to which all the calendar systems are connected to.
Scheduling process	The entire process of creating a meeting event with a suitable time slot. Includes gathering suitable times from users, selecting the time slot, sending meeting invitations and adding the meeting to the participants' calendar.
Invited user	A user of the system who has been invited to a meeting but has not accepted the invitation yet.
Participant	A user of the system who has accepted an invitation to a meeting.

3.3 Assumptions

The main purpose of this thesis is to design a software architecture and a prototype implementation for research purposes rather than a full-scale implementation for actual use. Therefore, some assumptions about the problem domain are made to simplify the challenge and to reduce the scope of the thesis. The assumptions are listed in Table 3.2.

Table 3.2: Assumptions made regarding the problem area.

ID	Description
AS01	Each user only has one calendar which contains all the calendar data that the users wish to be utilized by the MSS.
AS02	A calendar can only be accessed by the owner of the calendar or by users authorized by the owner, not by others.
AS03	It is possible to utilize the internet when transferring data.

Assumption AS01 is made because if the MSS had to read data from multiple calendars with possibly incomplete or conflicting data it would complicate the problem of gathering suitable meeting times too much to be tackled in this thesis. Notice that this assumption permits a user to have more than one calendar if only one of them is utilized by the MSS and the user herself is responsible for synchronizing the data between the calendars. Assumption AS02 is made because the concerns that we have about privacy in this thesis are only related to how calendar data is shared within the system and not how it is accessed by users outside of it. Even more importantly, there is very little we could do to prevent outside access to the users' calendar since they are on external systems which we cannot control. Assumption AS03 is made so that we don't have to concern ourselves with designing a custom infrastructure for the MSS. This assumption is a reasonable one to make since most of the mobile devices that we are focusing on also have the capability to connect to the internet.

3.4 Detailed requirements

The key needs that we have identified for the MSS are described in Section 3.1. In this section, those high-level requirements are divided into much more detailed functional requirements. The requirements have been given a priority on the scale of high/medium/low based on how relevant they are to the key needs for the MSS. A high priority means that the requirement must be fulfilled or otherwise the MSS does not do what is expected of it. A medium priority means that the requirement is not absolutely mandatory but ensures that the MSS functions in a sensible manner. Requirements of low priority are mostly nice-to-have features. The MSS can work well without them and they just make it easier to use.

For the sake of clarity and convenience, the list of requirements for the

MSS has been divided into five parts which are presented in the tables below. Each table contains requirements that roughly relate to one of the key needs of the MSS. The requirements are written using the Easy Approach to Requirements Syntax (EARS) (Mavin et al., 2009).

Table 3.3: Integration and privacy requirements.

ID	Description	Priority
R11	The MSS shall be integrable with external calendar systems in a way that allows the MSS to access the calendar information on the calendar system.	High
R12	The MSS shall have publicly available interfaces that are accessible by third party applications.	High
R13	The MSS shall allow third party applications to get, add and edit meeting scheduling related information through the interfaces.	High
R14	The MSS shall restrict users to see and edit only the information that is relevant to them through the interfaces.	High
R15	The MSS shall prevent users from directly accessing the calendar information of other users.	High
R16	The MSS shall never send the complete calendar information of the users (event details) outside of their calendar system, only the free time periods.	High

The requirements listed in Table 3.3 describe the integrability of the MSS as well as the needs for privacy protection. These aspects are very important for the MSS and are related to research questions RQ3 and RQ4.

Table 3.4: Group and available times customization requirements.

ID	Description	Priority
R21	The MSS shall allow the users to form groups with other users.	Medium
R22	The MSS shall allow the users to belong to any number of user groups (or none at all).	Medium
R23	The MSS shall allow group members to set their group to be joinable by everyone or just invited users.	Low

Table 3.4: Group and available times customization requirements. (continued)

ID	Description	Priority
R24	The MSS shall allow group members to set their group to be valid for only a certain period of time.	Low
R25	The MSS shall allow the users to specify before the scheduling process which periods of time from their calendar the MSS considers available when scheduling meetings with each different user group.	High
R26	The MSS shall allow the scheduling of meetings between users who aren't in the same group together.	Medium
R27	The MSS shall allow users to set what time periods are available for scheduling when scheduling a meeting between users who aren't in the same group.	Medium

The requirements in Table 3.4 relate to the possibility of the users to form user groups and to choose how they want to make their time available to those groups. These features are part of our efforts to ensure that the users are able to get satisfactory meeting times out of the MSS which is the problem area of research question RQ2.

Table 3.5: Meeting creation requirements.

ID	Description	Priority
R31	The MSS shall allow users to create meeting proposals.	High
R32	When a user has created a meeting proposal, the user shall be considered as the organizer for that meeting.	Medium
R33	When the organizer is creating a meeting proposal, the MSS shall require the organizer to enter at least the following information before the meeting proposal can be created: Participants, Topic, Date range, Location, Duration.	Medium
R34	When the organizer is creating a meeting proposal, the MSS shall require that the organizer selects a user group for the meeting.	High
R35	When the organizer has selected a user group for the meeting, the MSS shall add all the members of that group to the meeting as invited users.	Medium

Table 3.5: Meeting creation requirements. (continued)

ID	Description	Priority
R36	When the organizer is creating a meeting proposal, the MSS shall allow the organizer to add and/or remove individual users from the meeting before initiating the scheduling process, even if they were added as part of a group.	Medium

Table 3.5 contains the first half of the requirements that relate to the users' possibility to organize meetings with the MSS. These requirements specify how the users create new meeting proposals before the actual scheduling itself can commence. They are part of the basic scheduling functionality of the MSS which mostly falls under research question RQ1.

Table 3.6: Meeting time finding requirements.

ID	Description	Priority
R41	When the organizer has created the meeting proposal, the MSS shall present to the organizer a set of common time periods during which all the invited users can attend the meeting.	High
R42	When the MSS is gathering a set of common time periods, the MSS shall determine a user's availability at a certain time by reading data from that user's calendar.	High
R43	When the MSS is gathering a set of common time periods, the MSS shall consider a time to be available to a user if there is no other event at that time and if the user has specified that time to be available for the user group that is selected for the meeting being scheduled.	High
R44	The MSS shall present a set of common time periods to the organizer in less than 10 seconds after the organizer has created the meeting proposal.	Medium
R45	When the MSS has presented a set of common time periods to the organizer, the MSS shall allow the organizer to select a time slot that occurs within the common time periods.	High
R46	When the organizer has selected a time slot, the MSS shall add the time slot as the time slot for the meeting.	High

Table 3.6: Meeting time finding requirements. (continued)

ID	Description	Priority
R47	When the MSS has completed the scheduling process for a meeting and if no common time periods have been found, the MSS shall present to the organizer a set of the time periods when the highest amount of invited users are able to attend.	Medium
R48	While the scheduling process is on-going, the MSS shall allow all time slots used in the process to be available to other simultaneous scheduling processes as well.	Low

The most important aspect of the MSS is that it operates on data that is taken directly from the calendars of the users, which is part of providing the simplicity talked about in research question RQ1. The requirements in Table 3.6 specify this aspect and also describe the organizer's responsibility to select the actual time slot for the meeting. This is an effort to try and provide users with more satisfactory meeting times by allowing a human to contribute to the scheduling process instead of only relying on a completely automated algorithm. Satisfactory results are the focus of research question RQ2.

Table 3.7: Meeting notification and response requirements.

ID	Description	Priority
R51	When the organizer has selected a time slot for the meeting proposal, the MSS shall notify the invited users about the proposed meeting.	High
R52	When the MSS has notified the invited users about the proposed meeting, the MSS shall allow the invited users to view information about the proposed meeting's topic, time, duration, location and invited users.	Medium
R53	When the MSS has notified the participants about the proposed meeting, the MSS shall allow the invited users to choose if they accept the proposed meeting, reject it or request that a new time slot is selected for the meeting.	Medium
R54	When an invited user has accepted the proposed meeting, the MSS shall add the proposed meeting to the participant's calendar.	High

Table 3.7: Meeting notification and response requirements. (continued)

ID	Description	Priority
R55	When the MSS has notified the participants about the proposed meeting, the MSS shall allow the organizer to see which invited users have accepted, rejected or not yet responded to the meeting notification.	Medium
R56	When the MSS has notified the invited users about the proposed meeting, the MSS shall allow the organizer to change the topic and the location of the proposed meeting or cancel the proposed meeting at any point.	Medium
R57	When the organizer has canceled the meeting event, the MSS shall remove the meeting event from the calendars of the participants.	Medium
R58	When the organizer has changed the topic or the location of a meeting event, the MSS shall change the meeting events in participants' calendars accordingly.	Medium
R59	While the MSS has notified the invited users about the proposed meeting, the MSS shall allow the organizer to send reminders to the invited users about responding.	Low
R510	When the MSS has created a meeting event, the MSS shall allow each participant to exclude herself from the meeting event.	Low
R511	When a participant has excluded herself from a meeting event, the MSS shall remove that participant from the meeting event's list of participants as well as remove the meeting event from that participant's calendar.	Medium

Table 3.7 contains the second half of the requirements that relate to the meeting organizing functionality. In contrast to the other requirements on meeting organizing in Table 3.5, these ones specify what should happen *after* the actual meeting scheduling has happened. These requirements also relate to the basic scheduling functionality of research question RQ1.

Chapter 4

Architectural Views

In this chapter, the architecture for the MSS is presented using *architectural views* (Rozanski and Woods, 2011).

Architectural views are a way of describing a software architecture by dividing it into multiple separate representations of which each presents a different aspect of the architecture (Rozanski and Woods, 2011). This is done in order to avoid the architecture being represented with a single over-complicated model that has too much information to be useful. Each view usually focuses on one aspect of the architecture, for example what functional components are in the system or what information is transmitted within the system. Together the views describe the system as a whole.

To describe the significant architectural design decisions in this thesis, we have chosen to represent the architecture with four views: the functional view, the information view, the process sequence view and the integration view. Each view is presented in detail in the following sections and the rationale behind our architectural design decisions is discussed. Before the views, there is also a context diagram that clarifies in what kind of environment the MSS operates in.

4.1 Context diagram

The main decisions that we have made regarding the context that the MSS operates in are summarized in Table 4.1.

The context diagram in Figure 4.1 visualizes what different actors the MSS interacts with during its operation. In the diagram, the **Meeting scheduling system** denotes the MSS itself.

The MSS, of course, interacts with different types of users. **Organizer** is a special role that can be given to any user when they create a new meeting.

Table 4.1: Decisions regarding the context of the MSS.

Decision	Rationale
Utilize the external, existing calendars of the users.	Utilizing the existing calendars of the users is one of the key requirements for the MSS. An alternative solution would be to provide a proprietary calendar for the users.
Utilize an external notification agent.	Users are accustomed to receiving notifications about meetings through e-mail. An alternative solution would be to develop a custom notification method to the MSS itself.
Provide public interfaces through which third-party systems can utilize the functionality of the MSS.	Allows third-party systems to combine meeting scheduling with additional meeting-related functionality, such as resource or customer relations management.

Organizers can decide which users to invite to the meeting and select the final meeting time. **Invited user** is a person who has been invited to a meeting by the organizer. **Administrator** is a special user who has free access to the inner workings and data of the MSS for the purpose of maintaining it.

Additionally, the MSS must also interact with some external systems to operate. The most important of these is the **User's calendar** from which the MSS gathers calendar data to see when the user is available for a meeting. Note that the user's calendar component in the context diagram refers to the actual calendar itself, not the calendar system that it may reside in. The physical boundaries between the calendar system and the other components of the MSS are not visible in the context diagram. The meeting scheduling system element in the context diagram includes the components that are present in the users' calendar systems. The user's calendar can reside in any type of calendar system, for example on a mobile phone or an internet service but, as stated in assumption AS1 in Section 3.3, each user is assumed to have only one calendar. The **Notification agent** is a service that notifies the users when they have been invited to a meeting. The notification agent must also provide some way for the user to respond to the invitation. The notification agent is an external system, such as an e-mail service or a notification application in the user's mobile device. A custom notification agent could also be built into the MSS itself but, because there are other existing suitable notification channels, we do not think this is necessary. A

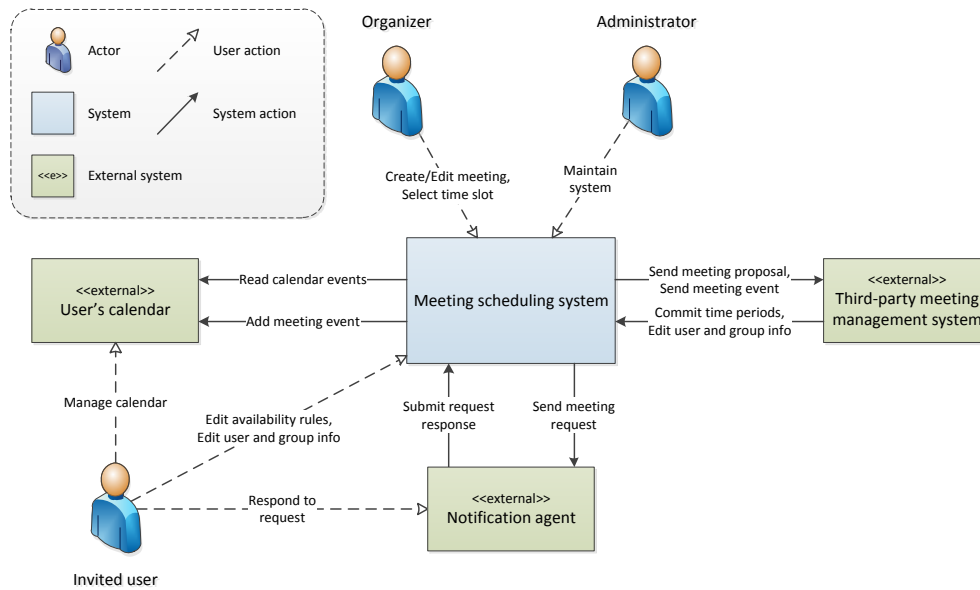


Figure 4.1: A context diagram of the MSS. Involved actors and how they interact with the MSS.

Third-party meeting management system is a larger external system, such as a CRM system, that can be optionally integrated into the MSS. The third-party meeting management system can act as a calendar system if it contains the calendar of one or more users. In this case, the MSS would gather available time periods from the meeting management system instead of from a personal calendar system. The meeting management system can also replace some of the User Interface (UI) components of the MSS. For example, the meeting management system could allow users to create and manage meetings or to respond to meeting requests through the UIs it provides instead of the ones provided by the MSS.

It is important to remember that, even though the user's calendar and the third-party meeting management system components are depicted in the context diagram only once, there can be and, in fact, should be more than one calendar integrated into the MSS simultaneously. There can also be multiple third-party meeting management systems integrated into the MSS alongside multiple personal calendar systems.

4.2 Functional view

The purpose of a functional view is to describe the architectural elements of the MSS, their responsibilities and the interfaces and interactions between them (Rozanski and Woods, 2011). In this section we present the functional view of the architecture by stating the main architectural decisions that we have made regarding the functional composition of the MSS and describing the functional elements of the MSS as well as their interfaces.

The decisions regarding the composition of the functional elements of the MSS are summarized in Table 4.2.

Table 4.2: Decisions regarding the functional elements of the MSS.

Decision	Rationale
Centralized topology with a separate dedicated central system.	A single location is needed to compute calendar information. We also have full control over a dedicated system's specifications. Alternative peer-to-peer and ad-hoc master device topologies do not offer both of these benefits.
User device is responsible for calendar access and availability rule management.	Calendar data and availability rules are user specific and private.
Central system is responsible for data management, scheduling and notifications.	Central system is the most reliable component, has guaranteed resources for computation and has best access to data.
Organizing and user management UIs are separate and replaceable.	Allows third-party meeting management systems to replace the UIs with their own solutions.

For the MSS, we chose a network topology where all the calendar systems are connected to a single dedicated central system. The central system manages all connections to the calendar systems and stores all of the data that is needed in the scheduling processes. We chose this approach because having a separate central system means that we have full control over its hardware capabilities, connections and availability. This allows us to reliably plan which tasks and responsibilities we assign to it. We could not do this if we had to rely only on the various calendar systems of the users since these systems can vary greatly in technical aspects. For example, the calendar

system could be a mobile device with very limited processing power and unreliable availability in which case it would be a poor choice to assign some demanding or long-term task to it. Additionally, having a central system is a benefit in the scheduling process because the calendar information that is collected from the calendar systems can be gathered to a single location. This makes it easy to calculate the commonly suitable times. Alternative topology choices include a peer-to-peer network where the calendar systems are connected to each other as equals or a topology where one of the calendar systems takes a coordinating master device role.

Figure 4.2 shows the functional view of the MSS architecture. The view displays all the logical functional elements of the MSS and how they are connected to each other. All of the elements in the functional view correspond to the elements in the context diagram in Figure 4.1 with everything inside the *system boundary* in the functional view being part of the *meeting scheduling system* element in the context diagram. The exception to this is that the *third-party meeting management system* element is displayed as a large outline over multiple functional elements in the functional view instead of as its own element as it is in the context diagram. This is because, as stated in Section 4.1, the *third-party meeting management system* can act as a calendar system and/or replace some of the UI components of the MSS. If the meeting management system acts as a calendar system, it must implement all of the components inside the *User device boundary*. The meeting management system can also implement the *User management UI*, the *Group management UI* and the *Organizer UI* or a combination of these.

As displayed in the functional view, there are four types of functional elements in the MSS: central system elements, user device specific elements, independent system elements and external elements. *Central system elements* are all part of the central system and there is only one of each of the central system elements in the MSS. *User device specific elements*, on the other hand, exist on each of the calendar systems of the users so there are multiple instances of each user device specific element simultaneously in the system. Meanwhile, *independent system elements* are not tied to any physical component and as such they can be implemented either on the central system or on the calendar systems. Indeed, it is also possible that the independent elements are implemented both on the central system and the calendar systems, in which case the users could choose which ones they want to use. Finally, *external elements* are already existing systems or other components that are not directly part of the MSS but with which the MSS must communicate to exchange necessary information.

The physical location, the responsibilities and the public interfaces of each functional element are summarized in the tables below. Alongside the tables

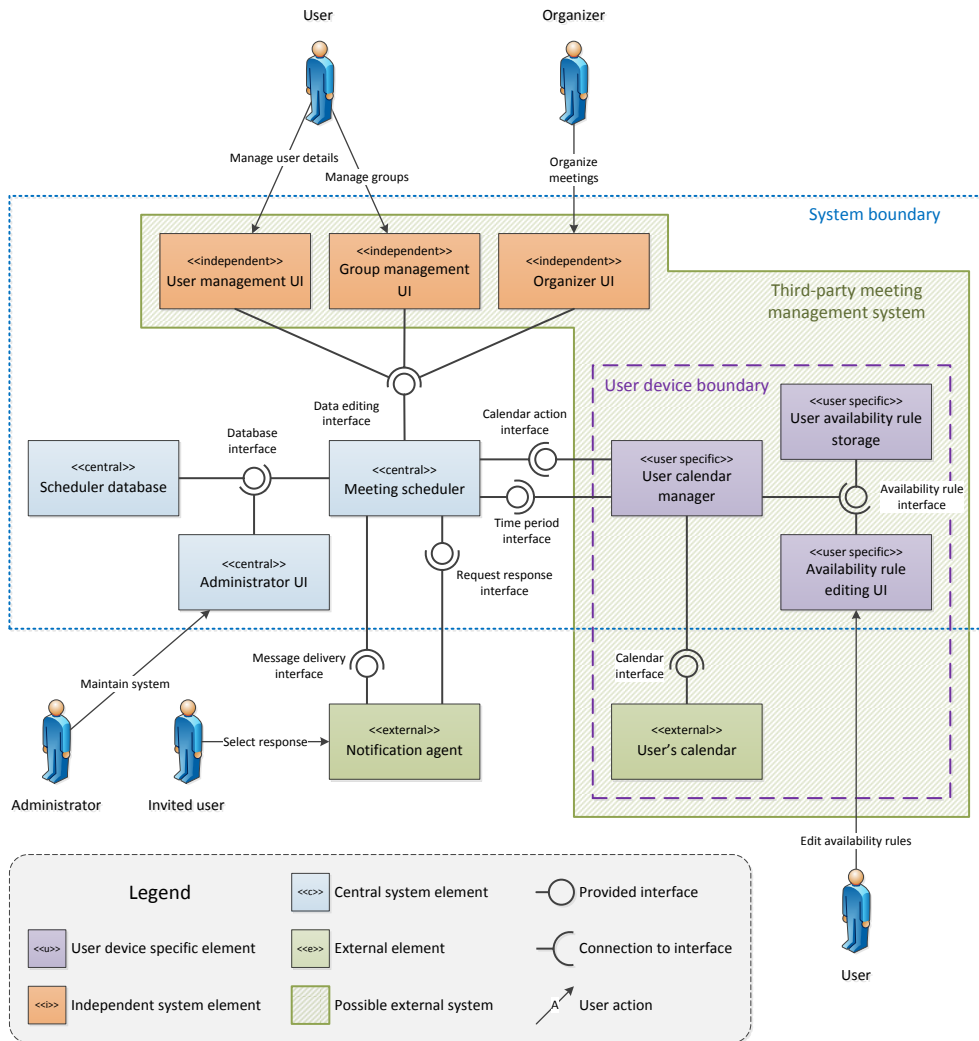


Figure 4.2: A functional view of the MSS displaying the different functional elements and how they are connected to each other.

we present the rationale behind some of the important design decisions we have made regarding these functional components. The public interfaces are also described in more detail in Section 4.5.

Meeting scheduler	
Location	Central system
Responsibilities	<p>Creates new meeting proposals based on the input from the <i>Organizer UI</i>.</p> <p>Calculates the common time periods from the individual available time periods which are received from calendar systems of invited users.</p> <p>Creates and sends meeting requests based on the time slot selection from the <i>Organizer UI</i>.</p> <p>Gathers the meeting request responses sent by the invited users and sends a new meeting event, an updated meeting event or instructions to remove a meeting from the user's calendar based on the response.</p>
Public interfaces	<p>Provides <i>Data editing interface</i> for editing user, group and meeting data.</p> <p>Provides <i>Time period interface</i> for committing time periods.</p> <p>Provides <i>Request response interface</i> for submitting responses to meeting requests.</p> <p>Connects to <i>Calendar action interface</i> provided by <i>User calendar manager</i> to send meeting proposals and meeting events.</p> <p>Connects to <i>Message delivery interface</i> provided by <i>Notification agent</i> to send meeting requests.</p>

The *Meeting scheduler* contains all of the logic of the central system and, as stated in Table 4.2, the central system is responsible for all of the actual scheduling functionality of the MSS.

For example, the commonly suitable time periods are calculated on the *Meeting scheduler* because the calculations require the user specific time periods as input. The time periods are gathered from the users' calendar systems. Therefore, it makes sense to perform the calculation in the central system where all the calendar systems are connected to. Additionally, the calculations can require a non-trivial amount of computing power if there are many participants in the meeting and the desired date range for the meeting is long. In these cases the guaranteed resources of the central system are a

benefit.

Scheduler database	
Location	Central system
Responsibilities	Stores all the data that is used in the operation of the meeting scheduling system such as user and group data as well as meeting events, time periods and request responses.
Public interfaces	N/A

A database is necessary on the central system because it is most rational to store certain pieces of information there, such as user list and meeting events. All the information elements in the MSS and their storage locations are explained in Section 4.3.

Administrator UI	
Location	Central system
Responsibilities	Allows system administrators to freely edit the data in the database in order to maintain the system.
Public interfaces	N/A

The *Administrator UI* is connected to the *Scheduler database* via the *Database interface* directly because it gives the administrators more freedom to maintain the MSS as they can access all the data freely. It is a reasonable assumption that the administrators will not abuse this power to corrupt the data. This is in contrast to the *User management UI*, the *Group management UI* and the *Organizer UI* that must access the data through the *Data editing interface* which restricts the user's ability to edit the data in order to protect its integrity.

User management UI	
Location	Central system or calendar systems
Responsibilities	Allows users to edit the data about themselves in the system and anyone to create new users into the system.
Public interfaces	Connects to <i>Data editing interface</i> provided by <i>Meeting scheduler</i> to edit user data.

Group management UI	
Location	Central system or calendar systems
Responsibilities	Allows users to create new groups, join existing groups and edit the details about the groups they belong to.
Public interfaces	Connects to <i>Data editing interface</i> provided by <i>Meeting scheduler</i> to edit user group data.

Organizer UI	
Location	Central system or calendar systems
Responsibilities	Allows users to enter the information required to create new meeting proposals. Users can also see information about and edit the meetings that they are the organizer of, for example, to select the time slot for the meeting, change the location or topic of the meeting or see how invited users have responded to the meeting requests.
Public interfaces	Connects to <i>Data editing interface</i> provided by <i>Meeting scheduler</i> to edit meeting data.

As stated in Table 4.2, *User management UI*, *Group management UI* and *Organizer UI* are independent elements, meaning that they can be implemented both on the central system side and the calendar system side. This is because their function does not rely on features of the central system or the user specific data on the calendar systems.

User calendar manager	
Location	User's calendar system
Responsibilities	Finds the available time periods for a user by reading the calendar events from the user's calendar and applying the appropriate availability rules to them. Adds and edits the meeting events scheduled by the MSS in the user's calendar.
Public interfaces	Provides <i>Calendar action interface</i> for sending meeting proposals and meeting events. Connects to <i>Time period interface</i> provided by <i>Meeting scheduler</i> to commit time periods.

User's calendar	
Location	User's calendar system, external
Responsibilities	Stores the user's calendar events.
Public interfaces	N/A

The *User calendar manager* contains all of the logic of the calendar systems and, as stated in Table 4.2, the calendar systems are responsible for the calendar access.

For this reason, the finding of available time periods is done in the *User calendar manager* as it utilizes the calendar data from the *User's calendar*. The calendar data must never leave the calendar systems of the users for privacy reasons, as stated in requirement R16, so any task that utilizes this data must happen within the calendar system.

We chose to also use the *User calendar manager* element to add the scheduled meeting events to the participant's calendar. This is because the access that the *User calendar manager* already has to the calendar to read the free times can easily be used to also write events into the calendar.

An optional method to deliver the scheduled meeting events to the participants is to publish the meeting to the calendar systems in a standardized format. Many calendar systems and applications already support some standardized formats and are able to add meeting events to their calendar based on the data. One such format is iCalendar (Dawson and Stenerson, 1998) which is supported by many of the scheduling applications that we inspected in section 2.1. However, we decided not to utilize the standardized formats in the architecture because the existing calendar connection that we already have is perfectly suited for this task.

User availability rule storage	
Location	User's calendar system
Responsibilities	Stores the availability rules of a user.
Public interfaces	N/A

Availability rule editing UI	
Location	User's calendar system
Responsibilities	Allows users to create and edit their availability rules.
Public interfaces	N/A

The *User availability rule storage* is on the user's calendar system because the availability rules are specific to each user. Because the availability rules are stored on the calendar system, it is also most sensible to also edit the availability rules directly on the calendar system, which is why the *Availability rule editing UI* is also located there. Availability rules are discussed in more detail in Section 4.3.

Notification agent	
Location	N/A (external)
Responsibilities	Distributes meeting requests to invited users. Allows users to respond to the meeting requests.
Public interfaces	Provides <i>Meeting delivery interface</i> for sending meeting requests. Connects to <i>Request response interface</i> provided by <i>Meeting scheduler</i> to submit responses to meeting requests.

The functionality of the *Notification agent* could be implemented within the MSS as well, but utilizing an external system in this case is justified because there are many existing communication channels that fit its purpose, such as e-mail or SMS. People are also already accustomed to actively monitoring these channels so they are unlikely to miss any notifications that they receive through them. If a custom notification solution was built, the users would have to learn to also check for messages that are sent through this new channel.

4.3 Information view

The purpose of an information view is to describe what information there is in the system and how it is stored, manipulated and distributed (Rozanski and Woods, 2011). In this section we state the main decisions that we have made regarding information in the MSS. We also list all of the elements of information in the MSS and explain how are they used in the various tasks of the MSS and where in the MSS they reside.

The main decisions that we have made regarding the information elements of the MSS are summarized in Table 4.3.

The calendar information that is used in the meeting scheduling is read from the calendar systems of the users but, since these systems are mobile phones or computers in personal use of the users, their availability cannot

Table 4.3: Decisions regarding the information in the MSS.

Decision	Rationale
Calendar data and availability rules are stored on the calendar systems. Only free times filtered with availability rules are sent to central system.	Calendar data and availability rules are user specific and private. There is also a requirement that calendar data never leaves the calendar systems as-is.
Time periods are calculated just-in-time and are not synchronized to the central server.	Because of the availability rules, the synchronization process would be too complex when compared to the achieved benefit.

be guaranteed. This will likely result in a situation where the MSS is unable to access one or more of the calendars that it needs to during the meeting scheduling process. Thus, in this situation the MSS cannot provide an accurate information about the availability of the invited users to the meeting organizer.

This kind of situation could be avoided if, instead of reading the calendar information from the calendars just-in-time for the scheduling process, the calendar information of every user would be stored on the central system. This way, the MSS would always have access to the calendar information of the users. However, this arrangement creates many difficulties that must be considered before deciding to include it in the architecture. First, according to requirement R16, the MSS must never send the complete calendar information of the users outside of their calendar system. This means that only the free time periods of the users could be stored on the central server. But, as stated in requirement R25, the users can specify which of the free times they want to make available to which user group. This, in turn, means that the availability rules must also be stored on the central system for each user so that the MSS is able to calculate the free times for each group. Alternatively, the central system could always store separate free time periods of each user for each group but this would increase the amount of calendar information that needs to be stored many-fold. Second, the calendar systems would be required to frequently synchronize their free times with the central server. This increases the amount of network traffic between the central system and the calendar systems. Third, the storing of calendar information on the central server sets extra demands for its hardware capabilities, most importantly the data storage.

We decided not to synchronize the users' calendar information to the central system. We consider the complexity of the synchronization process to be too great compared to the benefit that it provides. We believe that, in the case that the calendar information from some calendar systems cannot be acquired, it is sufficient to inform the meeting organizer about this so that she can decide the best course of action to mitigate the problem.

The information view of the MSS is illustrated in Figure 4.3. The elements and tasks in the figure are inside the functional elements in which they are stored or performed. All the different elements of information are also described in Table 4.4. In the text below we justify our decisions about where in the MSS the various elements are stored.

The **calendar data** and the **availability rules** are stored on the calendar systems of the users in the *User's calendar* and the *Availability rule storage*. This decision is stated in Table 4.3.

Both the **user data** and the **user group data** are stored in the *Scheduling database* on the central system. This is the most sensible solution because this information is used by multiple different functional elements such as the *User management UI*, the *Group management UI*, the *Organizer UI* and the *User calendar manager*. When they all read and update this data in one single location, it can be guaranteed that it is always valid and up-to-date.

Time periods and **common time periods** are information that is only needed temporarily in the MSS. For the time that they are used they are stored in the *Scheduler database* on the central system. This is because, although the time periods are theoretically only needed immediately after a meeting proposal has been made, in practice the *Meeting scheduler* needs to wait until all of the calendar systems have committed the time periods. The already received periods must be stored somewhere during that time. The common time periods, on the other hand, need to be stored until the organizer has selected a time slot for the meeting. The organizer is free to do this at any time so it might not happen immediately after the calculation is complete.

The meeting data objects referred to as **meeting proposals**, **meeting requests** and **meeting events** are stored in the *Scheduler database* during the scheduling process. They are kept in the storage also after the process so that the data about all the meetings in the MSS stays consistent and so that it is possible to refer back to the meeting data and make changes to it if necessary. It would have been an option to store the final meeting events only in the calendar of the users but this would have made it difficult to make any changes to the meetings afterwards since there would be no central repository where all the calendar systems could get the changes from.

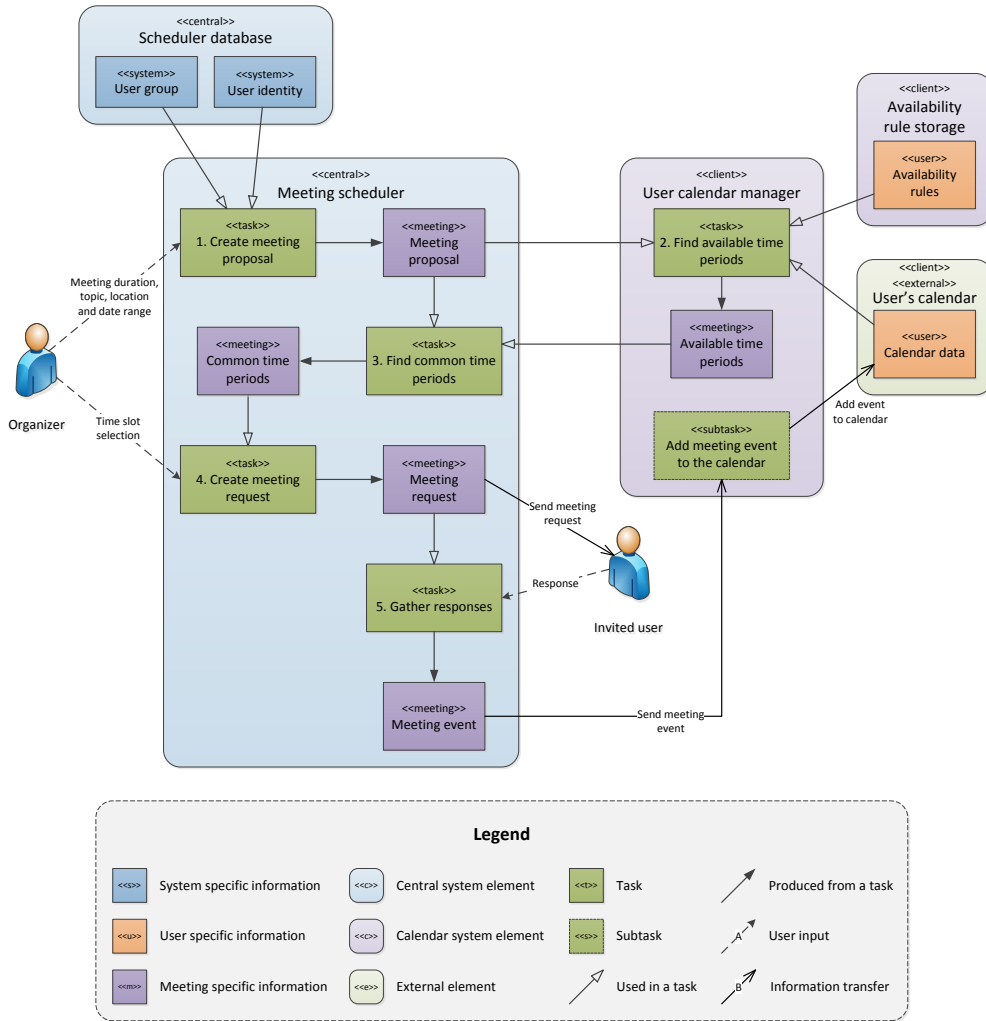


Figure 4.3: An information view of the MSS. The view shows what elements of information there are in the system, in which functional elements they are stored and what tasks they are used in and produced from. The tasks in the view are numbered to indicate the order in which they happen in the scheduling process.

Table 4.4: Information elements in the MSS.

Element name	Storage location	Description	How it is used
Calendar data	User's calendar	Information about the user's calendar events.	<i>User calendar manager</i> uses this data and availability rules to find available time periods for the user.
User data	Scheduler database	Information about registered users. Must at least contain enough information to distinguish different users from each other and restrict what information they can access in the MSS.	<i>Organizer UI</i> presents the user information to the organizer so she can select who to invite to the meeting.
User group data	Scheduler database	Information about what groups exist in the MSS and which users are in which group.	<i>Organizer UI</i> presents the user groups to the organizer so she can select which group the meeting is intended to. <i>User calendar manager</i> uses this information to select which availability rules to apply when finding time periods.
Availability rules	Availability rule storage	The preferences of the users about what times periods from their calendar they want to make available to which user group during the meeting scheduling process.	<i>User calendar manager</i> uses these and the calendar data to find available time periods for the user.
Time period	Scheduler database	Periods of time when a single user is available for a specific meeting.	<i>Meeting scheduler</i> uses time periods from all invited users to find the common time periods for the meeting.
Common time period	Scheduler database	Periods of time that are available for all the invited users of the meeting.	Presented to the meeting organizer in the <i>Organizer UI</i> to allow her to select a suitable time slot for the meeting.
Meeting proposal	Scheduler database	A data object of the meeting that contains only enough information to start the scheduling process.	<i>Meeting scheduler</i> sends this to the <i>User calendar manager</i> of each invited user to tell it to commit time periods for that user.
Meeting request	Scheduler database	A meeting proposal which also has a time slot.	Sent to invited user through the <i>Notification agent</i> to invite them to the meeting and allow them to respond to the invitation.
Meeting event	Scheduler database	A meeting that appears in the calendar of the users as a result of the scheduling process.	<i>Meeting scheduler</i> sends this to the <i>User calendar manager</i> of each participant so that the event can be added to their calendar.

4.4 Process sequence view

The purpose of this view is to clarify how the scheduling process unfolds by showing the sequence of interaction between the different functional elements. This is important because the scheduling process is the fundamental functionality of the MSS and it requires quite a specific order of interaction between many different and varying components.

One important decision about the scheduling process is highlighted in Table 4.5.

Table 4.5: Decision regarding the adding of the meeting event to calendars.

Decision	Rationale
Meeting events are added to the calendar as soon as the participant has accepted the invitation.	Alternative solution is to wait for the organizer to confirm the meeting and add it simultaneously to all participants' calendar. However, there is a danger that the participants' calendar changes between accepting the invitation and confirmation.

The operation of the MSS is divided into multiple individual scheduling processes. Each scheduling process consists of all the actions needed to schedule a single meeting, starting from the creation of the meeting and ending in the meeting being in the participant's calendars in a suitable time slot. There can be multiple scheduling processes going on in the MSS simultaneously but they all happen mostly in isolation from each other. Note, however, that in this thesis we will not consider the management of the concurrency and transactions of multiple simultaneous scheduling processes. This is because they are not a trivial issue and are not related to our research questions.

Some of the functionality of the MSS also takes place outside of the scheduling process, such as the editing of availability rules and managing the information of groups and users.

The order of these tasks and the interaction between the functional elements during the scheduling process is described in a sequence diagram in Figure 4.4. The functional elements *User management UI*, *Group management UI*, *Administrator UI* and *Availability rule editing UI* are not visible in the diagram because they are not used during the scheduling process. The interactions between the elements happen over the various interfaces between those elements. The functional elements and their interfaces can be seen in the functional view in Section 4.2. The usage of the interfaces between the

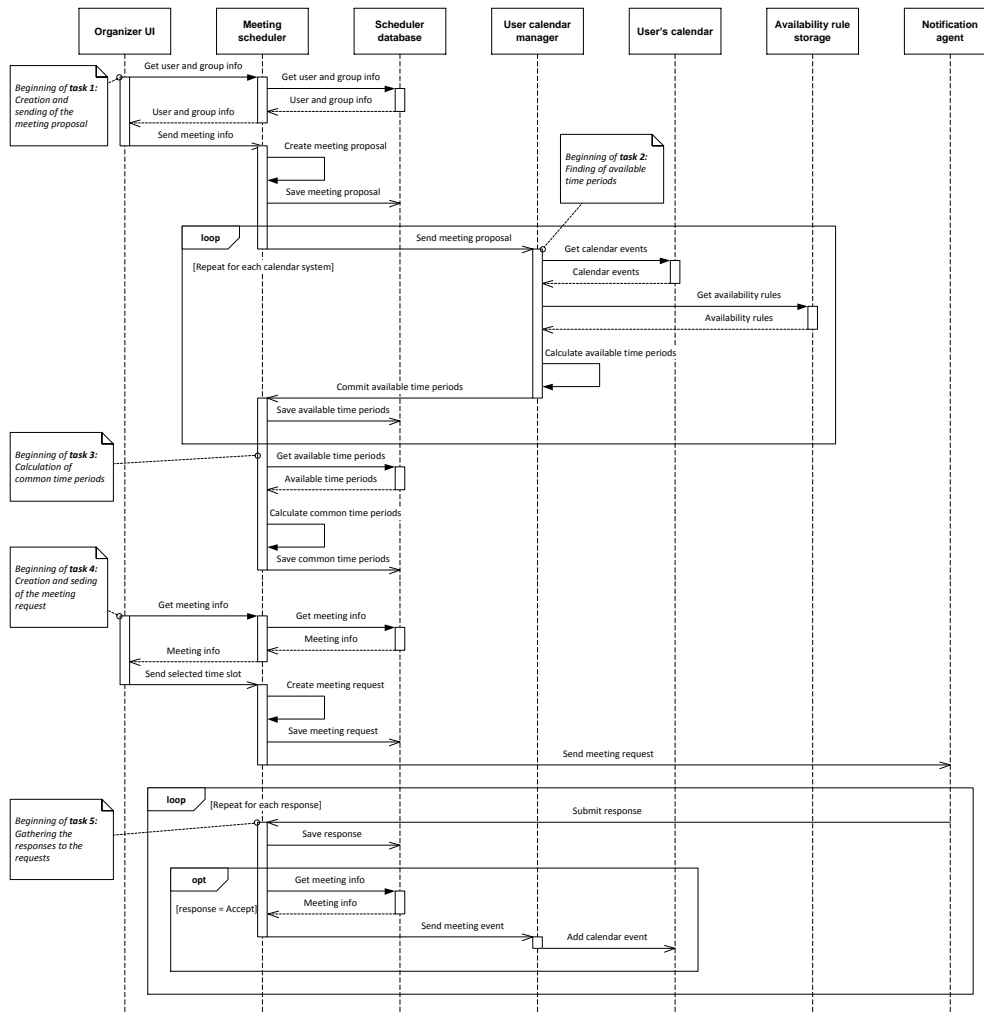


Figure 4.4: A high-level sequence diagram showing the order of interaction between the functional components during the scheduling process for a single meeting. Some of the UI components are not included in the sequence diagram since they are not required during a scheduling process.

central system and the other elements is explained in more detail in Section 4.5.

4.5 Integration view

Integrating the MSS with external calendar systems is one of the key features of the MSS that we identified in Section 3.1. The integration problem is two-sided: First, the calendars that reside within the various calendar systems of the users need to be integrated into the MSS so that it is possible to read the calendar data from them, as stated in requirement R11. Second, the calendar systems need to be integrated into the central system so that they can exchange data, as stated in requirements R12 and R13. In this integration view section, we describe how these two aspects of the problem are handled in the architecture.

Integrating the calendars of the users into the MSS is mostly an implementation problem. The issue revolves around the *Calendar interface* through which the *User calendar manager* reads data from the *User's calendar*. How this interface is provided in practice by the *User's calendar* depends entirely on the technical aspects of the calendar system in which the calendar resides. Since the MSS is meant to support theoretically any kind of calendar system, it is very difficult to determine the exact requirements for the calendar integration at the architectural level. Therefore, the problem must be solved at the implementation level by implementing the *User calendar manager* separately for each different type of calendar system so that it can utilize the provided *Calendar interface*.

We approached the second part of the integration problem - the integration of calendar systems to the central system - by providing the same common interfaces from the central system to all calendar systems. Creating only one set of interfaces for all calendar systems is considerably simpler than creating customized interfaces for each possible calendar system. This is also the obvious choice for us since we cannot know the technical specifications of the possible calendar systems in advance so it is not even possible to design interfaces for them. But, of course, requiring all the external calendar systems to conform to the same common interfaces means that some calendar systems might not be able to do so due to their technological constraints that we have no control over. However, we have decided that this is an acceptable limitation and it can be mitigated with correct technological choices on the central system's side in the implementation phase.

The decisions regarding the integration of three types of calendar systems into the MSS are summarized in Table 4.6.

Table 4.6: Decisions regarding the integration capabilities of the MSS.

Decision	Rationale
Personal calendar systems can be integrated by implementing the user device specific functional elements on the calendar system.	We designed the architecture primarily around this scenario because mobile phones are a focus area of this thesis.
Multi-user calendar systems can be integrated by utilizing the Calendar action and Time period interfaces.	This scenario is otherwise identical to the personal calendar system scenario except that the client component might require additional features to distinguish different users.
Web calendar services can be integrated by creating a proxy component between the interfaces of the MSS and the web service.	Our focus in this thesis was to integrate calendar systems, not web calendar services. This is why the architecture is not designed to directly integrate web services.

There are three main types of calendar systems that could be integrated into the MSS. The first one is a personal calendar system, such as a mobile phone or a desktop calendar application. The second one is a multi-user calendar system, such as a large CRM system or a calendar server. The third one is a web calendar service, such as Google Calendar (Google, 2012b). Personal and multi-user calendar systems can be integrated in a similar way by implementing a client component of the MSS on the calendar system itself. The client component reads data from the calendar and sends the available times to the central system. The difference between the client component of the personal and multi-user calendar systems is that multi-user system's client component must additionally perform some user management. Integrating web services in this fashion is not possible in practice because a client component cannot be installed to the service without cooperating with the service provider. An alternative solution is to have a proxy server between the MSS and the web service. The proxy functions otherwise like a client component but uses the API provided by the web service to read calendar data instead of accessing a calendar directly. These three use cases are depicted in Figure 4.5.

As we explained in the functional view in Section 4.2, the interfaces on the central system are provided by the *Meeting scheduling component* and it provides three different interfaces: the *Data editing interface*, the *Time period*

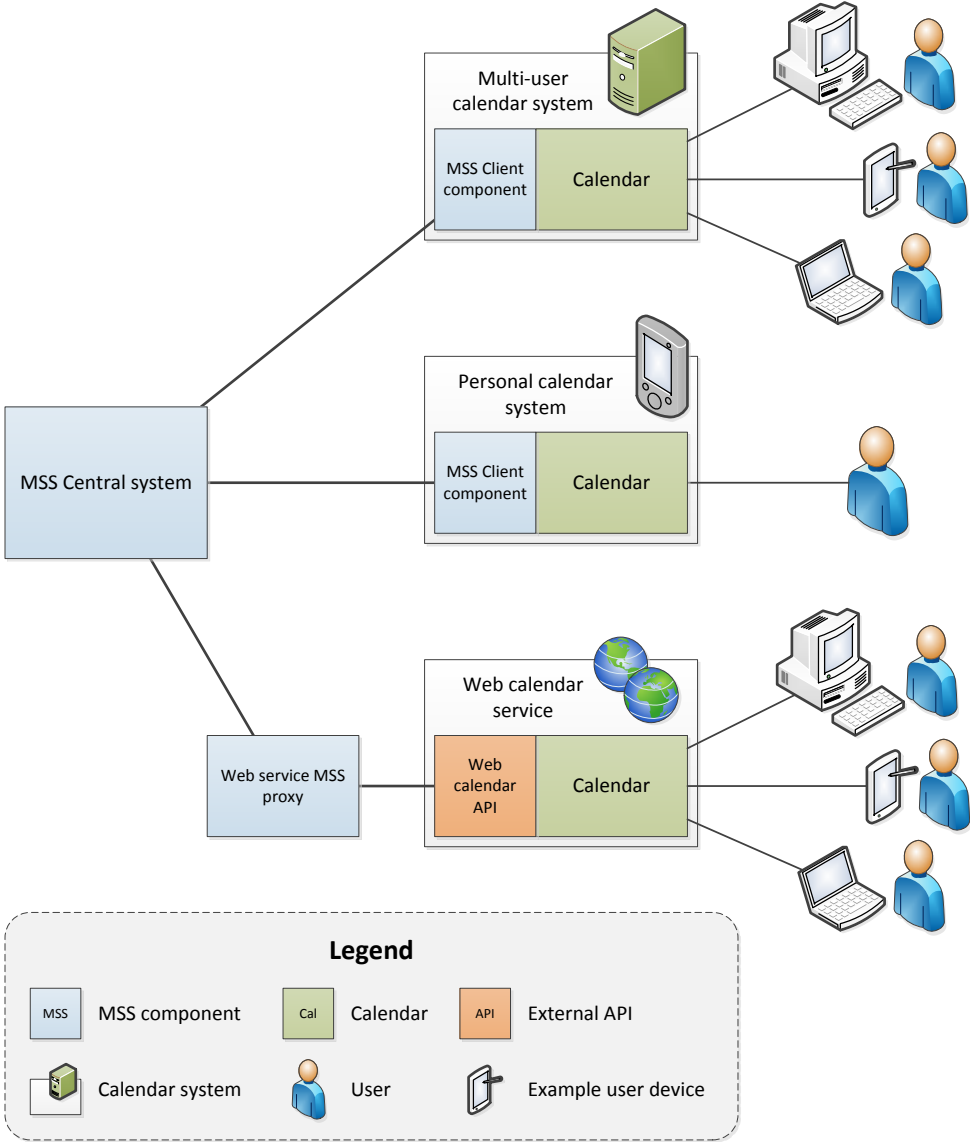


Figure 4.5: Depiction of how three different types of calendar systems can be integrated into the MSS. The symbols for the different devices indicate the heterogeneous nature of the various calendar systems that can be directly integrated into the MSS or can be otherwise involved with the MSS.

interface and the *Request response interface*. The *Data editing interface* is intended for the *User management UI*, the *Group management UI* and the *Organizer UI* so that they can access the information that they present to the users and send the changes or new data entered by the users. The *Time period interface* allows the *User calendar manager* to commit the time periods that it has gathered from the users' calendar upon receiving a meeting proposal. The *Request response interface* is intended for the *Notification agent* to submit the responses to the meeting request it has received from the invited users. In addition to these, the *User calendar manager* provides a *Calendar action interface* to which the *Meeting scheduler* can connect to to send meeting proposals and meeting events to the calendar systems. Also, the *Meeting scheduler* connects to the *Message delivery interface* provided by the *Notification agent* to send meeting requests to invited users.

The operation of the MSS during a scheduling process requires quite a specific interaction sequence between the central system and the other components through the different interfaces. Figure 4.6 shows a sequence diagram of this interaction. The figure is the same as the scheduling process sequence diagram in Figure 4.4 except that it shows only the messages that go through the interfaces between the *Meeting scheduler* element and other components.

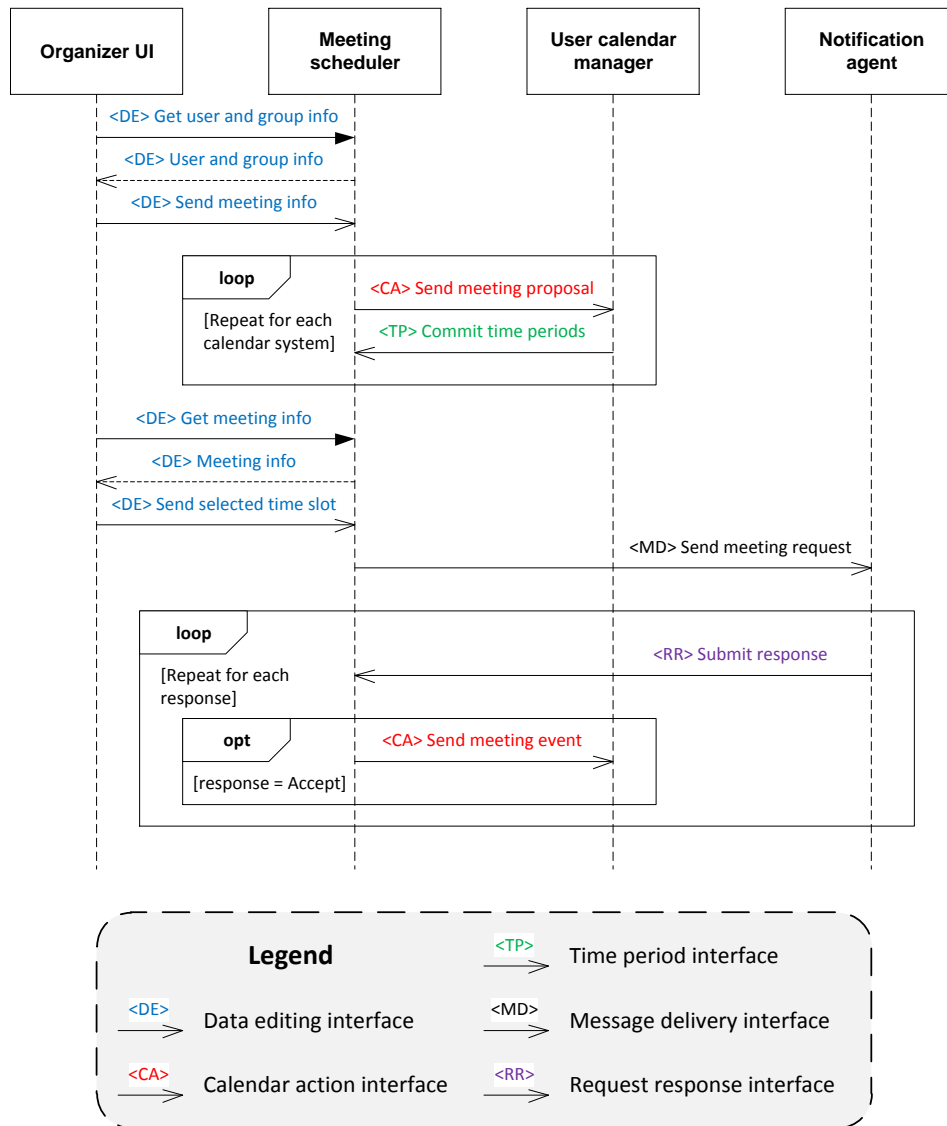


Figure 4.6: A sequence diagram showing the order in which the interfaces of the Meeting scheduler component are utilized during a meeting scheduling process. The different colors of the texts in the messages denote which interface the message goes through.

Chapter 5

Prototype Tool Implementation

To demonstrate the validity of the MSS architecture, we have developed a prototype implementation of the MSS. In this chapter we describe how the prototype functions and how we have implemented it.

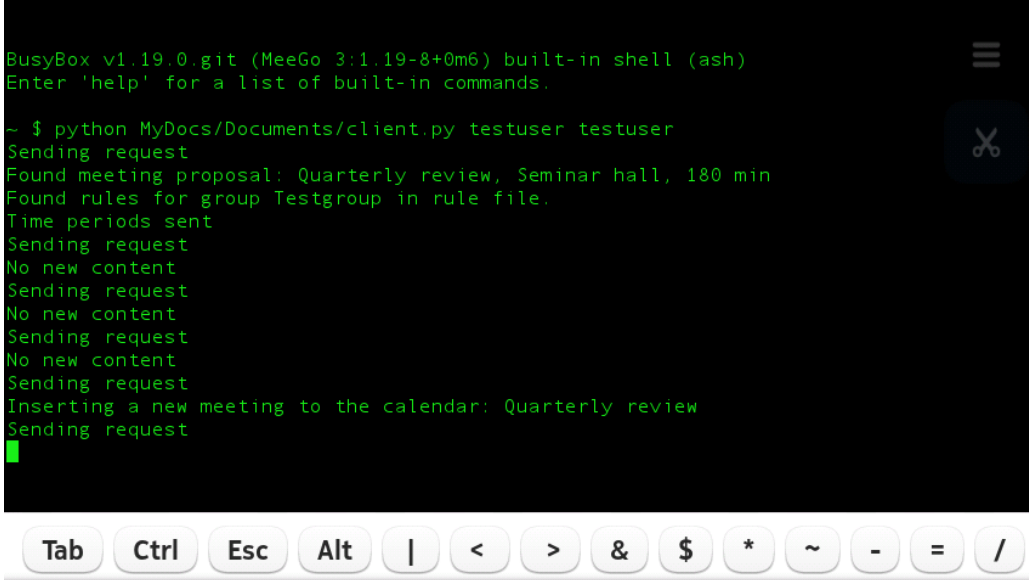
5.1 Prototype functionality

The prototype contains most of the features of the MSS architecture design. It is possible to perform a complete scheduling process with the prototype. This includes allowing users to create new meetings, collecting suitable time periods and presenting them to the organizer, sending meeting requests, allowing participants to respond to the request and adding the meeting to the participants' calendars. The requirements that we did not implement as features in the prototype are listed in Section 6.1.3.

The prototype consists of a web server and a script that is installed on the users' calendar systems, currently available only for the Nokia N950 mobile phone. The web server allows users to create and manage meetings as well as administrators to access the data in the system. The client script reads the user's calendar to find suitable meeting times and transmits the information to the web server.

To have a N950 device participate in the MSS, the client script must be running constantly on the device. This is required so that it can immediately react to the meeting proposals and meeting events coming from the central system. The script can be started through a terminal that is available when a developer mode has been activated on the N950. The username and password of the user who owns the device must be provided as parameters to the script so that the user who owns the device can be identified. The script requires no input from the user but it does print output about its actions into the

terminal window. A screenshot of the script running in the N950 terminal is shown in Figure 5.1. The script runs until it is manually terminated by the user.



```
BusyBox v1.19.0.git (MeeGo 3:1.19-8+0m6) built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ $ python MyDocs/Documents/client.py testuser testuser
Sending request
Found meeting proposal: Quarterly review, Seminar hall, 180 min
Found rules for group Testgroup in rule file.
Time periods sent
Sending request
No new content
Sending request
No new content
Sending request
No new content
Sending request
No new content
Sending request
Inserting a new meeting to the calendar: Quarterly review
Sending request
```

Figure 5.1: A screenshot of the output produced by the client script running on the N950.

New meetings can be created through an organizer website on the web server. The website allows any user who logs in to the site to create a new meeting by first entering the required information into a web form. Immediately after the user has submitted the form, the MSS sends a meeting proposal to the calendar systems of the invited users. The MSS waits until all of the calendar systems have committed their time periods or a maximum of ten seconds (the time limit is adjustable) before it calculates the common time periods based on the time periods it has received. After the common time periods are calculated, the organizer website redirects the user to a web page where she is able to see the common time periods and select a time slot for the meeting. If some calendar systems did not commit their user's time periods in time, the user is notified that information from some invited users is missing. Figure 5.2 shows a screenshot of the meeting creation web form.

Once the time slot selection has been confirmed, the MSS automatically sends a meeting request to the invited users via e-mail. The message displays all of the information about the meeting so that the users can decide if they want to participate in that meeting or not. The message also contains links

Misse meeting scheduling system

Logged in as testuser. [Log out](#)

Create a meeting

Topic:
Topic or name of the meeting.

Location:
Where the meeting is held.

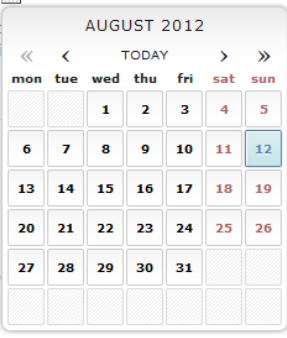
Duration in minutes:
Length of the meeting in minutes.

Date range start:
The start of the range of dates between which possible time slots will be searched and suggested by the system. The given date is included in the range.

Date range end:
The end of the range of dates between which possible time slots will be searched and suggested by the system. The given date is included in the range.

Group:
The group of users that participate in the meeting. Apply the correct availability rules during scheduling.

Invitees:
List of users who are invited to the meeting. Press "Command" on a Mac, to select more than one.



AUGUST 2012						
<<		TODAY		>>		
mon	tue	wed	thu	fri	sat	sun
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Figure 5.2: A screenshot of the meeting creation form on the organizer website of the MSS prototype.

The screenshot shows a web interface for a meeting scheduling system. At the top, there is a header with the title "Misse meeting scheduling system" and a user status "Logged in as testuser. [Log out](#)". Below the header, the section "Meeting details" is displayed. The details include: Topic: Architecture design session, Location: Room 11, Date and time: Tuesday 7.8.2012 - 10:00, Duration: 160, and Group: Testgroup. Underneath, the "Invitation responses:" section shows a table with four rows. The first two rows, for "testuser" and "testuser3", have green backgrounds and are labeled "Accepted invitation". The third row, for "testuser2", has a red background and is labeled "Declined invitation". The fourth row, for "testuser4", has a grey background and is labeled "Not yet responded".

User	Response
testuser	Accepted invitation
testuser3	Accepted invitation
testuser2	Declined invitation
testuser4	Not yet responded

Figure 5.3: A screenshot of the organizer website of the MSS prototype showing the details of a meeting that already has a selected time slot. The colored bars in the picture describe which invited users have responded to the meeting request and what are their responses.

that the users can use to respond to the meeting request. The links point to specific pages on the organizer site which each correspond to a different type of response: accept, decline or request rescheduling. The users are identified using a user- and meeting specific identifier strings included in the link so that the users do not have to log in manually. Our decision to use e-mail to send the requests, of course, requires that all the users of the system must have a valid e-mail address included in their information.

Our organizer website also allows the users to see a list of all the meetings that they are the organizer of and details about each meeting including how each invited user has responded to the meeting request. Figure 5.3 shows a screenshot of the meeting details page on the organizer website.

If an invited user sends an accepting response to the meeting request, the meeting event is automatically added to the user's calendar.

The prototype also has a web site that is intended for the administrators of the MSS. The site allows users of the MSS, who have been given the appropriate rights, to see and edit all data in the database. For example, an administrator can add or remove users, add or remove user groups, remove meetings or remove individual request responses. A screenshot of the administrator site is shown in Figure 5.4.

Misse meeting scheduler administration		Welcome, misse. Change password / Log out
Home > Meeting_scheduler > Meetings > Request: 2012-05-22 12:00, Discussions about cake, The lounge, 120 min		
Change meeting		History
Topic:	<input type="text" value="Discussions about cake"/>	Topic or name of the meeting.
Location:	<input type="text" value="The lounge"/>	Where the meeting is held.
Duration in minutes:	<input type="text" value="120"/>	Length of the meeting in minutes.
Date range start:	<input type="text" value="2012-05-14"/> Today	The start of the range of dates between which possible time slots will be searched and suggested by the system. The given date is included in the range.
Date range end:	<input type="text" value="2012-05-27"/> Today	The end of the range of dates between which possible time slots will be searched and suggested by the system. The given date is included in the range.
Organizer:	<input type="text" value="testuser3"/>	The user who created the meeting and is allowed to change it and select the time slot.
Group:	<input type="text" value="Testgroup"/>	The group of users that participate in the meeting. Used only to apply the correct availability rules during scheduling.
Common time periods:	<input type="text" value="2012-05-15 11:00,2012-05-15 12:59;2012-05-16 12:00,2012-05-16 13:59;2012-05-17 10:00,2012-05-17 11:59;2012-05-17 13:00,2012-05-17 16:59;2012-05-21 12:30,2012-05-21 14:59;2012-05-22 08:00,2012-05-22 09:59;2012-05-22 12:00,2012-05-22 13:59;2012-05-22 15:00,2012-05-22 16:59;2012-05-23 10:00,2012-05-23 14:59;2012-05-24 08:00,2012-05-24 09:59;2012-05-24 14:00,2012-05-24 16:59"/>	

Figure 5.4: A screenshot of the administrator site of the MSS prototype. The picture shows a meeting data object being edited.

5.2 Technology choices

The MSS prototype was almost entirely programmed with the Python programming language. Both the central system and the calendar system client component were done with Python utilizing some existing Python libraries and frameworks. Because our central system is a web server, all of the UIs were done with Hypertext Markup Language (HTML) and JavaScript.

We utilized the following libraries and frameworks in the prototype implementation:

- **Django, a web development framework for Python** (Django Software Foundation, 2012). Django enables the creation of web applications using Python. The entire central system of the MSS is built using the Django framework. Django is released under BSD license.
- **Tastypie, a web API framework for Django** (Lindsley et al., 2012). Tastypie simplifies the creation of Representation State Transfer (REST) style interfaces for Django applications. We used it to implement the *Data editing*, *Time period* and *Request response interfaces* on the central system. Tastypie is released under BSD license.
- **Requests, a Hypertext Transfer Protocol (HTTP) library for Python** (Reitz, 2012). Requests greatly simplifies the making of HTTP (Berners-Lee et al., 1996) requests in Python. We used it on the client component to implement the *User calendar manger's* communication with the central system. Requests is released under ISC license.
- **Unobtrusive Date-Picker Widget V5, a graphical JavaScript date selector** (McAllister, 2012). This date picker allows the selection of dates from a graphical calendar on a web page. We used it in the *Organizer UI* to simplify the selection of a date range for a meeting. Unobtrusive Date-Picker Widget V5 is released under Creative Commons Attribution-ShareAlike license.

We developed a client component for the Nokia N950 mobile phone to integrate it into the MSS. The N950 is a developer-only mobile device and as such it is not available commercially. However, it is almost identical to the Nokia N9 mobile phone which is available to the public. The biggest difference between these two devices is that the N950 has a physical keyboard while the N9 does not. The N950 runs the Linux-based Meego 1.2 Harmattan operating system. Python environment is installable on the N950.

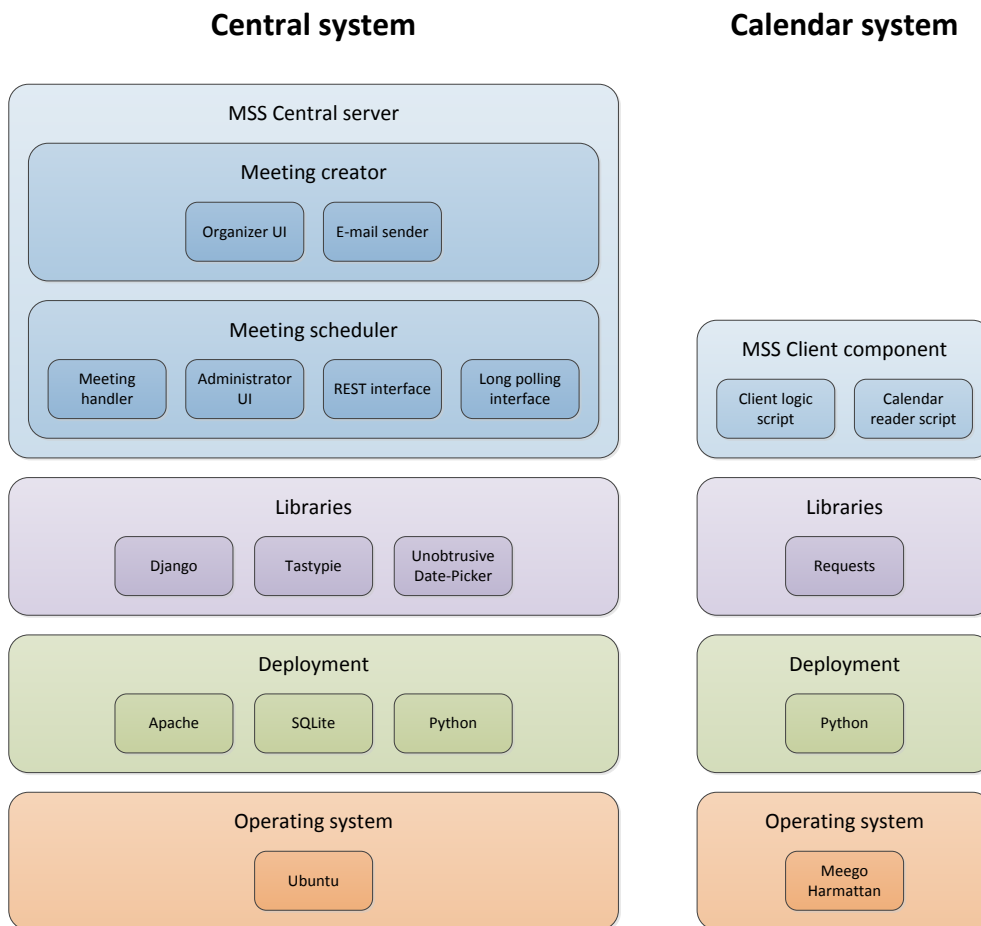


Figure 5.5: The solution stack of the MSS prototype showing what software is used in the central system and the calendar system. The outer rectangles are software categories or logical modules. Inner rectangles are the actual software components. Software that is higher up in the picture depends on some or all of the software below it.

The web server that acts as the central system of the MSS runs on Apache HTTP Server and Ubuntu Linux operating system. The server uses an SQLite database.

The software that is used in the prototype is visualized as a solution stack in Figure 5.5. It shows the platforms that the prototype runs on, the libraries we have used in the implementation as well as the modular structure of the prototype itself. The top-most blue rectangles represent the modules of the prototype.

5.3 Implementation of the functional elements

In this section we describe the details of how we have implemented the functional elements of the architecture in the prototype and how they function. Figure 5.6 shows which of the functional elements have been implemented in the prototype and how the central system interfaces have been implemented. This figure matches the functional elements and interfaces presented in Figure 4.2 in the functional view of the architecture.

5.3.1 Central system

As mentioned, the central system is a web server created with Python using the Django framework. The central system includes the functional elements *Meeting scheduler*, *Scheduler database* and *Administrator UI*. Also, the *Organizer UI* functional element has been implemented on the central system in the prototype. An optional solution would have been to implement the *Organizer UI* on the calendar system but we chose to do it on the central system because constructing a separate user interface on the N950 would have been much more difficult and time-consuming for us than creating a web-based UI on the server with Django.

Django projects consist of one or more applications or 'apps' which are independent modules with their own functionality. The central server of the MSS prototype consists of two Django apps. The first one is *Meeting scheduler* which implements the central system itself - that is the *Meeting scheduler*, the *Scheduler database* and the *Administrator UI* functional elements (Note: While the Meeting scheduler Django app and the Meeting scheduler functional element have the same names, they do not represent entirely the same thing. The Django app implements the functional element but also some other elements as well). The second one is *Meeting creator* which implements the *Organizer UI* functional element. The *Meeting scheduler* app is self-sustained and could be copied to other Django projects if

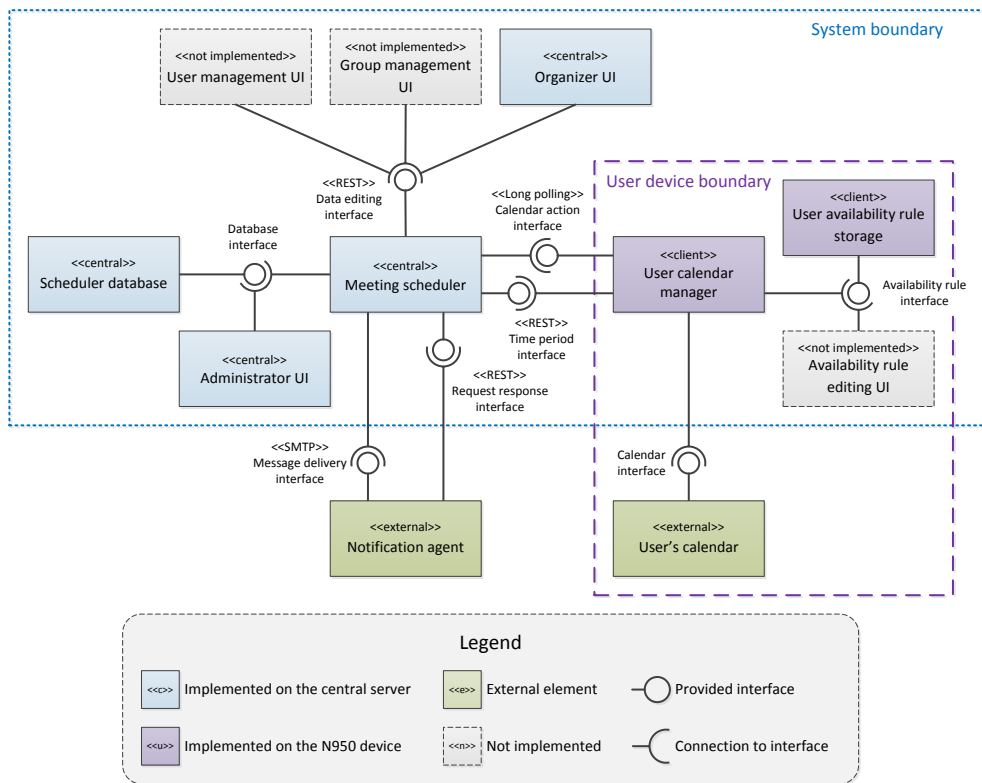


Figure 5.6: An adapted version of the functional view depicting which functional elements of the architecture have been implemented in the prototype. Also shows how the different interfaces of the Meeting scheduler element have been implemented.

desired but the *Meeting creator* app relies on the *Meeting scheduler* app to function. These two apps can be seen in the solution stack of the central system in figure 5.5.

The Django framework is based on the Model-View-Controller (MVC) UI paradigm where the concerns of data, its presentation to the users and the user input are separated from each other (Burbeck, 1987). The *model* consists of the data and the business rules of the system (Reenskaug, 1979). A *view* is one possible visual representation of some data in the model (Reenskaug, 1979). A *controller* handles the receiving of input from the user and translates the input into appropriate commands for the model or the view (Reenskaug, 1979). Django, however, uses its own set of terms that describe the distribution of concerns within the system. These terms are Model-Template-View but, apart from the term *model*, they do not directly correspond to those of the MVC. In Django, the *templates* are text documents, for example HTML documents, that act as the visual representation of the data in the model while the *views* are functions that prepare the data that is presented in the views. Together the templates and the views approximately correspond to what the views are in the traditional MVC paradigm. The *controller* in Django is the URL configuration that matches the HTTP requests coming from the user into the appropriate views. Each Django application can have its own model, views and templates.

In the prototype, the *Meeting scheduler* app has no templates since it does not directly interface with the users. The app also has only one view which is related to our implementation of the *Calendar action interface*. The implementation of the interfaces is explained in Section 5.4. However, the model of the *Meeting scheduler* app is important as it represents the **Scheduler database** functional element in its entirety. Django provides Object-Relational Mapping (ORM) features that allow the definition and usage of a data model directly with Python while the actual transactions between the database and the program code are performed automatically. We used SQLite as the database implementation, which is the default provided by Django. The database could easily be replaced with another implementation, such as MySQL (Oracle Corporation, 2012) or PostgreSQL (PostgreSQL Global Development Group, 2012). A database diagram of the *Scheduler database* is displayed in Figure 5.7. Table 5.1 describes what type of information each of the database tables contains.

The **Meeting scheduler** functional element is implemented in the prototype mostly as a set of functions in the central system app that are called when some data in the database changes. For example, when a new meeting is added to the database by the *Organizer UI* (in the prototype the *Organizer UI* is able to access the *Scheduler database* directly), a function is called

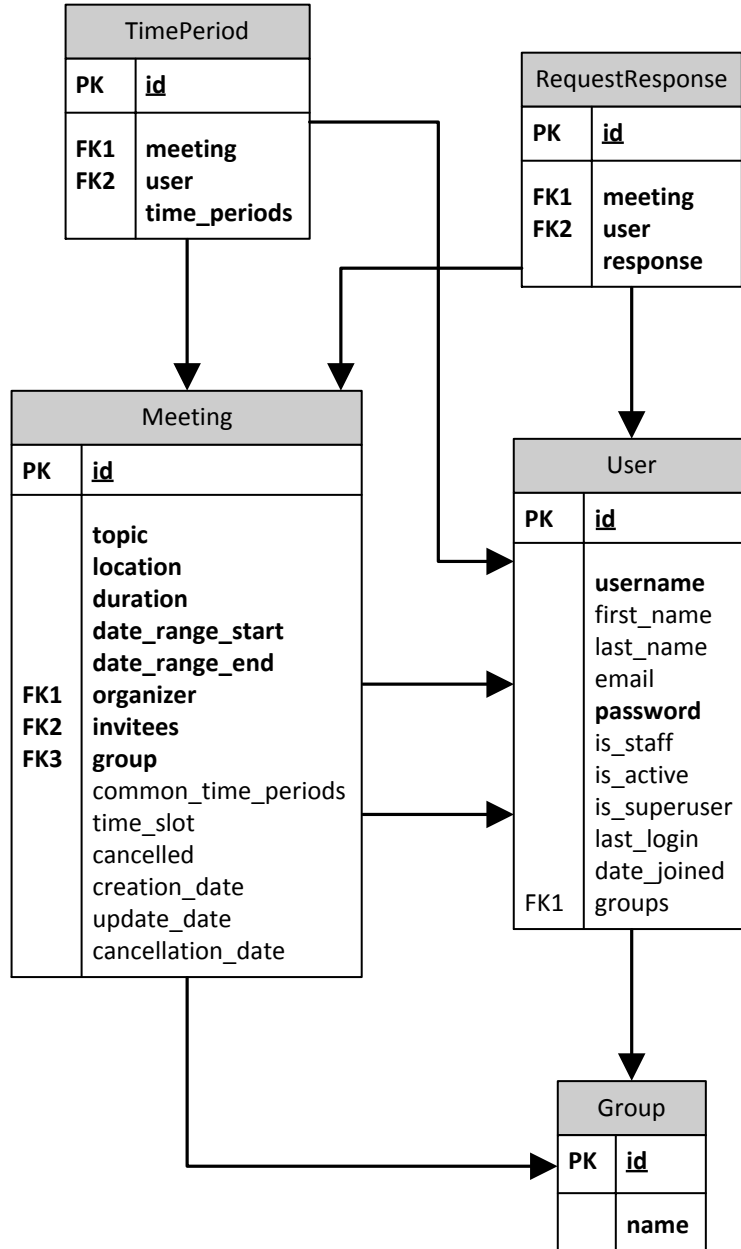


Figure 5.7: A database diagram of the *Scheduler database*. Shows the different tables in the database, their column names and the relations between them.

Table 5.1: Description of the information stored in the tables of the Scheduler database.

Database table	Content description
User	Data about the users.
Group	Data about the user groups.
TimePeriod	Meeting specific time periods from different users.
RequestResponse	Meeting specific responses to the meeting requests from invited users.
Meeting	Data about the meeting proposals, requests and events. Each meeting also contains its common time periods.

which sends meeting proposals to the calendar systems, awaits the time periods to be committed and then calculates the common time periods. Also, the interfaces provided by the *Meeting scheduler* element are implemented in the *Meeting scheduler* app. These interfaces are described in more detail in Section 5.4.

One task of the *Meeting scheduler* element is the calculation of common time periods for a meeting. In the prototype, this happens with a simple algorithm that the *Meeting scheduler* app performs on the invited users' time periods. The time periods are received from the calendar systems and saved to the database as a specifically formatted string. The string contains a series of time periods separated by semicolons (;) and each time period consists of a starting and ending time pair separated by a comma (,). The starting and ending times are given with an accuracy of one minute. A time period must have a length of over one minute but there is no upper limit for the length. An example of a time period string is given below:

```
2012-06-18 08:00,2012-06-18 09:59;2012-06-18 12:00,2012-06-18 16:59;
2012-06-19 08:00,2012-06-19 16:59;2012-06-20 08:00,2012-06-20 08:59;
2012-06-20 09:45,2012-06-20 12:59;2012-06-20 15:00,2012-06-20 16:59;
2012-06-21 08:00,2012-06-21 10:59;2012-06-21 15:30,2012-06-21 16:59
```

The algorithm first parses these time period strings into Python datetime objects so that they can be compared to each other easily. It then goes through all of the starting and ending times to find those between which all of the invited users are simultaneously available. Each of the starting and ending time pairs found this way forms a single common time period and is added to the list of common time periods. After all the common time periods are found, they are saved to the database as an attribute of the meeting they belong to, formatted similarly as the individual time periods.

Another task of the *Meeting scheduler* element is the creation of meeting proposals and meeting requests. The creation of meeting proposals is performed very simply in the prototype. The *Meeting scheduler* app creates a new meeting object based on the information received from the *Organizer UI* after which the Django ORM adds it also to the database. Meeting requests are also created simply by just adding the time slot information to the existing meeting proposals. The *Meeting scheduler* sends the meeting requests to the calendar systems as soon as they are created by sending a *proposal* type message through the long polling interface. The long polling interface and the different types of messages are described in Section 5.4.3.

One task of the *Meeting scheduler* is the gathering of responses to meeting requests. This is done by allowing users to submit the responses through the REST interface. When a response is received, its content is checked to see what the response is. If the response is "accept", the meeting event information is sent in a *sync* type message through the long polling interface. The responses are saved to the database so that it can be later known which users have accepted the meeting request in case there are any changes that need to be updated. If changes occur, a long polling message also of the type *sync* is sent to all participants who accepted the request for that meeting.

The **Administrator UI** functional element is implemented as an administrator website that Django creates automatically for all projects.

As said, the **Organizer UI** functional element is implemented as a web site in a separate Django app, the *Meeting creator*. The templates and views of this app compose the different pages of the site, such as the meeting creation form and the meeting detail page. The *Meeting creator* app, however, doesn't have a data model that would be significant to the scheduling process. The only data stored in the *Meeting creator* model are randomly generated identifier strings that are used when the users respond to meeting requests to make the responding more convenient.

As we explain in section 6.1.3, requirements R56 and R59 are not implemented in the prototype. This means that it is not possible for the organizer to change or cancel her meetings or send reminders to invited users through the organizer website. If those feature were implemented, however, the organizer website would be the most rational place to allow the organizer to do them.

In the MSS prototype, we use an e-mail server as the **Notification agent** functional element. We decided to use e-mail since most of our potential users likely already use it actively and many of the existing scheduling systems also utilize e-mail to notify their users about meetings. The prototype's communication with the e-mail server is implemented in the *Meeting creator* app.

The **User management UI** and the **Group management UI** functional elements are not implemented at all in our prototype. Instead, the editing of user and group information can be done using the administrator site. It is also possible to edit the data manually through our implementation of the *Data editing interface*. This interface is explained in Section 5.4.2.

5.3.2 Calendar system

The client component that runs on the N950 consists of two Python script files: *Calendar reader* and *Client logic*. Together the scripts implement the functionality of the **User calendar manager** functional element. The *Client logic* script file contains about 390 line of code including comments and blank lines and the *Calendar reader* script file contains about 240 lines of code. This indicates that both scripts are fairly simple and we estimate that developing a similar client component for other calendar systems should not be any more complicated.

The *Calendar reader* script handles only the reading and writing of data to the *User's calendar*. The *Client logic* script contains all of the actual logic of the *User calendar manager*. The *Client logic* script continuously monitors for incoming meeting proposals and meeting events from the central system and reacts accordingly when they are received. If a meeting proposal is received, the *Client logic* script asks the *Calendar reader* script to read the free time periods from the calendar. The *Client logic* script combines the free times with the availability rules and sends them to the central system. If a meeting event is received, the *Client logic* script asks the *Calendar reader* script to enter a matching calendar entry to the user's calendar.

The **User's calendar** is the default, built-in calendar application on the N950. The calendar data is stored in an SQLite database on the device. The database contains the individual events that appear in the device's calendar and also some other data that is not related to the calendar. The *Calendar reader* script reads and writes calendar events to the calendar database by executing SQL queries.

We implemented the **Availability rule storage** as a simple text file on the device. The file specifies the periods of time that the user wants to make available for meetings with each group for each week day. The rules of different groups are identified by the group name. The file also contains default availability rules that are used if no applicable available rules are present for the group that the meeting is intended for. Additionally, since it is possible to have multiple calendars on the N950, the file specifies by name which of those calendars are used in finding the available times for the meeting. The availability rule file is formatted in a specific way that allows

the *Client logic* script to read it automatically but also so that it can be easily understood by a human reader. An example of the file format is given below:

```
Calendars = Personal, Private

#default

Monday = 08:00-17:00
Tuesday = 08:00-17:00
Wednesday = 08:00-17:00
Thursday = 08:00-17:00
Friday = 08:00-15:00

#Testgroup

Monday = 10:00-14:00
Wednesday = 08:00-12:00
Thursday = 12:30-16:00
Friday = 09:00-11:00
```

We did not implement any specific **Availability rule editing UI** in the MSS prototype. Instead, the users can use any text editor to edit the availability rule file manually.

5.4 Design and implementation of the interfaces

In the architecture design, the MSS has five interfaces between the central system and the other components: *Data editing interface*, *Time period interface*, *Request response interface*, *Calendar action interface* and *Message delivery interface*. These interfaces and the technologies that they have been implemented with can be seen in figure 5.6. In this section, we describe how each of these interfaces is implemented in the prototype.

5.4.1 E-mail interface

As mentioned in Section 5.3.1, we use e-mail to send the meeting requests. This means that the *Message delivery interface* is implemented by sending the messages over Simple Mail Transfer Protocol (SMTP) (Postel, 1982) from the organizer website app to the e-mail server.

Table 5.2: A list of HTTP request methods that the REST interface accepts and the actions they result in.

Method	Target	Action
GET	Single object	Returns the object information.
GET	Object list	Returns the information of all objects in the resource.
POST	Single object	N/A
POST	Object list	Creates a new object.
PUT	Single object	Updates all information fields of the object.
PUT	Object list	N/A
PATCH	Single object	Updates only the given information fields of the object.
PATCH	Object list	N/A
DELETE	Single object	Removes the object from the database.
DELETE	Object list	N/A

5.4.2 REST interface

We have combined *Data editing interface*, *Time period interface* and *Request response interface* into one single interface in the prototype. This is done with a REST interface on the central web server that allows users to see and edit data in the central database using ordinary HTTP requests.

Each of the tables in the database is presented in the REST interface as a resource. Different resources and objects in each resource have unique Universal Resource Locators (URLs) which are used to identify them. For example, the URL `<server address>/rest/v1/meeting/` belongs to the Meeting resource and presents a list of all the meeting objects in the resource. A URL for a single meeting is `<server address>/rest/v1/meeting/11/` where the number 11 is an identifier for the meeting. To access or edit the resources, users must make an HTTP request to the URL of the resource. The method of the request determines what action is taken upon the resource. Table 5.2 lists the methods that are accepted in the prototype and their corresponding actions. Not all of the methods are usable on every resource.

Requests that have the method POST, PUT or PATCH must include the data that is sent to the server within them. All the data that is received or sent through the REST interface is serialized in JavaScript Object Notation (JSON) (Crockford, 2006) format and the request must have the 'Content-Type' header set to 'application/json'. Support for other formats could be implemented as well if necessary.

All of the requests to the REST interface must identify the user who makes that request. This is done with HTTP basic access authentication by providing the username and password of the user in the request. The authorization to see and edit data through the interface is restricted based on the user identity. Users are only able to see and edit the data that is relevant to them. For example, user can see the user data of all users but edit only the data about themselves. Also, the details about a meeting can only be seen by the users who are invited to that meeting and edited only by the organizer.

An example of an HTTP request sent to the interface and the returned response is displayed below. The top two lines of the example are the request made by the user and the following lines are the response from the server. The example was made using a command-line network application called cURL (Stenberg, 2012) on Windows (Microsoft Corporation, 2012b) operating system. The request is authenticated by a user with the username 'testuser' and also the password 'testuser'. The purpose of the request is to get the details about testuser herself.

```
C:\cURL>curl.exe --dump-header - -u testuser:testuser
http://kurre.soberit.hut.fi:8000/rest/v1/user/2/

HTTP/1.1 200 OK
Date: Fri, 20 Jul 2012 13:39:41 GMT
Server: Apache/2.2.20 (Ubuntu)
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8

{
  "email": "example@email.net",
  "first_name": "Test",
  "groups": ["/rest/v1/group/1/", "/rest/v1/group/2/"],
  "id": "2",
  "last_name": "User",
  "resource_uri": "/rest/v1/user/2/",
  "username": "testuser"
}
```

The first set of rows in the response represents the headers of the response. The actual data of the response is within the curly brackets.

An example of a request that sends data to the server is displayed below. This request creates a new meeting into the MSS. The user who sends the request is automatically made the organizer of the meeting. References to other objects, such as groups and users, are given as the URLs of those objects. URLs of each object are included in their data.

```
C:\cURL>curl.exe --dump-header - -u testuser:testuser
-X POST -H "Content-Type: application/json"
--data "{
  \"topic\": \"Quarterly review\",
  \"location\": \"Seminar hall\",
  \"duration\": 180,
  \"date_range_start\": \"2012-07-06\",
  \"date_range_end\": \"2012-07-12\",
  \"group\": \"/rest/v1/group/1\",
  \"invitees\": [\"/rest/v1/user/2\", \"/rest/v1/user/3\"]
}"
http://kurre.soberit.hut.fi:8000/rest/v1/meeting/

HTTP/1.1 201 CREATED
Date: Mon, 14 May 2012 12:07:11 GMT
Server: Apache/2.2.20 (Ubuntu)
Location: http://kurre.soberit.hut.fi:8000/rest/v1/meeting/2/
Vary: Accept-Encoding
Content-Length: 0
Content-Type: text/html; charset=utf-8
```

Since the request was made to create send new data and not to request existing data, the response contains only headers. Of particular interest is the 'Location' header which specifies the URL of the newly created meeting.

Details about the REST interface resources including their URLs, data fields and accepted HTTP methods can be seen in Appendix A.

5.4.3 Long polling interface

We have implemented the *Calendar action interface* using the 'long polling' technique. Long polling allows the emulation of pushing information from server to clients using standard HTTP requests. In long polling, the client sends a request to the server but, if there is no new content on the server, the server does not immediately send back a response. Instead, the server waits until new content becomes available (or until a predetermined timeout). When new content becomes available, the content is sent to the client as a response and the client immediately sends a new request to the server. In this way, there is a constant connection between the server and the client. The connection is also real-time since the server always sends the content to the client as soon as it is made available.

Because of this implementation decision, the central system of the prototype provides the interface to which the calendar systems connect to. This is contrary to our architecture design where the client component provides the interface.

A connection to the interface can be made by sending a HTTP POST request to the URL `<server address>/longpoll/`. The username and the password of a user must be provided as POST parameters with the request. Also, a parameter that specifies the date and time of the previous request made by this client should be provided. If a new meeting proposal or meeting event is available on the server or after a set time limit has been reached, the server sends back a response that contains the necessary information as a JSON formatted string.

The response string contains two attributes: 'type' and 'payload'. Type describes the purpose of the response and the payload contains the actual data of the response. Table 5.3 lists the different possible types of the responses, their description and content of the payload as well as what the client component should do after receiving this type of response. Note that, regardless of the response, the client component should always send a new request to the server immediately.

Table 5.3: The different types of responses sent by the central server via the long polling interface.

Type	Description	Payload	Expected action
empty	There is no new content on the server and the response was sent because the fail-safe time limit was reached.	String 'No new content'.	Nothing.
proposal	A new meeting has been created where the user is invited.	Details about the meeting.	The client component should gather the available time periods from the user's calendar and commit them to the scheduler using the REST interface.
sync	The user has accepted a request for a meeting or an existing meeting has been updated.	Details about all the updated meetings.	The client component should update the events in the user's calendar.
error	There was a problem with the long polling request, such as a missing parameter or authentication failure.	String describing the error.	The client component should send a valid request next time.

An example of a response to a request sent to the long polling interface using cURL is shown below. The response type is sync, meaning that a new meeting event has become available or has been updated. The payload is a Python dictionary containing the information of one meeting that has been updated since the `last_checked` parameter of the request.

```
C:\cURL>curl.exe --dump-header - -X POST
--data "username=testuser&password=testuser
&last_checked=2012-07-20T10:40:00"
http://kurre.soberit.hut.fi:8000/longpoll/

HTTP/1.1 200 OK
Date: Mon, 23 Jul 2012 09:38:21 GMT
Server: Apache/2.2.20 (Ubuntu)
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8

{"type": "sync", "payload": [{"
  "update_date": "2012-07-20T10:41:52.908200",
  "response_uri": "/rest/v1/meeting/7/response/",
  "response_count": 1,
  "group": {
    "users": ["/rest/v1/user/3/", "/rest/v1/user/2/"],
    "resource_uri": "/rest/v1/group/1/",
    "id": "1",
    "name": "Testgroup"
  },
  "time_period_uri": "/rest/v1/meeting/7/timeperiod/",
  "duration":180,
  "date_range_start": "2012-08-06",
  "time_slot": "2012-08-07T10:00:00",
  "date_range_end": "2012-08-12",
  "creation_date": "2012-07-20T10:41:36.440774",
  "topic": "Quarterly review",
  "cancellation_date": null,
  "id": "7",
  "location": "Seminar hall",
  "cancelled": false,
  "committed_time_period_count": 1,
  "organizer": "/rest/v1/user/2/",
  "resource_uri": "/rest/v1/meeting/7/",
  "common_time_periods": "2012-08-06 08:00,2012-08-06
    16:59;2012-08-07 08:00,2012-08-07 16:59;2012-08-08
    13:00,2012-08-08 16:59;2012-08-09 08:00,2012-08-09
    16:59;2012-08-10 08:00,2012-08-10 14:59",
  "invitees": ["/rest/v1/user/2/"]
}]}
```

5.4.4 Interface usage sequence

The REST interface and the long polling interface allow any third party system to connect to the central system of the prototype as designed in the architecture. The sequence diagram in Figure 5.8 shows examples of the messages that the *Organizer UI* and the *User calendar manager* functional elements would have to send to the central system if they were implemented in a third-party system. *Notification agent* is also included in the sequence for clarity. The notification agent could also be implemented as part of a third-party meeting management system but the current MSS prototype that we have implemented only supports sending meeting requests via SMTP to the e-mail server.

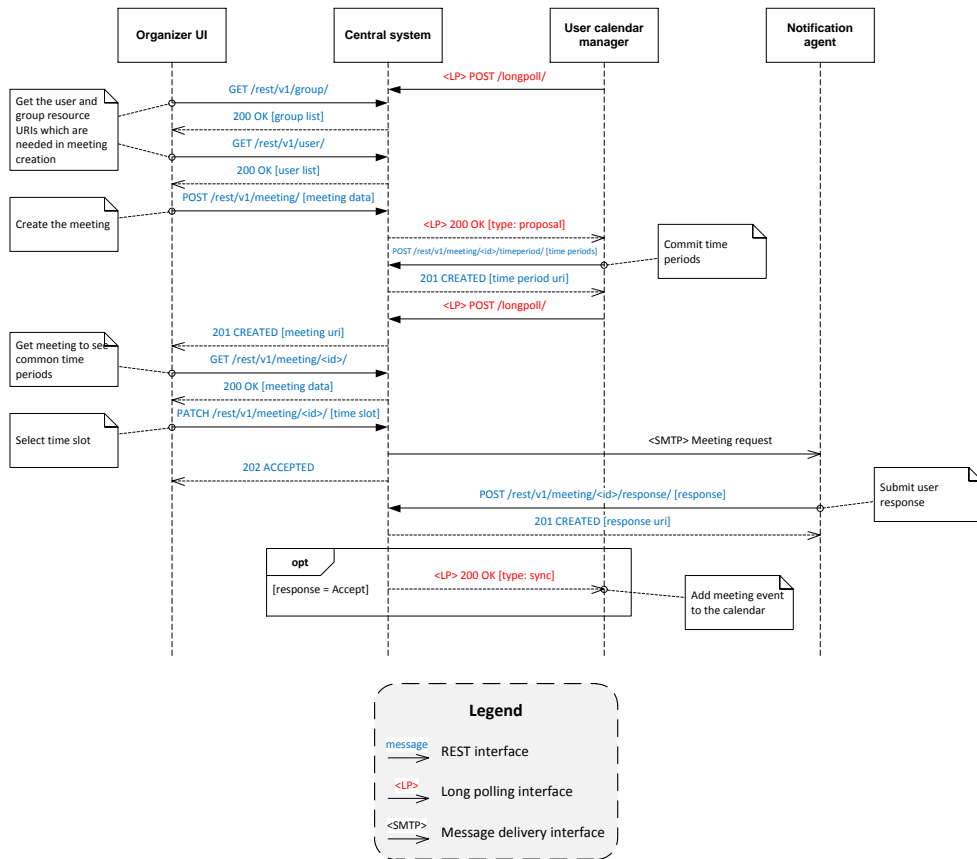


Figure 5.8: A sequence diagram showing the detailed messages sent through the REST and long polling interfaces during a meeting scheduling process in our prototype.

Chapter 6

Evaluation

Evaluation is an important part of good research. The results of the research should be evaluated through experience or systematic analysis to build confidence in their validity (Shaw, 2002). In this chapter we *verify* and *validate* (IEEE, 2012) the architecture and the prototype to ensure that the MSS is designed according to the specification and that it fulfills the user needs. We have used three different methods of evaluation: First, we traced the design decisions that we have made in the prototype and the architecture back to the requirements and research questions to verify that they are consistent. Second, we compared the scheduling process of the MSS to that of some other existing meeting scheduling applications to see if the MSS is more effortless to use than they are. Effortlessness was evaluated because it is one of the key success factors for the MSS. Third, we held an evaluation session with a group of experts at IPSS so that they could assess the feasibility of the architecture and validate that the MSS provides value to the users.

6.1 Traceability from research questions to implementation

In this section we *verify* (IEEE, 2012) the architecture. Verification means ensuring that the system that has been built matches the specification that was set for it beforehand. In other words, we ensure that the system has been built correctly. We do this by tracing the features that we have implemented in the MSS prototype and the architecture design back to the research questions. The research questions are the starting point of the MSS specification.

6.1.1 From research questions to requirements

The goal of the MSS architecture and the prototype tool is to help us answer the research questions of this thesis. In Section 3.1 we formulated five high-level requirements for the MSS in collaboration with IPSS. The requirements were selected on the basis that if the MSS is able to fulfill them it would also provide answers for the research questions. We expanded these high-level requirements into numerous detailed requirements which are listed in Section 3.4. In that section, the requirements are divided into multiple categories and for each category it is explained how those requirements relate to the different research questions. All of the four research questions have been taken into account in the requirements.

6.1.2 From requirements to architecture

The requirements are the basis of the architecture design. All of the requirements that we elicited for the MSS have been taken into account when designing the architecture. That said, not all of the requirements are architecturally significant and, as such, are not explicitly specified in the architecture documentation. However, the MSS architecture is designed so that these requirements can be satisfied in the implementation.

Requirements R11-R14 are about the interfaces that enable calendar systems to communicate with the MSS. The interfaces in the architecture design are visible in the functional view Figure 4.2 and documented in Section 4.5. Requirements R15 and R16 are about protecting the privacy of the users' calendar data. Privacy issues in the architecture are documented in the information view in Section 4.3 and in the *User calendar manager* element's documentation in the functional view in Section 4.2.

Requirements R21-R24 regard the user groups. The groups are specified as user group data element in the information view. The users' possibility to join, leave and customize the groups is enabled by the *Group management UI* functional element in the functional view. Requirements R25-R27 are about the user's possibility to customize their available time based on the user groups. These are satisfied in the availability rules which are documented in the information view. *Availability rule storage* and *Availability rule editing UI* functional elements are also present in the functional view.

The process of creating new meetings before the actual scheduling starts is specified in requirements R31-R36. Most of these requirements are not architecturally significant but the *Organizer UI* functional element makes it possible to take them all into account in the implementation.

The design of the *Organizer UI* functional element also supports require-

ments R41, R45, R46 and R47 which are about selecting a time slot for the meeting. Requirements R42-R44 and R48 are about reading the users calendar to find out the suitable time periods. In the functional view it is specified that the *User calendar manager* handles these responsibilities.

Requirements R51-R53 are about sending and responding to meeting requests. The external *Notification UI* functional element handles these tasks, as is documented in the functional view. Requirements R54-R511 specify how the meetings can be altered and how they change in the user's calendar if they have already been added there. These requirements are not architecturally significant, but features that satisfy them could be implemented in the *Organizer UI* and the *User's calendar manager* functional elements.

6.1.3 From architecture to implementation

The prototype of the MSS contains all of the key features that we identified in Section 3.1. This includes most of the detailed requirements listed in Section 3.4 but, to reduce the amount of effort required to create the prototype, we decided to omit some of the less important detailed requirements. Table 6.1 lists the requirements that have *not* been implemented in the prototype or that have been implemented only partially. All of the rest of the detailed requirements specified in Section 3.4 have been implemented in the prototype.

All of the omitted requirements are about features that increase the ease-of-use or usefulness of the MSS, giving more options to group management and meeting organizing. Some of them are simple nice-to-have features with a low priority while others are of medium priority because they are quite important in providing a good user experience in a fully implemented MSS. However, none of the requirements are architecturally significant and are not part of the key functionality of the MSS which is why we felt that it is acceptable to omit them in the prototype.

Most of the functional elements in the architecture are implemented in the prototype. The elements that are not implemented are *User management UI*, *Group management UI* and *Availability rule editing UI*. These user interface elements were omitted because the data that they edit can also be edited by other means. User and group data can be edited through the *Administrator UI* and the availability rules can be edited directly in the text file. All public interfaces on the central system are implemented in the prototype. Figure 5.6 visualizes what functional elements are implemented in the prototype and how the interfaces are implemented. Finally, all of the information elements in the architecture are utilized in the prototype.

Table 6.2 shows how the different steps of the scheduling process are done in the MSS prototype as compared to the architecture design. The table

Table 6.1: Omitted requirements in the prototype.

ID	Description	Priority
R23	The MSS shall allow group members to set their group to be joinable by everyone or just invited users.	Low
R24	The MSS shall allow group members to set their group to be valid for only a certain period of time.	Low
R47	When the MSS has completed the scheduling process for a meeting and if no common time periods have been found, the MSS presents to the organizer a set of the time periods when the highest amount of invited participants are able to attend.	Medium
R56	When the MSS has notified the invited participants about the proposed meeting, the MSS shall allow the organizer to change the details of the proposed meeting or cancel the proposed meeting at any point.	Medium
R57	When the organizer has canceled the meeting event, the MSS shall remove the meeting event from the calendars of the participants.	Medium
R58	When the organizer has changed details of a meeting event, the MSS shall change the meeting events in participants' calendars accordingly.	Medium
R59	While the MSS has notified the invited participants about the proposed meeting, the MSS shall allow the organizer to send reminders to the invited participants about responding.	Low
R510	When the MSS has created a meeting event, the MSS shall allow each participant to exclude herself from the meeting event.	Low
R511	When a participant has excluded herself from a meeting event, the MSS shall remove that participant from the meeting event's list of participants as well as remove the meeting event from that participant's calendar.	Medium

uses the same format as Table 2.1 where a number of existing scheduling applications are compared against each other. As is apparent in the table, the only difference between the MSS prototype and the MSS architecture regarding these process steps is that the possibility to reschedule a meeting has not been implemented in the prototype.

Table 6.2: Comparison of how different steps of the scheduling process are done in the MSS architecture versus in the MSS prototype.

Application	Selecting participants	Finding free times	Agreeing on a time	Adding the event to a calendar
MSS Architecture design	Organizer selects from a list of registered users.	The system automatically finds available time periods from the users' calendars according to the availability rules. Time periods which are available to everyone are presented to the organizer. + Supports heterogeneous calendars + Very little effort required from the organizer and none from participants + Free times are found immediately - Users' calendars must always be connected to the system	Organizer selects a time slot which the participants can accept, decline or request rescheduling. Possible to reschedule.	Automatically added to participants' calendars. + No manual effort required
MSS Prototype	Organizer selects from a list of registered users.	The system automatically finds available time periods from the users' calendars according to the availability rules. Time periods which are available to everyone are presented to the organizer. + Supports heterogeneous calendars + Very little effort required from the organizer and none from participants + Free times are found immediately - Users' calendars must always be connected to the system	Organizer selects a time slot which the participants can accept, decline or request rescheduling. - Rescheduling not implemented	Automatically added to participants' calendars. + No manual effort required

6.2 Comparison to other existing scheduling applications

A large part of the motivation behind this thesis and the MSS were the problems with the currently existing scheduling applications. Additionally, the first success factor for the MSS that we identified in Section 1.2 was that the MSS should provide easier scheduling between heterogeneous calendar system than existing applications. In this section, we compare the scheduling process of the MSS to those of some of the popular existing meeting scheduling applications to see if the MSS is more effortless to use.

In Section 2.1, we analyzed some of the existing scheduling applications that are available today. Perhaps the most popular of those that support scheduling between people with heterogeneous calendars is Doodle (Doodle AG, 2012). The scheduling process with Doodle involves the meeting organizer guessing good suggestions for a meeting time and then waiting for the participants to respond. To respond, the participants must check their own calendars and mark which times are suitable for them. This requires quite a lot of effort from both the organizer and the participants. Much simpler scheduling features are available in, for example, Microsoft Exchange calendar (Microsoft Corporation, 2011) and Google Calendar (Google, 2012b). In these applications, the organizer is provided automatically with suggestions for suitable meeting times and is also able to see the availability of all participants directly. This allows the organizer to make an educated decision about the meeting time without having to interact with the participants manually. However, the organizer is not able to see the availability of those participants who use different calendar systems.

The scheduling process in the MSS prototype is quite similar to that of Microsoft Exchange calendar and Google Calendar. The prototype automatically finds out possible suitable meeting times and presents them to the meeting organizer. The times are not presented graphically in the prototype as they are in Exchange and Google Calendar but such a feature could be added to the prototype. After the organizer has selected a meeting time, the prototype notifies the participants about the meeting via e-mail. Meeting notifications are done similarly in the other applications. Both the prototype and the other applications also add the meeting to the participants' calendar automatically. Unlike Exchange and Google Calendar, however, the MSS prototype allows heterogeneous calendar systems to participate in the scheduling process by providing interfaces that the calendar systems can connect to. The Nokia N950 device has been successfully integrated with the prototype to verify this.

The differences in the scheduling process between the MSS architecture, the MSS prototype and the existing scheduling applications that we have inspected are described in Table 6.3. This table contains exactly the same information as the application comparison table 2.1 and the MSS architecture and prototype table 6.2 and is merely a combination of them.

To summarize, the scheduling process in the MSS prototype is roughly as simple as in the most popular competing applications. These applications cannot schedule between heterogeneous calendar systems while the MSS can. Based on these observations we can say with confidence that the MSS prototype is more effortless to use in scheduling meetings between heterogeneous calendar systems than existing applications.

Table 6.3: Comparison of how different steps of the scheduling process are done in some of the existing meeting scheduling systems as well as in the MSS architecture design and prototype.

Application	Selecting participants	Finding free times	Agreeing on a time	Adding the event to a calendar
Microsoft Exchange	Any e-mail address.	<ul style="list-style-type: none"> Calendars of other Microsoft Exchange users who are in the same domain are visible to the organizer so she is able to select a suitable time. The application also suggests times. + No effort needed from participants + Free times are found immediately - No help for scheduling with people who are not in the same domain or use a different application - Must share at least free times from the whole calendar 	<ul style="list-style-type: none"> Organizer selects a time. Meeting invitation is sent via e-mail in iCalendar format which the participants can accept, accept tentatively or decline. Possible to reschedule. 	<ul style="list-style-type: none"> Added automatically by Microsoft Exchange and possibly other applications based on an iCalendar file included in the e-mail. + No manual effort required
Google Calendar	Any e-mail address.	<ul style="list-style-type: none"> Calendars of other Google Calendar users who have shared them are visible to the organizer so she is able to select a suitable time. The application also suggests times. + No effort needed from participants + Free times are found immediately - No help for scheduling with people who have not shared their Google calendars - Must share at least free times from the whole calendar 	<ul style="list-style-type: none"> Organizer selects a time. Meeting invitation is sent via e-mail in iCalendar format which the participants can accept, accept tentatively or decline. Possible to reschedule. 	<ul style="list-style-type: none"> Added automatically by Google Calendar and possibly other applications based on an iCalendar file included in the e-mail. + No manual effort required
Doodle	<ul style="list-style-type: none"> Share a link by any means. Also possible to send e-mail directly from the service. + Freedom to choose how to share the link 	<ul style="list-style-type: none"> Organizer creates date and time suggestions. Participants mark which times are suitable for them. + Participants do not need to share calendars - Organizer must guess good suggestions - Participants have to manually check and mark their free times - Could take a long time for participants to respond while situation might change 	<ul style="list-style-type: none"> Organizer selects one or more of the suggested times. Possible to change selection later. 	<ul style="list-style-type: none"> Manually. Possible to also export an iCalendar file which has the meeting data. Possible to add directly to Google Calendar. Possible to add directly to Microsoft Outlook or Apple iCal via plug-ins. - Requires manual effort or plug-ins
Meetings	Any e-mail address.	<ul style="list-style-type: none"> Organizer creates date and time suggestions. Participants mark which times are suitable for them. + Participants do not need to share calendars - Organizer must guess good suggestions - Participants have to manually check and mark their free times - Could take a long time for participants to respond while situation might change 	<ul style="list-style-type: none"> Organizer selects one of the suggested times. Possible to change selection later. 	<ul style="list-style-type: none"> Manually. Possible to also export an iCalendar file which has the meeting data. Possible to add directly to Google Calendar. - Requires manual effort
MSS Architecture design	Organizer selects from a list of registered users.	<ul style="list-style-type: none"> The system automatically finds available time periods from the users' calendars according to the availability rules. Time periods which are available to everyone are presented to the organizer. + Supports heterogeneous calendars + Very little effort required from the organizer and none from participants + Free times are found immediately - Users' calendars must always be connected to the system 	<ul style="list-style-type: none"> Organizer selects a time slot which the participants can accept, decline or request rescheduling. Possible to reschedule. 	<ul style="list-style-type: none"> Automatically added to participants' calendars. + No manual effort required
MSS Prototype	Organizer selects from a list of registered users.	<ul style="list-style-type: none"> The system automatically finds available time periods from the users' calendars according to the availability rules. Time periods which are available to everyone are presented to the organizer. + Supports heterogeneous calendars + Very little effort required from the organizer and none from participants + Free times are found immediately - Users' calendars must always be connected to the system 	<ul style="list-style-type: none"> Organizer selects a time slot which the participants can accept, decline or request rescheduling. - Rescheduling not implemented 	<ul style="list-style-type: none"> Automatically added to participants' calendars. + No manual effort required

6.3 Expert evaluation

In this section we present the findings from an architecture evaluation session that we held with a group of experts at IPSS. The primary goal of the session was to *validate* (IEEE, 2012) the architecture and the prototype. Validation means ensuring that the system that has been built is suitable for its purpose and fulfills the needs of the users. In other words, we ensure that we have built the correct system.

6.3.1 Evaluation procedure

The expert group consisted of four persons. Three of the group members were software developers, including one integration specialist, who were not previously familiar with the architecture. The fourth member was the executive director of IPSS who has been assisting us in designing the MSS architecture from the beginning.

Because most of the experts were not previously familiar with the MSS, in the beginning of the session we presented the MSS architecture design and the prototype implementation to them. This way, they could get an understanding on what kind of design decisions we have made. The experts then evaluated the architecture and provided us with feedback on how well they think that the architecture achieves its goals.

We conducted the presentation by first describing the problem area and motivation as well as the key requirements for the MSS. Then we showed a demo video about the usage of the MSS prototype and explained on a high level how the prototype functions. After this, we described the architecturally significant decisions that we had made during the design process and the reasoning behind them. The experts asked questions about the architecture and the prototype throughout the presentation. Additionally, at the end of the presentation we requested them to evaluate the architecture based on our research questions.

To help us analyze the results from the evaluation, we recorded the audio of everything that was said in the session. After the session, we transcribed that audio into text. We analyzed the text and the notes that we had taken ourselves during the session to extract the important points and comments about the architecture and the prototype. From these points we picked the most important ones regarding the strengths and weaknesses of the architecture. We organized these points based on which research question they relate to the most. The points are presented in the following sections.

6.3.2 Evaluation of effortlessness

Research question RQ1 is about how to make scheduling between heterogeneous calendars more effortless than with currently existing applications. Effortlessness refers to how little and what kind of tasks the users have to do manually to successfully use the system. We asked the experts to evaluate how effortless the MSS seems to be compared to their experiences with other scheduling tools.

The experts unanimously agreed that the MSS appears to be more effortless to use than the other existing scheduling applications that they have used. They commented that a significant positive factor that contributes to this is that, when creating a new meeting, the MSS only takes the meeting details and the desired date range as input. This demands much less effort than, for example, Doodle (Doodle AG, 2012) because Doodle requires that the organizer manually selects each of the time slot suggestions for the meeting. The experts also pointed to another factor which is that, as output, the MSS only shows the periods of time which are available for all participants. This allows the organizer to easily select the time slot that she believes is suitable. In contrast, applications that allow the sharing of calendars between users, such as Microsoft Exchange (Microsoft Corporation, 2011), show the calendars of the meeting participants side-by-side. This can make it difficult for the organizer to find a suitable time slot if the calendars are heavily booked. It should be noted, however, that at least Microsoft Exchange and Google Calendar do have a feature that suggests meeting times for the organizer but the experts did not comment on this.

Overall, the approach of having the scheduling process of the MSS be semi-automated was deemed a good one by the experts. Semi-automated means that some parts of the process are automated while others must be manually performed. This is suitable because automating even some tasks reduces the amount of effort required from the users, which makes the MSS more pleasant to use. However, creating a completely automated process that produces satisfying results would be a too difficult problem to be tackled in the scope of this thesis.

The most significant feature of the MSS according to the experts, however, is that it supports scheduling meetings across different organizations. Because there are no other tools that would enable this as easily, the MSS should significantly reduce the amount of effort required in inter-organizational scheduling. The experts' opinion was that even connecting two different companies together via the MSS would be a large boon for the users.

6.3.3 Evaluation of the scheduling results

Research question RQ2 is about how to ensure that the scheduling results from the MSS are satisfactory to the users. If the MSS provides meeting times that are not actually suitable to the participants, then the results are unsatisfactory.

The discussion about satisfaction revolved mostly around the accuracy and the availability of the user's calendar data because actually assessing whether the results are satisfying or not is impossible without proper user testing. The accuracy and availability of the calendar data contribute towards getting satisfying results from the MSS because utilizing good quality data in the scheduling process results in more accurate meeting time suggestions.

Based on the experts' experience, it is common for people to forget to put some of their appointments into their calendar. It is then possible that, when a new meeting is scheduled using the MSS, it conflicts with the unmarked appointment. In this case, the participant would have to decline the meeting request or ask the organizer to reschedule. Some of the experts were concerned that finding a new time for a meeting that has been rejected is too cumbersome at least in the prototype. They suggested some features that could help in rescheduling a meeting. One feature idea was that the participants could still update their calendar after they have received a meeting request and initiate the rescheduling process without the organizer having to do it manually. Other ideas were the possibility for the participants to leave a comment on why they rejected a suggested time and also that the system would automatically not suggest a certain time to the organizer during rescheduling if that time was rejected previously. On the other hand, one of the experts was strongly of the opinion that it is always the organizer's responsibility to select a suitable time even when rescheduling a meeting. The expert also thought that the currently existing feature of displaying who has accepted and who has rejected the request is enough for the organizer to make an educated decision about a new meeting time.

One feature of the MSS that the experts recognized to be beneficial towards receiving satisfying results is that it uses the latest calendar data directly from the calendar systems of the users. This is better than reading the data from some secondary source, such as a server where the user's calendar data has been synchronized, because based on the experts' experience there are often problems with synchronization. This means that the data on the server could often be out-of-date and, thus, inaccurate. Reading the data directly from the calendar system where the user edits it is a way to ensure that the latest data is used in scheduling. Relating to this, however, the experts

also proposed an idea that a user could have more than one calendar data source which would be synchronized together and prioritized by the MSS. The data sources with higher priority would have the latest calendar data but the sources with lower priority would have better availability. During the meeting scheduling process, the MSS would use the highest priority data source that is available. The opinion was that even if the data on the low priority data source is out-of-date, it is still more useful to have out-of-date data in the scheduling process than none at all.

An interesting comment from one expert was that because the calendar data is utilized directly from the users' calendars it might motivate them to more rigorously maintain their calendar. This is because the users will know that the more accurate calendar data they can provide, the more suitable meeting time suggestions they receive from the MSS.

6.3.4 Evaluation of integration capabilities

Research question RQ3 is about how to make it possible to integrate a wide variety of calendar systems to the MSS. The challenge is the "wide variety" criterion because different calendar systems are based on different technologies and the MSS should have a way to support all of them.

The experts stated that it is difficult to estimate solely based on the architecture design and the current prototype how easy it is to integrate new calendar systems into the MSS. However, they believed that the interfaces in the architecture design and their implementation in the prototype should be suitable for integrating a wide range of calendar systems to the MSS.

The challenges in integrating new calendar systems into the MSS are not only technical but also related to privacy concerns. The experts believed that people and especially companies are not willing to install the MSS client component into their own or their employees' calendar systems if they do not trust that the MSS will not read more data than they have agreed to. On the other hand, since the interfaces of the MSS allow anyone to develop their own client component, the companies could also create one for themselves. This would allow them to verify how the integration works and what data is sent out to the central system. The companies might also be willing to adopt an existing integration solution if it has been certified by a trusted party.

The iCalendar standard (Dawson and Stenerson, 1998) was very briefly inspected during the session and the experts believed that it would be suitable to be used in the MSS for sending the meeting events to the participants. The experts were willing to see the standard being used in the future because it would allow the MSS to take advantage of the features of the calendar systems that are built around the iCalendar standard. For example, Microsoft

Outlook (Microsoft Corporation, 2012a) allows users to accept or reject iCalendar meeting invitations through a graphical UI.

The experts also had some comments about the integration features of the MSS prototype specifically. There was concern that the long polling technique that is used to communicate with the calendar systems might be quite demanding for the calendar systems especially regarding power consumption. Additionally, it is not possible to estimate how many long polling connections the central server is able to maintain simultaneously which is a cause for concern. On the other hand, long polling was considered suitable for the current prototype due to its simplicity. It was agreed that the long polling protocol should be changeable to something else if a more suitable communication protocol becomes available. The experts also believed that it might be a better choice to send the meeting requests via SMS or using iCalendar format because the currently used plain e-mail messages might be easily missed by people who receive a lot of e-mail.

On the positive side, the experts observed that since the client component in the prototype is done with only several hundred lines of Python code it suggests that the client component should be simple to implement for any device.

The amount of data traffic that the MSS generates should not be a problem because the experts estimated that the MSS will most likely be used to schedule meetings that have 5 to 10 participants. Smaller meetings are likely so easy to schedule informally that users do not bother to use the MSS. Large meetings, on the other hand, are rarely scheduled by agreeing on a time with all participants but instead the organizer just dictates the meeting time directly. Also, in the prototype, the size of each message between the central system and the calendar systems is quite small so a large number of messages is required before they become a problem.

6.3.5 Evaluation of privacy protection

Research question RQ4 is about how to prevent the private calendar data of the users from leaking to other users. Of course, the MSS must access and share some calendar data to be able to schedule meetings, so an important question is how strictly do the users want to protect their calendar data.

We asked the experts about their opinions on what is an acceptable amount of calendar data that they would be willing to share to the central server. They all agreed that they are willing to divulge their free times but only to a limited group of people that they have voluntarily joined. They also want to be able to detach themselves from this group and stop sharing their calendar information at any point.

The experts highlighted the user groups as a privacy concern. The user groups that exist in the system should not be visible to every other user but instead to only those who are members of these groups. In this case, new users could only join groups by being invited by existing members.

These privacy concerns are not present as requirements for the MSS and they have not been taken specifically into account in the architecture design. However, the current architecture design does not prevent these issues to be corrected in the implementation. Currently in the prototype, every user is able to see every other user and user group that exists in the system. Every user is also able to organize a meeting and invite anyone. This basically allows the organizer to see the free times of any user that she desires. Based on the feedback from the experts, these issues should be corrected in future versions.

6.3.6 Other evaluation findings

During the session, the experts also gave some other comments about the MSS that do not specifically relate to any of the research question topics.

They remarked that assumption AS01, which states that each user only has one calendar, is actually quite realistic. Based on the experts' experience, people often have only one master device that they primarily use. If they have more than one device, they can be expected to handle the synchronization between those devices themselves.

One expert asked us if we had thought about the ease of installation of the client application that integrates a calendar system with the MSS. While we recognize that this is an important issue since a complicated installation process makes the MSS unappealing to new users, this is not an architecturally significant problem. It also is not within the scope of this thesis, as we outlined in Section 1.3. There was also some discussion about including the location context when searching for available times for a user. This would mean that the physical location of the user would impact her availability. However, we have also left the location context out of our scope, as stated in Section 1.3.

The experts had many small improvement suggestions for the prototype. Especially the user and user group management features could be improved in many ways. There was even some discussion about integrating the MSS with large external user management systems. However, one of the experts was of the opinion that the MSS will most likely be used among a small circle of people and not with a large number of unknown people. For this reason, the integration of external user management systems is unnecessary. Also in other respects the user and user group management are not interesting

to us in the context of this thesis since they are not specifically related to meeting scheduling or any architectural issues of the MSS. One expert also pointed out that new features should always be added with caution because they increase the complexity of the MSS.

6.3.7 Expert evaluation summary

The following list summarizes the most important points made by the experts in the evaluation of the architecture.

- They were certain that the MSS is more effortless to use than existing scheduling applications thanks to the simple input and clear output.
- Possibility to schedule meetings between different organizations is a great advantage compared to existing scheduling applications.
- The fact that the calendar data is read just-in-time directly from the calendar should result in accurate meeting time suggestions.
- Some concern that rescheduling a meeting time that some participants have rejected is cumbersome.
- Difficult to estimate how easy it is to integrate new calendar systems to the MSS but the current interfaces should be helpful in this regard.
- Currently privacy issues with the visibility of time periods and user groups in the prototype. Correcting these is possible within the current architecture design, however.

Chapter 7

Discussion

In this chapter we discuss the results of our research. Most importantly we summarize the answers to our research questions. We also identify the issues that could threaten the validity or accuracy of our results.

7.1 Answers to the research questions

In this section we return to the foundation of the research in this thesis by answering the research questions that we set in section 1.2.

RQ1. How to enable the scheduling of meetings between people who use heterogeneous calendar systems in a way that is more effortless than with currently existing tools or methods?

The problem with scheduling meetings between heterogeneous calendar systems using the currently available tools and methods is that they require significant manual effort from the participants to pin down and communicate their own availability. This is because the calendar systems themselves, where the users store their calendar information, are not utilized directly at all. In traditional methods, such as negotiating over e-mail or phone, it is the participants who are responsible for knowing and communicating their availability to others. Whether they want to use some calendar system or keep it all in their memory is up to them. Even the software applications that can help in scheduling meetings between heterogeneous calendar systems, such as Doodle (Doodle AG, 2012), still leave it up to the users to mark their own available times. The only thing these tools can do to help schedule meetings is to reduce the amount of communication required between the participants.

The software applications that remarkably simplify the meeting scheduling process are those that directly utilize the calendars of the participants.

For example, Microsoft Exchange (Microsoft Corporation, 2011) allows the organizer to see the calendars of all participants which allows her to make an educated decision about a suitable meeting time. The problem with these systems is that they do not support heterogeneous calendar systems.

Based on these observations, the solution to simplifying meeting scheduling between heterogeneous calendar systems is to find a way to directly access the calendar data in all of these heterogeneous systems. There does not seem to be any scheduling applications available currently that would do this. We have designed an architecture and implemented a prototype of an application that is capable of integrating into and reading the calendar of heterogeneous calendar systems.

RQ2. How to ensure that the scheduled meeting times are satisfactory to the users?

As we mentioned when discussing satisfaction as a success factor for the MSS in section 1.2, a way to increase satisfaction with the results of the scheduling process is to allow the participants to provide input for the scheduling process while still taking most of the manual effort away from them with automation. In the MSS architecture, the finding of possibly suitable meeting times is done automatically so that the users don't have to manually check their own availability or communicate it to others. But the MSS also allows the users to influence the scheduling results in two ways.

First, they can decide which times in their calendars are considered free by the automatic scheduling algorithm by specifying availability rules. This allows users to easily avoid having meetings proposed at times when they certainly do not want to have one. The rules are also user group specific so that the users can adjust their time preferences based on the social context of the meeting.

Second, the automatic scheduling algorithm only proposes possible meeting times instead of selecting one automatically. This allows the organizer to make the final time slot selection. The organizer might have some knowledge about what times are actually suitable for a meeting that the automatic scheduling algorithm does not have, so allowing her to select the time slot personally should help avoid the non-satisfactory time slots that the automatic algorithm might produce. Additionally, after the organizer has selected the time slot, the MSS still allows the individual participants to notify the organizer whether the selected time slot is suitable for them or not. This is because the participants of course have better knowledge about what times are suitable for them personally than the organizer.

Although this arrangement could result in similar problems that are possible in manual scheduling, such as participants keeping constantly rejecting

the time slot suggestions of others, we believe that allowing the participants to make the final decision about the meeting time is the best way to ensure satisfaction.

RQ3. How to enable the MSS to take advantage of a wide variety of existing calendar systems?

The difficulty in utilizing the various calendar systems that people might be using is that they are running on different technologies. Some systems might be offering a public interface that can be used to read and write data into their calendar but most calendar systems probably do not. For example, Google Calendar (Google, 2012b) is a web calendar service that does offer an interface for outsiders to utilize. In contrast, the calendar on the Nokia N950 mobile phone is an SQLite database that cannot be easily accessed from outside of the device. Because the calendar systems that could possibly be used with the MSS are so widely different, there is no way for us to know in the designing phase what kind of calendar system specific integration solutions we would need. At best, we could select a few different calendar systems and integrate the MSS with only those but this would not qualify as taking advantage of a "wide variety" of existing calendar systems.

What we did to solve this problem was to shift the responsibility of integrating the various calendar systems away from us and to the owners of those systems. In other words, we did not develop an integration to any calendar system specifically but we provided the means for any calendar system to be integrated into the MSS. This, of course, makes it more difficult for users of new calendar systems to take advantage of the MSS because someone must first develop the client component for that calendar system. Still, we think this is the only way to realistically integrate the MSS into a wide variety of calendar systems. Also, the experts who evaluated the MSS architecture in Section 6.3.4 agreed that the interfaces should be helpful in integrating different types of calendar systems.

In the MSS architecture design, we have made the integration of devices possible by providing common interfaces from the central system that anyone can use to take advantage of the scheduling and meeting organizing features of the MSS. How these interfaces are implemented in practice is another issue, however, because, as said, the various calendar systems are based on different technologies so it might be difficult to implement the interfaces in such a way that they would support all of these calendar systems. While it might be impossible to support all systems, we believe that it is possible to make educated choices about the implementation of the interfaces to maximize the number of calendar systems that can take advantage of them. In the MSS

prototype implementation, we chose to use interfaces that work using HTTP requests, due to the pervasiveness of HTTP and the high likelihood that any given calendar system will support it.

RQ4. How to avoid exposing too much user's private calendar information to other users while allowing the MSS to function effectively?

In the expert evaluation of the architecture in Section 6.3.5, we asked the experts about what is an acceptable amount of calendar data to be shared. Their opinion was that it is acceptable to share free times but only to a limited group of users. In the MSS architecture and prototype, users can only access the available times of other users because the complete calendar data of the users is never sent to the central system. It is the responsibility of the client component, which is installed on the user's calendar system, to read the calendar data and extract the free times; combine the free times with the availability rules; and send the available times to the central system.

Additionally, in the prototype, the visibility of the available times on the central system is limited to only the user herself and the organizer of the meeting for which the times have been sent. However, to fully satisfy the requirement of limiting the visibility of the free times, it should also be possible to limit who is able to organizer meetings for which participants. Currently this is not possible in the prototype but it is not restricted by the architecture design. Currently, the prototype also allows the users to see every user group in the MSS. According to the expert evaluation, this is a privacy problem that should be corrected. Changing the visibility of the groups is also permitted by the current architecture design.

7.2 Threats to the validity of the results

The results of this thesis are the architecture design and the prototype implementation of the MSS. In this section we identify some issues that might undermine the validity and accuracy of the results.

The main method of verifying the feasibility of the architecture design is through the prototype implementation that we have made. The prototype includes all of the features that are essential to the functionality of the MSS including meeting creation, gathering of available time periods, sending and responding meeting requests etc. However, due to time constraints, the integration to external calendar systems was done for only one type of device: the Nokia N950. Integrating only one type of calendar system to the MSS is not

adequate to concretely display that the architecture can support integration to a wide range of heterogeneous calendar systems. Theoretically there is no reason why such integration would not be possible but, again, the prototype fails to prove it in practice. To properly evaluate the integration capabilities of the architecture, the prototype integration should be extended to at least a few different calendar systems that are based on different technologies. For example, an integration could be created for the Microsoft Outlook (Microsoft Corporation, 2012a) desktop application and the Google Calendar (Google, 2012b) web application in addition to the N950 integration that we have already made.

Mostly due to the lacking integration support and also the overall unpolished state of the prototype, it is difficult to conduct any thorough user testing with it in real-world situations. For this reason we have not been able to properly assess the usefulness of the MSS prototype or the architecture in practice. This is a problem because we chose design science as our research methodology and the usefulness of the design science artifacts is the best criterion to measure the results of the research. The best way to fix this would be to continue developing the prototype to get it into a state where it can be used for user testing and to gather feedback from the users. However, we have evaluated the architecture and the prototype in Chapter 6 by comparing the effortlessness of its use against that of other meeting scheduling applications. Effortlessness is an important factor in the usefulness of a system like the MSS. We also verified and validated our results by tracing the prototype and the architecture back to the specification and by allowing a group of experts to evaluate the architecture. These evaluation efforts should give at least some confidence that the architecture design and the prototype are useful.

Chapter 8

Conclusions

In this chapter we conclude the thesis. We also discuss some ideas for how the research made in this thesis could be expanded upon in the future.

8.1 Conclusions

Finding a suitable time for a meeting between multiple people is often a cumbersome and a time-consuming task. The difficulty with meeting scheduling is that individual meeting participants initially only have full knowledge of their own schedule and not of the schedules of the other participants. The meeting scheduling process involves the participants sharing information about their available times with each other or otherwise negotiating possible meeting times. Depending on the method used in the sharing or the negotiation, the process can take a long time and involve numerous transactions between the participants. Many software applications exist that seek to make the meeting scheduling process more effortless for the participants. For example, some applications allow users to share calendar information directly with each other so that it is easy for one user to see when every participant is available. Some other applications provide tools that are meant to simplify the negotiation about meeting times. However, these applications often have shortcomings that make them unsatisfying to use or unusable in some use cases. For example, negotiating meeting times using the provided tools can still be too cumbersome to be satisfying. The sharing of calendar information is limited to within a single domain and domains often encompass only single organizations. This means that scheduling meetings between people from different organizations must still be done manually.

In this thesis we designed a new meeting scheduling system that enables inter-organizational meeting scheduling in an effortless way. More specifi-

cally, we designed a system that automates parts of the meeting scheduling process between people who use heterogeneous calendar systems by directly reading the calendar data from those calendar systems. A calendar system is any type of device or system that contains a person's calendar information, such as a mobile phone; a desktop calendar application; or a calendar server. Solving the problem of enabling scheduling between heterogeneous calendar systems in an effortless way is one of the research questions of this thesis. Other research questions focus on how to ensure that the scheduling results are satisfying, how to extend the integration support to a wide range of heterogeneous systems and how to protect the privacy of the user's calendar information. The design science approach was selected as the research methodology for this thesis. The contributions of this thesis are a software architecture for a meeting scheduling system (MSS) that embodies the answers to these research questions and a prototype of the MSS based on the architecture design.

The architecture is divided into two main types of components. There is a single central system component that contains most of the logic and data that is needed in the meeting scheduling process. The other components are the calendar systems. The calendar systems contain the calendar information of the users which is automatically read during a meeting scheduling process to find out the time periods when that user is available for a meeting. The calendar systems are connected to the central system so that the time periods from multiple users can be gathered to the central system. Due to privacy concerns, the calendar information never leaves the calendar system as-is. Instead, only the available time periods, which are calculated by combining the calendar events and the availability rules, are sent to the central system. Based on the time periods from all participants, the central system is able to calculate what times are available to all participants simultaneously and suggest suitable meeting times. However, there are some factors that might affect the suitability of some time slots that the automated scheduling algorithm cannot take into account, such as events that have not been marked to the calendar. For this reason, the meeting organizer is required to make the final selection of the meeting time based on the suitable times calculated by the MSS. Additionally, the other participants of the meeting are asked to accept or reject this time slot selection. These manual steps are taken to ensure that the meeting time that eventually gets selected will be satisfactory to the participants.

Connecting a calendar system to the central system requires that a client component of the MSS is integrated into the calendar system. The client component reads the calendar information from the user's calendar and sends the available time periods to the central system. The calendar systems con-

nect to the central system via generic interfaces which are also open to third parties. This allows third party developers to create a client components for any calendar system that they wish. In this way, the MSS can be integrated to any calendar system regardless of its type.

The prototype contains all of the key features of the architecture. It is possible to perform an entire meeting scheduling process with the prototype. The only missing features relate to user group management and meeting management that happens after the meeting has been scheduled. The central system in the prototype is a web server. The web server allows registered users to organize new meetings and view information about the meetings that they have organized previously. The interfaces between the server and the calendar systems are implemented as HTTP-based REST- and long polling interfaces. One type of calendar system is integrated into the prototype: the Nokia N950 mobile phone. The client component is implemented on the phone as a script file. The participants receive notifications about new meetings via e-mail and can accept, reject or request rescheduling for the meeting by clicking links in the message.

The architecture and the prototype were evaluated in three ways. We traced the prototype implementation and the architecture design back to the requirements and research questions to verify that the design matches the original specification. It was found that all of the concerns in the research questions and requirements were taken into account in the architecture and the prototype. We compared the prototype to other existing meeting scheduling applications to ensure that the prototype enables scheduling between heterogeneous calendar systems more effortlessly than the other applications. This was found to be the case. Finally, we allowed a group of experts to evaluate the architecture and the prototype to validate that they fulfill the needs of the users that are the motivation behind the MSS. The experts agreed that the prototype is more effortless to use than other scheduling applications. They were also optimistic that the interfaces in the prototype should be suitable for integrating heterogeneous calendar systems into the MSS, although they stated that it is difficult to estimate this accurately based on the current prototype. It was also found that the experts' privacy criteria are not entirely matched by the current prototype but these issues can be corrected and are not restricted by the architecture design.

8.2 Future work

Many of the threats to the validity of the results in this thesis could be addressed by continuing to work on the prototype implementation in the future.

Developing an integration support for more systems than just the N950 would provide more insight into if and how the architecture could be improved in the future to better support different heterogeneous calendar systems. Our focus in this thesis was on mobile phones but the requirements to integrate, for example, web services or desktop applications differ somewhat from those that we have identified. Especially integrating the MSS with large external CRM systems that govern the calendar data of dozens of users is a topic that should be studied in practice. Some of the implementation decisions that we have made in the prototype should also be studied in more detail. For example, our decision to use the long polling technique for communication between the central system and the calendar systems sets some requirements to both systems that we have not studied. Having a better understanding of the implications of these implementation decisions would be helpful in creating future versions of the MSS. Additionally, improving the prototype in other ways as well could allow it to be used in users testing. Feedback from the users would be valuable in refining the perceptions about what are the success factors for this kind of a meeting scheduling system. A better understanding of the success factors might lead to different approaches in designing the architecture for future systems.

One challenging topic for future work is to improve the automatic scheduling process to make it more "intelligent". In our architecture, the automatic scheduling process takes into account user preferences by allowing the users to limit their calendar availability based on the user group of the meeting. The user group could be seen as the social context of the meeting. But as we discussed already in section 1.3, meetings can also have other contexts, such as location or topic, that might influence whether invited users want to or are able to participate in that meeting. For example, the location context could indicate how long it takes for a user to get from her previous appointment to the meeting that is being scheduled. The scheduling algorithm would then not suggest such time slots to the organizer that would leave too little time for the participants to travel to the meeting. The topic context, on the other hand, could allow users to set preferences about when they want to participate in meetings about certain topic, or if they even want to participate at all. The challenges in taking these contexts into account in the automatic scheduling process is to identify what are the relevant contexts that should be acknowledged (is topic a meaningful context?), how can these contexts be classified; measured; or tracked (what different topics there are?), and what kind of preferences are the users able to specify for these contexts (accept or reject invitations automatically based on the topic or some kind of priority system?). The benefit of making the scheduling process more intelligent is that it provides results that the users are satisfied with. Additionally, some

of the burden of decision making can be taken away from the users as they can rely more on the automatic process.

The social context which we have taken into account in our architecture design could be considered more carefully in future work. In our design, the social context is based only on the user groups but an alternative is to regard the meeting participants as individuals. This would give the users more refined control over their preferences about who to meet with and when. The handling of the social context could also be expanded in other ways. For example, it could be useful to be able to prioritize the importance of different participants already when creating the meeting. Some participants might be required attend while other non-essential participants could be optional. The automatic scheduling algorithm would then suggest all of the times when there required participants have free time even if all of the optional participants do not.

Bibliography

- Apple Inc. Apple OS X operating system product website, 2012a. [online] Available at: <http://www.apple.com/osx/> [Accessed 8.8.2012].
- Apple Inc. Apple iOS mobile device operating system product website, 2012b. [online] Available at: <http://www.apple.com/ios/> [Accessed 20.9.2012].
- T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.
- I. Bilogrevic, M. Jadliwala, P. Kumar, S. S. Walia, J.-P. Hubaux, I. Aad, and V. Niemi. Meetings through the cloud: Privacy-preserving scheduling on mobile devices. *Journal of Systems and Software*, 84(11):1910 – 1927, 2011. ISSN 0164-1212. doi: 10.1016/j.jss.2011.04.027.
- A. E. Blandford and T. R. G. Green. Group and individual time management tools: What you get is not what you need. *Personal Ubiquitous Comput.*, 5(4):213–230, Jan. 2001. ISSN 1617-4909. doi: 10.1007/PL00000020.
- S. Burbeck. *Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC)*. Softsmarts, Incorporated, 1987.
- D. Crockford. The application/json media type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006.
- F. Dawson and D. Stenerson. Internet calendaring and scheduling core object specification (iCalendar). RFC 2445 (Proposed Standard), Nov. 1998. Obsoleted by RFC 5545.
- L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski. A personal learning apprentice. In *Proceedings of the tenth national conference on Artificial intelligence, AAAI'92*, pages 96–103. AAAI Press, 1992. ISBN 0-262-51063-4.

- Django Software Foundation. Django web framework for Python website, 2012. [online] Available at: <https://www.djangoproject.com/> [Accessed 20.9.2012].
- Doodle AG. Doodle scheduling service website, 2012. [online] Available at: <http://www.doodle.com> [Accessed 8.8.2012].
- S. F. Ehrlich. Social and psychological factors influencing the design of office communications systems. *SIGCHI Bull.*, 17(SI):323–329, May 1986. ISSN 0736-6906. doi: 10.1145/30851.275651.
- C. Glezer. A conceptual model of an interorganizational intelligent meeting-scheduler (IIMS). *The Journal of Strategic Information Systems*, 12(1):47 – 70, 2003. ISSN 0963-8687. doi: 10.1016/S0963-8687(02)00034-3.
- Google. Android mobile device operating system product website, 2012a. [online] Available at: <http://www.android.com/> [Accessed 20.9.2012].
- Google. Google Calendar feature overview website, 2012b. [online] Available at: <http://www.google.com/intl/en/googlecalendar/about.html> [Accessed 8.8.2012].
- S. Gregor and D. Jones. The anatomy of a design theory. *Journal of the Association for Information Systems*, 8(5):312–335, 2007.
- T. Haynes, S. Sen, N. Arora, and R. Nadella. An automated meeting scheduling system that utilizes user preferences. In *Proceedings of the first international conference on Autonomous agents*, AGENTS '97, pages 308–315, New York, NY, USA, 1997. ACM. ISBN 0-89791-877-0. doi: 10.1145/267658.267733.
- Hello Industries, Inc. Hello Scheduling employee scheduling software product website, 2010. [online] Available at: <http://www.helloscheduling.com> [Accessed 8.8.2012].
- A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):pp. 75–105, 2004. ISSN 02767783.
- K. Higa, B. Shin, and V. Sivakumar. Meeting scheduling: An experimental investigation. In *Systems, Man, and Cybernetics, 1996., IEEE International Conference on*, volume 3, pages 2023 –2028 vol.3, oct 1996. doi: 10.1109/ICSMC.1996.565442.

- IEEE. IEEE standard for system and software verification and validation. *IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004)*, pages 1 –223, 25 2012. doi: 10.1109/IEEESTD.2012.6204026.
- W. S. Jeong, J. S. Yun, and G. S. Jo. Cooperation in multi-agent system for meeting scheduling. In *TENCON 99. Proceedings of the IEEE Region 10 Conference*, volume 2, pages 832 –835 vol.2, dec 1999. doi: 10.1109/TENCON.1999.818547.
- J. F. Kelley and A. Chapanis. How professional persons keep their calendars: Implications for computerization. *Journal of Occupational Psychology*, 55(4):241–256, 1982. ISSN 2044-8325. doi: 10.1111/j.2044-8325.1982.tb00098.x.
- C. M. Kincaid, P. B. Dupont, and A. R. Kaye. Electronic calendars in the office: An assessment of user needs and current technology. *ACM Trans. Inf. Syst.*, 3(1):89–102, Jan. 1985. ISSN 1046-8188. doi: 10.1145/3864.3868.
- R. Kozierok and P. Maes. A learning interface agent for scheduling meetings. In *Proceedings of the 1st international conference on Intelligent user interfaces, IUI '93*, pages 81–88, New York, NY, USA, 1993. ACM. ISBN 0-89791-556-9. doi: 10.1145/169891.169908.
- D. Lindsley, C. Soyland, M. Croydon, J. Bohde, and I. Kelly. Tastypie web API framework for Django website, 2012. [online] Available at: <http://django-tastypie.readthedocs.org/en/latest/index.html> [Accessed 20.9.2012].
- S. T. March and G. F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251 – 266, 1995. ISSN 0167-9236. doi: 10.1016/0167-9236(94)00041-2.
- A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy Approach to Requirements Syntax (EARS). In *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, pages 317 –322, 31 2009-sept. 4 2009. doi: 10.1109/RE.2009.9.
- B. McAllister. Unobtrusive Date-Picker Widget V5 JavaScript calendar UI widget, 2012. [online] Available at: <http://www.frequency-decoder.com/2009/09/09/unobtrusive-date-picker-widget-v5> [Accessed 20.9.2012].
- Meetin.gs Ltd. Meetin.gs meeting organizing service website, 2012. [online] Available at: <http://www.meetin.gs> [Accessed 8.8.2012].

- Microsoft Corporation. Microsoft Exchange Server and Exchange Online website, 2011. [online] Available at: <http://www.microsoft.com/exchange/en-us/default.aspx> [Accessed 24.7.2012].
- Microsoft Corporation. Microsoft Office Outlook e-mail and calendar application website, 2012a. [online] Available at: <http://office.microsoft.com/en-us/outlook/> [Accessed 11.9.2012].
- Microsoft Corporation. Microsoft Windows operating system product website, 2012b. [online] Available at: <http://windows.microsoft.com/en-US/windows/home> [Accessed 8.8.2012].
- Mimosa Software Ltd. Mimosa scheduling software product website, 2012. [online] Available at: <http://www.mimosasoftware.com> [Accessed 8.8.2012].
- J. Mosier and S. Tammara. When are group scheduling tools useful? *Computer Supported Cooperative Work (CSCW)*, 6(1):53–70, 1997.
- Oracle Corporation. MySQL open source database website, 2012. [online] Available at: <http://www.mysql.com/> [Accessed 25.9.2012].
- S. J. Payne. Understanding calendar use. *Hum.-Comput. Interact.*, 8(2): 83–100, June 1993. ISSN 0737-0024. doi: 10.1207/s15327051hci0802_1.
- K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45 – 77, 2007. ISSN 07421222.
- J. Postel. Simple mail transfer protocol. RFC 821 (Standard), Aug. 1982. Obsoleted by RFC 2821.
- PostgreSQL Global Development Group. PostgreSQL open source database website, 2012. [online] Available at: <http://www.postgresql.org/> [Accessed 25.9.2012].
- T. Reenskaug. Models-views-controllers. *Technical note, Xerox PARC*, 1979.
- K. Reitz. Requests HTTP library for Python website, 2012. [online] Available at: <http://docs.python-requests.org/en/latest/index.html> [Accessed 20.9.2012].
- E. Rogers. *Diffusion of Innovations*. Marketing/Social science / The Free Press. Free Press, 1995. ISBN 9780029266717.

- N. Rozanski and E. Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2011. ISBN 9780321718334.
- A. Sell. Mobile digital calendars in knowledge work. *International Journal of Mobile Communications*, 6(6):696–713, 2008.
- A. Sell and P. Walden. Mobile digital calendars: An interview study. In *System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 1, page 23b, jan. 2006. doi: 10.1109/HICSS.2006.349.
- S. Sen and E. Durfee. On the design of an adaptive meeting scheduler. In *Artificial Intelligence for Applications, 1994., Proceedings of the Tenth Conference on*, pages 40–46, mar 1994. doi: 10.1109/CAIA.1994.323694.
- M. Shaw. The coming-of-age of software architecture research. In *Proceedings of the 23rd International Conference on Software Engineering, ICSE '01*, pages 656–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1050-7.
- M. Shaw. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7, Oct. 2002.
- M. Shaw. Writing good software engineering research papers: Minitutorial. In *Proceedings of the 25th International Conference on Software Engineering, ICSE '03*, pages 726–736, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1877-X.
- T. E. Starner, C. M. Snoeck, B. A. Wong, and R. M. McGuire. Use of mobile appointment scheduling devices. In *CHI '04 extended abstracts on Human factors in computing systems, CHI EA '04*, pages 1501–1504, New York, NY, USA, 2004. ACM. ISBN 1-58113-703-6. doi: 10.1145/985921.986100.
- D. Stenberg. cURL open source data transfer application website, 2012. [online] Available at: <http://curl.haxx.se/> [Accessed 10.8.2012].
- B. van den Hooff. Electronic coordination and collective action: Use and effects of electronic calendaring and scheduling. *Information & Management*, 42(1):103 – 114, 2004. ISSN 0378-7206. doi: 10.1016/j.im.2003.12.006.
- D. Wu and M. Tremaine. Knowledge worker adoption of time management tools: Satisfaction and perceived effectiveness. In *AMCIS 2004 Proceedings, Paper 433*, 2004.

Appendix A

REST Interface Methods and Fields

This appendix lists the details of the resources of the REST interface in the MSS prototype. The included details are the unique URL of each resource or object, the HTTP methods accepted by each resource and the data fields of each resource. The usage of the REST interface is described in Section 5.4.

Table A.1 displays the URL of each resource (object list) and object. The URLs are used to access the resources by sending HTTP requests to them. The bracketed portions of the URL, such as `<meeting_id>`, are to be replaced with integers that identify the object in question.

Table A.1: REST interface resource URLs.

Resource	URL
Meeting list	<code>/rest/v1/meeting/</code>
Meeting	<code>/rest/v1/meeting/<meeting_id>/</code>
Time period list	<code>/rest/v1/meeting/<meeting_id>/timeperiod/</code>
Time period	<code>/rest/v1/meeting/<meeting_id>/timeperiod/ <time_period_id>/</code>
Request response list	<code>/rest/v1/meeting/<meeting_id>/response/</code>
Request response	<code>/rest/v1/meeting/<meeting_id>/response/ <response_id>/</code>
User list	<code>/rest/v1/user/</code>
User	<code>/rest/v1/user/<user_id>/</code>
Group list	<code>/rest/v1/group/</code>
Group	<code>/rest/v1/group/<group_id>/</code>

Table A.2 lists the actions that are taken on the resources based on different HTTP methods. Not all methods are supported by all resources. The

result can also depend on the what the authenticated user's relationship to that object is.

Table A.2: REST interface methods for each resource.

Resource	Method	Authentication	Result
Meeting list	GET	Any user	Get a list of meetings that the user is invited in or is the organizer of. Common time periods of a meeting can only be seen from meetings where the user is the organizer.
Meeting list	POST	Any user	Create a new meeting. The user is set as the organizer for that meeting.
Meeting	GET	Invited user, Organizer	Get the meeting details. Only the organizer can see the common time periods of that meeting.
Meeting	PUT	Organizer	Update the meeting details (all fields).
Meeting	PATCH	Organizer	Update the meeting details (only given fields).
Time period list	GET	Invited user	Get the time periods committed by the user for the meeting.
Time period list	GET	Organizer	Get the time periods committed by all the invited users of the meeting.
Time period list	POST	Invited user	Commit the user's time periods to the meeting. Only one commit per user allowed.
Time period	GET	Time period owner	Get the time period.
Time period	GET	Organizer	Get the time period.
Request response list	GET	Invited user	Get the response submitted by the user for the meeting.
Request response list	GET	Organizer	Get all the responses submitted by all invited users of the meeting.
Request response list	POST	Invited user	Submit a new response to the meeting by the user. Only one response per user allowed.
Request response	GET	Response owner, Organizer	Get the response.
Request response	PUT	Response owner	Change the response.
User list	GET	Any user	Get a list of users.
User list	POST	Outsiders	Add a new user.
User	GET	Any user	Get the details of a user.

Table A.2: REST interface methods for each resource. (continued)

Resource	Method	Authentication	Result
User	PUT	User herself	Update user (all fields)
User	PATCH	User herself	Update user (some fields)
User	DELETE	User herself	Remove the user
Group list	GET	Any	Get a list of groups.
Group list	POST	Any	Add a new group.
Group	GET	Any	Get the details of a group.
Group	PUT	Group member	Change the details of the group.
Group	PATCH	Group member	Change some details of the group
Group	DELETE	Group member	Remove the group.

Table A.3 and the following tables describe the data fields of each resource. These fields are received when requesting a resource with the GET method and sent with the POST, PUT and PATCH methods. The information for each field includes name, data type, and a description of the data that the field contains. There are also notes on most fields that specify any limitations regarding that field. *Read only* fields are only received in a response but cannot be sent with a request. *Write only* fields can only be sent but are not visible in responses. *Required* fields must be sent when creating a new object.

Table A.3: Data fields of the Meeting REST resource.

Field	Type	Notes	Description
id	integer	<i>read only</i>	Unique identifier for the meeting. Automatically set.
topic	string	required	Topic or name of the meeting.
location	string	required	Where the meeting is held. Currently this is only a string but if in the future it would be desirable to take the location into account in the scheduling process this field should also probably include some additional information, such as GPS coordinates.
duration	integer	required	Length of the meeting in minutes.
date_range_start	date (YYYY-MM-DD)	required	The start of the range of dates between which possible time slots will be searched and suggested by the system. The given date is included in the range.

Table A.3: Data fields of the Meeting REST resource. (continued)

Field	Type	Notes	Description
date_range_end	date (YYYY-MM-DD)	required	The end of the range of dates between which possible time slots will be searched and suggested by the system. The given date is included in the range.
organizer	User	<i>read only</i>	The user who created the meeting and is allowed to change it and select the time slot. This is automatically set to be the user who creates the meeting and can not be changed manually.
group	Group	required	The group of users that participate in the meeting. Used in the scheduling process by the clients to select the appropriate availability rules to apply when searching for available times. As an exception to other relationship fields, this field returns the full details of the group, not just a URI of the related Group resource to reduce the number of required requests.
invitees	list of Users	required	List of users who are invited to the meeting. Does not have to match the member list of the selected group.
common_time_periods	string	<i>read only</i>	The periods of time that are available to all users, calculated during the scheduling process from the individual time periods of invited users. The periods are presented as single string formatted in a specific way: The different periods are separated by a semicolon (;) and each period consists of the starting and ending time of that period, separated by a comma (,). Both the starting and ending times are formatted YYYY-MM-DD HH:MM and are included in the period. Only visible to the organizer.

Table A.3: Data fields of the Meeting REST resource. (continued)

Field	Type	Notes	Description
time_slot	datetime (YYYY-MM-DDTHH:MM:SS)	-	The date and time when the meeting is held.
cancelled	boolean	-	Shows wether the meeting is cancelled or not.
creation_date	datetime (YYYY-MM-DDTHH:MM:SS.ms)	<i>read only</i>	The date and time when the meeting instance was created. Automatically set.
update_date	datetime (YYYY-MM-DDTHH:MM:SS.ms)	<i>read only</i>	The date and time when the meeting instance was last updated. Automatically set.
cancellation_date	datetime (YYYY-MM-DDTHH:MM:SS.ms)	<i>read only</i>	The date and time when the meeting was canceled. Automatically set and null if the meeting isn't cancelled.
committed_time_period_count	integer	<i>read only</i>	The number of users that have committed their available time periods for this meeting.
response_count	integer	<i>read only</i>	The number of users that have submitted their response to the meeting request.
time_period_uri	string	<i>read only</i>	URI where a list of submitted time periods for this meeting can be retrieved or new time periods committed.
response_uri	string	<i>read only</i>	URI where a list of responses for this meeting can be retrieved or new responses submitted.
resource_uri	string	<i>read only</i>	The REST URI that points to this resource.

Table A.4: Data fields of the Time Period REST resource.

Field	Type	Notes	Description
id	integer	<i>read only</i>	Unique identifier for the time periods. Automatically set.
meeting	Meeting	<i>read only</i>	The meeting where these time periods have been committed to. This is set automatically based on the meeting id given in the url.

Table A.4: Data fields of the Time Period REST resource. (continued)

Field	Type	Notes	Description
user	User	<i>read only</i>	The user who committed these time periods. Set automatically based on the user who made the request.
time_periods	string	required	The periods of time from the user that are available for the meeting. The periods are presented as single string formatted in a specific way: The different periods are separated by a semicolon (;) and each period consists of the starting and ending time of that period, separated by a comma (,). Both the starting and ending times are formatted YYYY-MM-DD HH:MM and are included in the period.
commit_date	datetime (YYYY-MM-DDTHH:MM:SS.ms)	<i>read only</i>	The date and time when these time periods were committed. Automatically set.
resource_uri	string	<i>read only</i>	The REST URI that points to this resource.

Table A.5: Data fields of the Request Response REST resource.

Field	Type	Notes	Description
id	integer	<i>read only</i>	Unique identifier for the response. Automatically set.
meeting	Meeting	<i>read only</i>	The meeting where this response has been submitted to. This is set automatically based on the meeting id given in the url.
user	User	<i>read only</i>	The user who submitted this response. Set automatically based on the user who made the request.
response	string	required	The actual response. A single character and one of three options: 'A' for 'accept invitation', 'R' for 'request rescheduling' or 'D' for 'decline invitation'.

Table A.5: Data fields of the Request Response REST resource. (continued)

Field	Type	Notes	Description
response_date	datetime (YYYY-MM-DDTHH:MM:SS.ms)	<i>read only</i>	The date and time when this response was submitted. Automatically set.
resource_uri	string	<i>read only</i>	The REST URI that points to this resource.

Table A.6: Data fields of the User REST resource.

Field	Type	Notes	Description
id	integer	<i>read only</i>	Unique identifier for the user. Automatically set.
username	string	required	Unique name for the user. Used to authenticate to the system.
password	string	required, write only	User's password. Notice that this field can only be written, not read.
first_name	string	-	User's first name.
last_name	string	-	User's last name.
email	string (e-mail formatted)	required, write only	User's e-mail address.
groups	list of Groups	-	List of groups that the user is a member of.
resource_uri	string	<i>read only</i>	The REST URI that points to this resource.

Table A.7: Data fields of the Group REST resource.

Field	Type	Notes	Description
id	integer	<i>read only</i>	Unique identifier for the group. Automatically set.
name	string	required	The name of the group.
users	list of Users	-	List of users who are part of this group.
resource_uri	string	<i>read only</i>	The REST URI that points to this resource.