Secure Meeting Scheduler With Mobile Software Agents

Thomas Herlea

August 25, 2000
Abstract

Meeting scheduling is a negotiation process in which private information about the participants (their schedules) is transmitted, exchanged and processed in order to find a time when all participants can attend the meeting. Mobile Agents are a very promising programming paradigm, especially with the advent of mobile computing devices but introduce new security problems, for which solutions are still in research. This project presents the design of a meeting scheduler using mobile software agents which addresses the privacy and security needs of the user of the system, as well as of the administrators of the agent servers.
Copyright by K.U.Leuven

Without written permission of the promoters and the authors it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to K.U.Leuven, Departement Elektrotechniek - ESAT, Kardinaal Mercierlaan 94, B-3001 Heverlee (Belgium). Tel. +32-16-32 11 30, Fax. +32-16-32 19 86.

A written permission of the promotor is also required to use the methods, products, schematics and programs described in this work for industrial or commercial use, and for submitting this publication in scientific contests.
## Contents

1 Introduction .............................................. 5

2 Mobile Agents ............................................ 7
   2.1 Network Computing Paradigms ...................... 8
   2.1.1 Client-Server Paradigm .......................... 8
   2.1.2 Code-on-Demand Paradigm ....................... 8
   2.1.3 Mobile Agent Paradigm ......................... 9
   2.2 Mobile Agents: A Good Idea? ....................... 10
   2.3 Mobile Agent Platforms ......................... 13
   2.4 Aglets ........................................... 14

3 Security and Privacy ..................................... 17
   3.1 Desiderata ........................................ 17
   3.2 Cryptographic Tools .............................. 18
      3.2.1 Encryption .................................. 18
      3.2.2 Digital Signatures ......................... 20
      3.2.3 Certificates ................................ 22
   3.3 Agent specific issues ............................. 23
      3.3.1 Protecting the Host against the Agent .... 23
      3.3.2 Protecting the Agent against the Host ... 27
      3.3.3 Protecting the Agent in Transit ............ 29
      3.3.4 Protecting the Agent against Other Agents . 31
3.4 Proposed Security Goals .................................. 35

4 Negotiation Protocols and Architecture .......................... 37
  4.1 Schedule Representation .................................. 37
  4.2 Protocol Elements .............................................. 39
    4.2.1 Quick and Dirty .......................................... 39
    4.2.2 Hiding the Schedule From the Participants .............. 40
    4.2.3 Hiding the Times From the Intersector .................. 40
    4.2.4 Hiding the Contents From the Intersector .............. 41
    4.2.5 Hiding Non-solutions From the Participants .......... 42
    4.2.6 Hiding Both Time and Contents From the Intersector . 43
    4.2.7 Sharing the Secret ....................................... 44
    4.2.8 Bit by Bit ............................................... 45
  4.3 Proposed Negotiation Protocol ................................. 46
  4.4 Distributed Masking ......................................... 47
  4.5 Protocol Set .................................................. 49
  4.6 Scheduling Phases ............................................ 52
    4.6.1 Invitation Phase ......................................... 52
    4.6.2 Pre-negotiation Phase .................................. 54
    4.6.3 Negotiation Phase ....................................... 54
    4.6.4 Confirmation Phase ..................................... 55
  4.7 Architecture .................................................. 56
  4.8 User Interface ............................................... 61

5 Results .................................................................. 64
  5.1 Complexity ...................................................... 64
  5.2 Privacy .......................................................... 68

6 Conclusions .......................................................... 72

A Existing Meeting Schedulers ........................................ 76
  A.1 CDE Calendar ................................................... 76
CONTENTS

A.2 Yahoo! Calendar ............................................. 77
A.3 Proposals From the Literature ............................... 78

B Manuals ......................................................... 80
B.1 Installation Manual .......................................... 80
B.2 User’s Manual .................................................. 81
  B.2.1 Operating the Main Window .............................. 82
  B.2.2 Creating a New Invitation ............................... 83
  B.2.3 Answering to an Incoming Invitation .................. 85

C Code ............................................................ 87
Chapter 1

Introduction

Meetings play an important role in people’s social and economic lives. Besides casual meetings there are meetings scheduled in advance, that become part of the participants’ agendas. Most of the time new meetings are scheduled according to the unoccupied positions in the agendas.

Historically, meeting scheduling methods have evolved towards an increased specialization and efficiency. Initially each person would schedule their meetings by themselves, through direct communication with their counterparts. When the task became too time consuming for people to do it by themselves, the person’s secretary would take it over and assume the responsibility to manage the agenda and schedule new meetings on behalf of the employer. The task was delegated to an individual better specialized and more efficient than the actual participant to the meeting.

Today, when more and more of the routine tasks are handled by computers on behalf of their users, this task can be done faster, more reliable and cheaper than before. Secretaries themselves could use it in their work. But often, people have no other secretary than their laptop, personal digital assistant or cellular telephone.

These could benefit from the use of mobile agent technology. Mobile agents are programs that can migrate to other hosts during their execution.
CHAPTER 1. INTRODUCTION

Why they are appropriate for this class of computing devices and what their advantages are is described in length in chapter 2, “Mobile Agents”. In the same chapter we describe the flavor of mobile agents we chose for implementing the proof-of-concept program and we motivate the decision.

Mobile code introduces new security concerns compared to the case of static code. An extensive analysis of security issues specific to mobile agents is made in chapter 3, “Security and Privacy”. The cryptographic techniques that can help alleviate those problems are presented at the beginning of the chapter.

When negotiating meetings, the parties look up, communicate and process information about each other’s agendas trying to find a moment when all parties have the schedule free and can attend the meeting. Due to the private nature of a person’s schedule, as little as possible should be revealed to any other party during negotiation, besides the time of the successful scheduling of the meeting.

One goal of this project is to design the system in such a way that it satisfies privacy requirements better than the analyzed schedulers. In chapter 4, “Negotiation Protocols and Architecture” we describe the approach we took for satisfying this goal.

Chapter 5, “Results” provides numerical estimates of the complexity of communication and the degree to which privacy is protected by the system.

The last chapter, “Conclusions” evaluates the degree in which the goals were achieved and indicates directions of possible future work.
Chapter 2

Mobile Agents

According to Merriam-Webster’s Collegiate Dictionary, an “agent” is “one who is authorized to act for or in the place of another”. A software agent is a computer program that acts as a user’s agent.

What differentiates a software agent from a simple software program? Yoav Shoham, presents his own view in Chapter 13 of [2]:

In this view, therefore, agenthood is in the mind of the programmer. What makes any hardware or software an agent is precisely the fact that one has chosen to analyze and control it in these mental terms.

Mobile software agents are the class of software agents whose execution is not confined to the computer on which they were started but can migrate from one computer to another during their execution. “Created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution”, say the authors of [9]. Mobile agents actually require a program to run on each computer they visit, to provide them with a uniform execution environment and access to the local resources.
2.1 Network Computing Paradigms

Danny B. Lange, author of [7] says:

Our experience shows us that mobile agents provide a very powerful uniform paradigm for network computing. Mobile agents can revolutionize your design and development of distributed systems.

For putting this affirmation into perspective, we will describe the three programming paradigms for network computing. In all cases we have resources (e.g., databases) that have to be processed by a program possessing know-how. The place in the network where this program runs is the processor.

2.1.1 Client-Server Paradigm

The server has the resources, the know-how and is the processor. It advertises its services and clients that are interested in accessing the resources must use the server’s services. The client only needs enough “intelligence” to decide which services to request.

![Figure 2.1: Client-Server Paradigm](image)

Example client-server technologies are remote procedure calling (RPC), object request brokers (CORBA), Java remote method invocation (RMI).

2.1.2 Code-on-Demand Paradigm

With this paradigm the client is the processor. The resources are local. When the client senses the lack of know-how to process them it appeals to
a server which provides it with the necessary know-how by downloading the respective code.

![Code-on-Demand Paradigm](image)

**Figure 2.2: Code-on-Demand Paradigm**

An example code-on-demand technology are Java applets.

### 2.1.3 Mobile Agent Paradigm

“A key characteristic of the mobile agent paradigm is that any host in the network is allowed a high degree of flexibility to possess any mixture of know-how, resources and processors.” [7]

Know-how is allowed to move around in the network of processors, even at its own initiative.

![Mobile agent Paradigm](image)

**Figure 2.3: Mobile agent Paradigm**
“If you compare these three paradigms, you will see a chronological trend towards greater flexibility. The client and the server have merged to become a host.” [7]

### 2.2 Mobile Agents: A Good Idea?

Seven reasons are mentioned in [9] in favor of adopting mobile agents. They describe the advantages that the usage of mobile agents sometimes brings along.

**They reduce the network load**

This can be achieved by the agents’ ability to transform remote communication into local communication. In the case of negotiation especially, it is more the number of messages rather than the amount of transferred data that is reduced (see Figure 2.4).

![Diagram showing the comparison between RPC-based approach and Mobile Agent-based Approach](image.png)

*Figure 2.4: Mobile Agents Reduce Network Load*

**They overcome network latency**

Mostly in the case of controlling real-time systems such as manufacturing robots, network latency causes a decrease in the system’s performances. Mobile agents, by transferring control to a computer that is not as affected
by network latency help overcome this inconvenience. The same considerations apply to interactive applications in which long response times trigger a degradation of the user experience.

**They encapsulate protocols**

In the classical approach “each host involved in an interaction owns the code that implements the protocols needed to properly code outgoing data and interpret incoming data.” [9] When the protocols need to be upgraded on a large scale it is very difficult to synchronize all upgrades and protocols often become legacy problems. Mobile agents can carry with them the code that implements a protocol and can establish “channels” based on the most recent protocol.

**They execute asynchronously and autonomously**

“Mobile devices often rely on expensive or fragile network connections. Tasks requiring a continuously open connection between a mobile device and a fixed network are probably not economically or technically feasible.” [9] But mobile agents can take off from a mobile device which can then disconnect from the network, perform the task autonomously and deliver the result at a later time when the mobile device reconnects for a short time to the network, as seen in figure 2.5.

**They adapt dynamically**

“Mobile agents can sense their execution environment and react autonomously to changes.” [9] This means that multiple collaborating agents can distribute themselves in the network in a manner that takes advantage at best of the execution capabilities of these hosts.
They are naturally heterogenous

Mobile agents provide an abstraction level above the hardware, operating system and programming language level. They depend only on their execution environment and therefore “they provide optimal conditions for seamless system integration.” [9]

They are robust and fault tolerant

“Mobile agents’ ability to react dynamically to unfavorable situations and events makes it easier to build robust and fault-tolerant distributed systems.” [9] Not being tied to a specific host during their execution, mobile agents can leave a host or a network that has problems, without abandoning their task.

There have also been other efforts to assess whether Mobile Agents “are a good idea”. In 1995 IBM de-classified an internal report for use in its strategic decision making in which mobile agents were compared to “alternate methods of achieving the same function.” [5]

The alternatives taken into consideration were: messaging, simple datagrams, sockets, remote procedure call and conversations. Several proposed application areas and claims of agent superiority versus alternate methods...
were analyzed and assessed. The main questions that the use of agents raise were identified as efficiency, flexibility and security [5].

Their conclusions stated:

With one rather narrow exception, there is nothing that can be done with mobile agents that cannot also be done with other means. This exception is remote real-time control [...] While none of the individual advantages of mobile agents is overwhelmingly strong, we believe that the aggregate advantage of mobile agents is overwhelmingly strong.

2.3 Mobile Agent Platforms

Research on mobile agents has been going on for some time in the academic and industrial research centers. Nowadays there are numerous mobile agent platforms available. The best centralized list found on the web was “The Mobile Agent List.” [16]

It is a list maintained by the agent platform developers themselves and it lists no less than 72 system names. Out of these, 58 have an extensive profile including web page of the platform, organization, what standards it implements, technical details, available literature and licensing terms.

As for development tools, a web site called “Agent Construction Tools” [14] was found, where 60 such tools are listed. Generally each agent platform is available with its set of agent construction tools.

The entries are categorized as either commercially available products or academic and research projects. Descriptions are provided and the programming language is listed for each agent construction tool.

The range of programming languages used in today’s mobile agent systems includes: Java, Lisp, C++, Tcl, Python, Scheme.

The number and variety of mobile agent systems indicates that there is much ongoing research and interest in the mobile agent paradigm and seems
to indicate that agents are not only a temporary trend. To promote interoperability some aspects of mobile agent technology have been standardized by the Object Management Group (OMG) in the standard called “Mobile Agent System Interoperability Facility” (MASIF).

MASIF addresses the interfaces between agent systems. It standardizes the following four areas:

- Agent management (agent life cycle)
- Agent transfer
- Agent and agent system names (standard way to identify agents and agent systems) and
- Agent system type and location syntax (agents can only migrate if the destination system type can support the agent)

2.4 Aglets

Apparently IBM decided that mobile agents are enough of a “good idea” (see [5]) to give it a try themselves and in 1996 IBM’s Tokyo Research Laboratory created “Aglets”, one of the first mobile agent platforms written in Java. Programmers can download the Aglets Software Development Kit from the web [15].

The package contains the Java classes necessary for writing aglets, an aglet server which also acts as an aglet viewer, example aglets, classes which implement useful design patterns and the HTML documentation of the API, installation, configuration and about the examples.

Being implemented in Java, Aglets benefit of the advantages of this modern programming language:

- Platform independence
• Object orientation
• Multithreading
• Exception handling
• Efforts for enforcing secure execution
• Dynamic class loading
• Object serialization
• Reflection

The aglets run on top of an aglet server program, in a so-called aglet context. A lot of protection measures can be implemented in the server’s configuration files. The security model is similar to Java 1.2’s model, with security policy files.

Aglets have captured mindshare, there is a large programmer base worldwide, a dedicated mailing list, tutorials and web pages. Most technical support comes from the aglet programmers community itself. Part of this success is due to the fact that Aglets can be downloaded for free. It is expected that Aglets will have even bigger success since IBM has recently approved the open sourcing of Aglets.

There are also some drawbacks related to Aglets.

Aglets must provide workarounds for some of Java’s security weaknesses. How they do it is described in subsection 3.3.4.

Aglets must provide a workaround even for the following security strength of Java: the inaccessibility of the execution stack and program counter to the program itself. This constrains Aglets to weak migration, that is, they do not transport their full state to the destination. Aglet programmers have to implement their own finite state machine with internal variables so that the aglet knows what state to resume execution from after a migration.
As of the year 2000 Aglets still do not to Java 1.2. That is about to change in the future because it is the first goal to be achieved since Aglets turned into Open Source Software. Until then Aglets will not benefit of some of the advantages of Java 1.2 over Java 1.1.

Taking in consideration all these aspects and after comparing Aglets to several other mobile agent platforms we have decided that Aglets are the best candidate for implementing the proof of concept prototype of the project.
Chapter 3

Security and Privacy

3.1 Desiderata

When people use computers, they sometimes need one or the other of following conditions related to security and privacy to be respected:

Confidentiality Data is not illegitimately read

Integrity Data is not illegitimately modified or deleted

Availability Services to which a user is entitled are not illegitimately disrupted

Anonymity The impossibility to link a communication exchange to a real life entity

Authentication The possibility to link a communication exchange to a real life entity

Non-repudiation The impossibility to deny that a communication exchange took place

The use of cryptography for protection of secrecy of information is as old as writing itself. Cryptography is the science of designing ciphers. It has
CHAPTER 3. SECURITY AND PRIVACY

solid mathematical foundations and its range of applications stretches from e-business and health care to military applications.

3.2 Cryptographic Tools

3.2.1 Encryption

Encryption serves for protecting the confidentiality of information. In [10] the main concepts and terms are defined for encryption and decryption:

The basic idea consists of applying a 'complicated' transformation to the information to be protected. When the sender (usually called Alice in cryptography) wants to send a message to the recipient (Bob), she will apply to the plaintext $P$ the mathematical transformation $E()$. This transformation $E()$ is called encryption algorithm; the result of this transformation is called the ciphertext or $C = E(P)$. Bob will decrypt $C$ by applying the inverse transformation $D = E^{-1}$; in this way he recovers $P$ or $P = D(C)$. For a secure algorithm $E$, the ciphertext does not make sense to outsiders: Eve, who is tapping the connection, can obtain $C$, but she cannot obtain (partial information on) the corresponding plaintext $P$.

It works only when $E$ and $D$ can be kept secret. Keeping a secret algorithm for every pair of Alices and Bobs is not a scalable solution. A good solution is to introduce into the encryption algorithm a secret parameter $K$, then the algorithm does not have to be kept secret itself. $K$ is called the key. There has to be a corresponding key $K*$ for the decryption algorithm.

When $K = K*$ the type of algorithms is called symmetric ciphers (see figure 3.1). In the case of asymmetric ciphers, the two keys are always different; moreover, it should be difficult to compute one key from the other. This simplifies key management.
With symmetric keys, the problem of securely transmitting a message was reduced to the problem of securely transmitting a key. In the case of public key cryptography the encryption key can be made public by Bob (\textit{public key}) and every Alice can use it when sending encrypted messages to Bob. Note that decryption can only be performed with the decryption key (\textit{private key}), which Alice does not have and can not deduce from the public key, even if she analyses plaintext/ciphertext pairs obtained with the public key. This guarantees that encrypted messages can only be decrypted by Bob (see figure 3.2).

Public-key cryptography relies on trapdoor one-way functions, which are one-way functions with the property that some additional information, the so-called trapdoor, makes the function invertible. A one-way function $f : X \mapsto Y$ is a function with the properties that:

- $f(x)$ is easy to compute for all $x \in X$
• given \( y \in Y \) it is computationally infeasible to find an \( x \in X \) satisfying 
\[ f(x) = y. \]

Good candidates are found in mathematics: modular exponentiation and the group of elliptic curves over a finite field.

The main disadvantages of public-key algorithms with respect to conventional (symmetric-key) algorithms are the larger keys needed for obtaining the same degree of security and the slow performance.

The best of both worlds is combined in the approach presented in [10]:

Because of the large difference in performance and the larger block length (which influences error propagation), one always employs *hybrid* systems: the public-key encryption scheme is used to distribute a secret key, which is then used in a fast conventional algorithm.

### 3.2.2 Digital Signatures

When information is transmitted, sometimes it is desired to have some form of information authentication. This includes *data origin authentication* (who has originated the information) and *data integrity* (has the information been modified).

Encryption does not always guarantee authenticity. When two parties share the same key for encryption each can produce encrypted messages and claim they were produced by the other party. What is needed is a mechanism that offers protection between two mutually distrustful parties (which is often the case in commercial relationships), an electronic equivalent of a manual signature. In cryptographic terms this is called a *digital signature*.

Data integrity is achieved by reliably communicating to the other party some redundancy information. The authenticity of the information is replaced by the protection of a short string, which is a “fingerprint” of the information. Such a “fingerprint” is computed as a *hash* result. If the hash
can be communicated over a secure channel then the hash algorithm can be public and the hash is called Manipulation Detection Code (MDC). If the hash itself must be conveyed over insecure channels, the hash algorithm must have a secret parameter (the key) and the hash is called Message Authentication Code (MAC). The receiver recomputes the hash based on the received message and compares it to the MDC or MAC. If they match, the message is authentic.

A good hash ("collision resistant hash") has to satisfy the following three conditions:

- It should be hard to find an input given a hash result (preimage resistance)
- It should be hard to find a second input with the same hash result as a given input (second preimage resistance)
- It should be hard to find two different inputs with the same hash result (collision resistance)

A good design rule for MAC algorithms is that the MAC should be a complex function of every bit of the message and every bit of the key.

What digital signatures are is explained in [10]:

A digital signature is the electronic equivalent of a manual signature on a document. It provides a strong binding between the document and a person and in case of a dispute a third party can decide whether or not a signature is valid. Of course a digital signature will not bind a person and a document, but will bind a key and a document.

Additional measures are required to couple a person to a key, as can be seen in section 3.2.3.

The most elegant and efficient constructions for digital signatures rely on public-key cryptography. A message signed with Bob’s private key can be
verified by anyone who knows his public key. This authenticates that it was Bob who signed the message because no one but he knows the secret key.

Bob's message must contain verifiable redundancy, otherwise any random string can be claimed to be coming from Bob. Verifying it with Bob's public key "proves" that the signature was applied on an original message which is a random string itself. This may or may not be useful to the attacker. This redundancy can for example be a MDC appended to the message.

When signing very long messages which do not have to be secret, decrypting the whole message with the secret key is impractical. For Bob it is enough to sign the MDC of the message and append it to the plaintext message.

### 3.2.3 Certificates

For binding a key to a person, real-world authorities are needed. This task has become so important in the New Economy that the Public Key Infrastructure (PKI) is an entire industry.

Most of the time people can not verify the binding directly and then binding a key to an entity is done with certificates which basically state "Key K belongs to entity X." Certificates are issued by Certification Authorities (CA) which first verify the binding and must be trusted by their users to do so. How can one know that a certificate was issued by a trusted CA if it is not received directly from the CA? The CA digitally signs the certificate and the entity can distribute the certificate itself from that point on.

But how can one verify that the signature belongs to the CA? One can retrieve the CA's certificate from another CA one trusts. There are basically two approaches. First, a pyramid of CAs, each providing certificates for the ones on the level below it. Second, mutual certification in a web of equally ranked CAs.
3.3 Agent specific issues

3.3.1 Protecting the Host against the Agent

Problem description

Computers were built with the implicit assumption that the programs that are launched into execution are in concordance with the intentions of the owner of the computer. This is not always true because of two reasons. One is that the user who launches the program might not be the owner of the computer and might have intentions opposite to the owner of the computer. This aspect has been exploited by malicious users. The other reason is that due to the complexity of today’s programs, the user can not always know in advance what exactly a program will do, even if he is the owner of the computer that would run the program. This aspect has been exploited by trojan horses, in privacy invasion by browser cookies and by the online registration wizards of some programs.

Mobile software agents are pieces of foreign code whose nature is unknown in advance and fall in the second category of potential threats. If the originators of the agents remain unknown or hard to track down it is not possible to prosecute them for any malicious actions their agents might take.

Malicious agents may try to:

- Read confidential data from the host they run on and disclose it to an unwanted party

- Alter or delete data stored on the host

- Perform a denial of service attack, that is, deliberately using a resource so heavily that service to other legitimate users of that service is disrupted
Aglet solutions

The defensive measures must be taken at least at host level, but the aglets may voluntarily take some measures, too.

First of all, agents must be identifiable and linkable to real life entities. Aglets uses the term principal for identifying the entities involved in aglets exploitation. Some principals are the aglet, the aglet’s owner (the entity that launched the aglet), the aglet’s manufacturer, the owners and the manufacturers of the contexts.

This is how [6] describes the Aglets approach to the identification issue:

Before the owner can launch an aglet, the context authenticates the owner as a registered user. Within the creation request, the aglet owner defines security preferences to be applied on the aglet. When the context instantiates the aglet from the corresponding Java class it might include information about the manufacturer, owner and the aglet’s original context that is, about itself.

The same source states that “every aglet has an identifier that is unique over its life time and independent of the context it is executing in” and that the principals “can be properly identified, for example by their public key and with the help of a suitable certification infrastructure.” Real life entities are linked to an aglet by the fact that they have to sign the aglet before the aglet is executed. With these means principals can be made responsible for the misbehavior of their aglets and their aglets can be refused in the future (post-factum measures).

Agents should be prevented from doing harm in the first place. Considering that the harm they can do is proportional to the capabilities they have, agents should be granted just enough rights to be able to do their job. Minimal capabilities is a good security policy.

Aglets use Java’s security features like the bytecode verifier, the sandbox model and security managers. The bytecode verifier ensures that the
 CHAPTER 3. SECURITY AND PRIVACY

bytecode is well formed, does not contain illegal instructions and does not violate Java’s type system. The sandbox provides the aglet with an isolated execution environment so that the aglet can not interfere with other execution environments. The security manager monitors the aglet’s requests for resources and grants or refuses them. Security managers are implementing the security policies that govern the activity of the aglets.

In the paper that describes the security model for Aglets the authors describe the Aglets security architecture [6]:

The security architecture implements the security model by providing a set of components and their interfaces. (...) we introduce two components of the aglets security architecture: the policy database of the context master and the security preferences of the aglet owner. Because both context master and aglet owner have their own specific interests concerning what an aglet should be able to do both may want to restrict its capabilities. Such restrictions might apply to either accessing the local resources of a context or offering services to other aglets. The policy database and security preferences therefore constitute powerful elements in introducing security into the Aglets Workbench.

Following resource types are currently considered:

- File (all or just some files in the local file system)
- Net (network access by protocol, host and port)
- Awt (the local window system)
- System (any kind of system resources, such as memory and CPUs)
- QoP (quality of protection)
- Context (resources of the context)
• Aglet (resources of the Aglet)

Specific access to each type of resource is granted or denied in the policy
data base on a set of principals basis. Each entry has the syntax:

<label> : <set of principals> -> <privileges>

Following example will show how an entry is interpreted:

TRUSTED:

aglet=com.ibm.examples.HelloWorld AND owner=JohnDoe ->
  File /tmp read, write
  Net TCP bavaria.utcluj.ro 22 accept
  AWT Top_level_windows 3

This states that aglets created from the class com.ibm.examples.HelloWorld
sent by “JohnDoe” are allowed to read and write files in the directory /tmp,
establish TCP connections to the FTP port of host bavaria.utcluj.ro and
can open up to three top level windows.

The aglet owner also has the possibility to request the restriction of the
resources given to an aglet. It can be made out of politeness or for limiting
the owner’s liability in case anything goes wrong on the destination host.
Since the context has the power to honor or not the request, the owner’s
security preferences are mere indications. In any case, security preferences
can not be used to enlarge the set of rights that a context grants to an aglet.

A security preference might look like

context=atp://bavaria.utcluj.ro:4444/ ->
  ITINERARY set
  MESSAGE welcome subscribe

This requests that the aglet’s itinerary is not changed at other contexts than
atp://bavaria.utcluj.ro:4444/ (the “atp” comes from “Agent Transfer
Protocol”) and that the aglet is allowed to subscribe to messages of kind
“welcome” when it is in that aglet context.
3.3.2 Protecting the Agent against the Host

Problem description

“Can a program actively protect itself against its execution environment that tries to divert the intended execution towards a malicious goal?” is the question posed by Thomas Sander and Christian F. Tschudin in chapter “Protecting Mobile Agents Against Malicious Hosts” of [13]. They also state that:

We firmly believe that in many cases fully software based cryptographic solutions exist for protecting mobile agents against malicious hosts. This belief contradicts the folklore saying that because the host must execute and thus is in possession of an agent’s code, the agent is to the host’s full mercy.

A malicious host can, according to Fritz Hohl in [13]:

- Spy out data
- Spy out code
- Spy out control flow
- Manipulate data
- Manipulate code
- Manipulate control flow
- Execute code incorrectly
- Masquerade the host
- Deny service
- Spy out interaction with other agents
• Manipulate the interaction with other agents

• Return wrong results to system calls

• Delay or block the departure of an agent from the host

• Copy the agent and misuse the copy for illegitimate purposes

The owner of the agent has no control over the agent when it is on a remote host. The problems are very serious when the agent carries sensitive information like credit card numbers, personal information or e-cash or when the agent is used to perform sensitive tasks like negotiations, acquisitions and contract closing on behalf of the owner.

Solutions

Some partial solutions to the problem are:

Using trusted hosts only This hurts the concept of an open system where new servers can join the system easily and relies on the trust model to be effective, which is not easy to enforce.

Detection of tampering If it’s in the host’s interest to be trusted, detection of tampering techniques are an incentive for the host to refrain from any tampering. This does not, however, reverse the effects of the tampering.

Making tampering infeasible Obfuscation of code and data tries to make tampering difficult by making the program illegible. So far, this is a race of arms, each new obfuscating technique being immediately paralleled by countermeasures.

Protecting the application as a whole This approach distributes the functionality of an application over several hosts and uses fault tolerance techniques to ensure proper functionality as a whole, treating tampering as faults.
Provably secure prevention Computing is done with encrypted programs or clueless agents. These ideas are still in research and often are only applicable to restricted areas of remote computation.

The strongest approach is “provably secure prevention”. Computing with encrypted programs is motivated by the observation that “if we can execute encrypted programs without decrypting them, we automatically have a) code privacy and b) code integrity in the sense that specific tampering is not possible, attacks by a malicious host would then be reduced to actions at the surface of the mobile agent: denial of service, random modifications of the program or of its output, as well as replay attacks.” (Thomas Sander and Christian F. Tschudin in [13])

3.3.3 Protecting the Agent in Transit

Problem description

The Internet was designed for reliability rather than security. Since the Internet is most likely the navigation medium of an agent at least on sections of its trip, it is exposed to attacks during its transit through network sections which are under the control of hostile parties.

The possible attacks include:

- Traffic analysis, revealing which parties are communicating
- Eavesdropping, spying on data in transit
- Replay, retransmitting a copy of an agent for illegitimate purposes
- Masquerade, pretending to be a different agent host

The problems are generally the same as for transmitting other types of data through the network. Therefore the same measures at transport and application layer will be applied for Aglets, too.
Solutions

The aglets servers establish two by two Virtual Private Networks (VPNs). These are logical networks that connect computers by a secure channel using insecure segments of public networks (see figure 3.3).

![Virtual Private Network](image)

Figure 3.3: Virtual Private Network over public Internet

The VPNs make use of cryptographic techniques to defend the channel from eavesdropping, replay and masquerading attacks. They make use of local Internet connections and do not require long distance calls or leased lines. VPNs ensure host authentication in the VPN setup phase and non-repudiation by signing the transmitted packets. The payload is usually encapsulated in the data formats of the networks it passes through, a process called “tunneling”, but this is transparent to the user.

If not present at the transport layer, the VPN can be emulated at agent server level. The aglet server that comes with the Aglets package has the possibility to protect communications with another server based on “shared secrets” between the two servers. But the ASDK allows for custom aglet servers to be built, which can implement VPNs in whatever means they want.

Against the traffic analysis attack the best defense known is anonymous communication [3]. This is a set of measures to ensure that it is hard for a party that is not involved in a communication to know of the existence of the communication. It is achieved by making it hard to extract from all the
network traffic the exchanges implied in the communication. This is usually done by adding hops to rerouters to the normal itinerary of the traffic.

“Chaum mix” rerouters hide the correspondence between items in their input and those in their subsequent output by delaying, reordering and padding the packets that pass through them. Onion routers encapsulate payload in several envelopes of metadata. Each router is able to decrypt the outer envelope and use the metadata to transmit the rest of the “onion” to the next onion router indicated in the envelope. These rerouters may generate constant traffic and make sure that disruptions in the traffic are propagated uniformly in the entire network, so that an observer can not gain information about the routes by analyzing traffic volume (e.g., PipeNet). Crowds consist of groups of computers that perform rerouting services for each other, such that traffic outside the crowd can not be associated with a particular user in the crowd.

Anonymous communication can be implemented at network level, at agent server level or at agents level.

3.3.4 Protecting the Agent against Other Agents

Problem description

When interacting with other agents, an agent is supposed to maintain its integrity and ability to perform its task correctly. This implies the ability to identify other agents it interacts with and communicate with them in a secure way.

Possible attacks are:

- Spying (reading another agent’s confidential data)
- Eavesdropping (reading another agent’s messages not addressed to itself)
- Alteration (changing another agent’s data or normal behavior)
• Masquerade (pretending to be another agent)
• Repudiation (denying that a communication exchange took place)

Since agents sometimes carry secrets, other agents might try on behalf of their owners to learn these secrets. They might try to change the behavior or state of other agents to obtain benefits for their owners. Sometimes the owners might try to deny an exchange done by their agent on their behalf.

**Solutions**

Measures can be taken at two levels. The agent host should ensure proper isolation between the agents (to prevent alteration and spying) and implement secure communication channels for point-to-point messages (channels that do not alter the data and prevent it from being intercepted by any third party, to prevent eavesdropping).

Aglets circumvent some security weaknesses of the Java language itself. Java has no protection of references. Any object can access another object's public interface if it has a reference to the object. We certainly want to prevent other aglets from accessing the maintenance methods of an aglet (e.g., the creation/destruction ones). Aglets solves this problem by enforcing the use of the proxy design pattern in interactions between aglets [8]. This limits the visibility of the maintenance interface to the server only.

Java has no ownership of references. As long as another aglet has a reference to an aglet the Java garbage collector will not destroy the aglet object. This would open the possibility for an aglet to keep another aglet alive against the owner’s will. Aglets takes every measure to ensure that aglets do not get references to other aglets. Even if an aglet tries to explicitly send another aglet a reference to itself by means of a message a security exception is thrown.

Other features of security arise even if the communication support is secure. The same issues that exist at transport layer appear in legal inter-agent
communication, too. One of the papers that address this issue introduces Secure KQML [11].

The Knowledge Query and Manipulation Language was developed at the University of Maryland Baltimore County as part of the Knowledge Sharing Effort, a project of the Advanced Research Projects Agency (ARPA). It is a high-level language and a protocol intended for the run-time exchange of knowledge between intelligent systems. It views this exchange as having three layers:

- The content layer (the payload of the exchange)
- The message layer (speech acts, protocols and conversations)
- The communication layer (identities of the sender and receiver, unique identifier associated to the exchange)

The authors of [11] have designed an extension of KQML called secure KQML with support for following security capabilities:

**Authentication of principals** Agents should be capable of proving their identities to other agents and verifying the identity of other agents

**Preservation of message integrity** Agents should be able to detect intentional or accidental corruption of messages

**Protection of privacy** The architecture should provide facilities for agents to exchange confidential data

**Detection of message duplication or replay** A rogue agent may record a legitimate conversation and later play it back to disguise its identity. Agents should be able to detect and prevent such playback security attacks

**Non-repudiation of messages** An agent should be accountable for the messages that it has sent or received, i.e., they should not be able to deny having sent or received a message
**Prevention of message hijacking** A rogue agent should not be able to extract the authentication information from an authenticated message and use it to masquerade as a legitimate agent.

Secure KQML adds keyword/value pairs to standard KQML messages and introduces some new performatives necessary for the security architecture. Since KQML tries to develop secure protocols, it assumes that the used cryptographic primitives are perfect. If a primitive can be successfully attacked, secure KQML may be successfully attacked, too.

When an agent wants to authenticate itself to another agent, or when it is requested to do so, the handshake protocol from figure 3.4 is used.

![Handshake Protocol Diagram](image)

**Figure 3.4: The self authentication protocol**

When the integrity of the message is essential, message digests are used. They are appended to the message in the special parameter `:auth-digest (<digest-type> <encrypted-digest>)`. Message digests were explained in section 3.2.2.

Privacy is protected by encrypting the message contents. There is a special performative, `auth-private`, which has as contents an encrypted message, which could be a KQML performative as well. Secure KQML supports two types of keys: master keys, which are long-term, and session keys, which are used in only one communication session. Since only the agents which are supposed to understand the contents have the key, secrecy is ensured.
Replay attacks are prevented by using a message ID system to aggregate individual messages into meaningful conversations. The standard KQML system using the parameters :reply-with and :in-reply-to is replaced by the parameter :auth-msg-id (<msg-id> <encrypted-msg-id>). This has as value a list which contains the message ID and the encrypted ID of the next message. An eavesdropper will not be able to decrypt the ID of the next message and any other message ID will be rejected as an attack.

Non-repudiation comes from the fact that in some cases producing a signed digest ties the sender to the sent message. That case is when the key used for signing is the private part of an asymmetric key. Then, any message that contained a signed digest would be impossible to repudiate.

Hijacking of the authentication information is almost impossible due to the signature. If the message is changed, the signature will not verify anymore and the message will be rejected. Without the encryption key it will be impossible to generate a new signature.

Even when agent interaction is secure and secure communication protocols are used to protect the message contents, in some cases one party wants the other party to learn as little as needed about its private information. This is particularly true about negotiations. Special negotiation protocols are then used, which try to reveal only the information necessary to bring the negotiation to a successful conclusion. This project brings some contributions in this area, presented in chapter 4.

3.4 Proposed Security Goals

Considering all available information so far, following security goals were established for the meeting scheduler:

- Allow the negotiation to be performed on an arbitrary schedule, which may be derived from a user’s real schedule
• Perform parties authentication

• Use a negotiation protocol which reveals minimal information besides what is needed for the success of the negotiation
Chapter 4

Negotiation Protocols and Architecture

4.1 Schedule Representation

The first question was to represent the schedule in such a way that it

- Captures all the information needed for negotiation
- Makes it easy to perform the negotiation
- Makes it easy to perform the negotiation even on the protected form

It was found that it is possible to represent the schedule by a data structure that reduces the representation of any time interval to one bit. The participants must know before creating the representation what the proposed meeting length is. Then they can determine the set of all possible time intervals when they can attend the meeting. The data structure will represent only the starting time of each of these intervals. In order to make the resulting set discrete the participants must use the same granularity (e.g., hours, minutes). Then, each beginning of an hour (or minute) when a meeting can start will be represented by a 1 and each beginning of an hour (or minute)
when a meeting can not start will be represented by a 0 (see figure 4.1). These bits are called *time slots*.

The schedule which is converted into the bit string representation need not be the actual schedule of the user. This could contain only a subset of the free times (or a superset), allowing the user to lie about their schedule if they want and bringing an additional protection to the user's privacy: if the bit string is captured it contains data about a slightly different schedule than the user has.

The preferred method of protecting the schedule when it is transmitted to another party is bitwise applying the binary exclusive-or operation (⊕) with a secret mask. This allows for unprotected only parts of the schedule if needed and offers perfect information-theoretic confidentiality if the secret mask is a random string. It also has another important property. Equal time slots remain equal after masking with any value. They are called *identical slots* (see figure 4.2).
4.2 Protocol Elements

The protocol elements will be presented sequentially starting from the simplest protocol to the most secure. As the weaknesses of each protocol are pointed out, new elements are added for improving it.

We call the agent that represents a user in the negotiation “negotiator”.

4.2.1 Quick and Dirty

The easiest way to schedule a meeting is following:

1. Each negotiator broadcasts its schedule to the other negotiators

2. Each negotiator finds the earliest time slot when all participants can start the meeting (every schedule has 1 in that time slot) and that is the result

Whenever the negotiators themselves are trying to find the time slot where the meeting can be scheduled, the protocol is said to be using *distributed intersection*.

However, this way all negotiators learn any negotiator’s full schedule. A negotiator’s schedule should be hidden from the other negotiators.
4.2.2 Hiding the Schedule From the Participants

If a negotiator’s schedule doesn’t get to another negotiator, it is well enough protected. For that a trusted third party can be used, called *intersector* (see figure 4.3).

1. Each negotiator sends its schedule to the intersector

2. The intersector finds the earliest time slot when all participants can start the meeting (identical slots of value 1) and sends this result back to each negotiator

Still, this way, the system is vulnerable to a single point of failure, the intersector. If the intersector is malicious then it can misuse the full schedules of all agents because it knows them all. Therefore the schedules must be hidden from the intersector.

![Figure 4.3: Agents communicating with an intersector agent](image)

4.2.3 Hiding the Times From the Intersector

The schedules can be hidden from the intersector by submitting instead of the schedule, a permutation of the schedule, using the same permutation for all negotiators. This hides the real order of the time slots.

1. In the setup phase the negotiators agree on a permutation function, applicable to bit strings of the same length as the schedules

2. Each negotiator creates a permuted version of its schedule
3. Each negotiator submits the permuted version to the intersector

4. The intersector chooses a time slot when all participants can start the meeting (identical slots of value 1) and sends this result back to each negotiator

In this scenario a malicious intersector will not be able to reconstruct by itself the original schedules except for the cases when a schedule consists entirely of zeros or ones (then the original and the permutation are identical). But it is able to find out how busy or free each schedule is just by counting the zeros and ones. The contents of the time slots must be hidden, not the order.

4.2.4 Hiding the Contents From the Intersector

The contents of the schedules can be hidden by using a masking operation instead of the permutation.

1. In the setup phase the negotiators agree on a random bit mask of the same length as the schedules

2. Each negotiator creates a masked version of its schedule by performing \( \oplus \) between its schedule and the bit mask

3. Each negotiator submits the masked version to the intersector

4. The intersector doesn’t know whether it has to look for identical slots of 1 or identical slots of 0 so it returns the entire set of identical slots (be they 0 or 1)

5. Each negotiator looks in the set of identical slots for the earliest time slot for which it has in its own, unmasked schedule the value 1. Since all other negotiators also have 1 in that slot, that is the result
CHAPTER 4. NEGOTIATION PROTOCOLS AND ARCHITECTURE

The intersector is confused by the masking, but not the negotiators. They can use the returned set of identical slots for deducing the schedules of the others for all slots in the returned set. For these slots the others have the same value as the agent itself. For very empty or very busy schedules this reveals a lot of information to each negotiator about the other schedules. The negotiators should receive as little as possible from the set computed by the intersector.

4.2.5 Hiding Non-solutions From the Participants

The conclusion was reached that the intersector should not return the whole set of identical slots if we do not want the negotiators to be able to deduce too many things about each other’s schedule. The intersector can be modified to prevent this problem by making it return the indices one by one and stopping when the negotiators have found a valid result.

1. In the setup phase the negotiators agree on a random bit mask of the same length as the schedules

2. Each negotiator creates a masked version of its schedule

3. Each negotiator submits the masked version to the intersector

4. The intersector computes the set of identical slots and returns one at random

5. Each negotiator checks whether for that time slot it has value 1. If yes, since all other negotiators also have 1 in that slot, that is the result

6. If not, each agent sends a query to the intersector, asking for another identical slot. When the intersector has received queries from all negotiators it returns another identical slot. The querying is repeated until a result is found or the set of identical slots is depleted
This method is said to use “queried intersection”, because the intersector is queried for identical slots.

The negotiators now get to know less about the identical slots because the process stops revealing information once the result is found. But now the intersector is able to tell whether the negotiation was successful or not by the fact that the querying stopped before the depletion of the set of indices. It knows the schedules of all negotiators for the identical slots which were returned. The intersector should be confused at least with respect to which time moments those slots represent.

4.2.6 Hiding Both Time and Contents From the Intersector

The intersector can be confused by combining the two techniques for hiding the schedules from the intersector: permutation and masking.

1. In the setup phase the negotiators agree on a permutation function, applicable to bit strings of the same length as the schedules and on a random bit mask of the same length as the schedules

2. Each negotiator creates a masked version of its schedule

3. Each negotiator creates a permuted version of the masked schedule

4. Each negotiator submits the permuted version to the intersector

5. The intersector computes the set of identical slots and returns one at random

6. Each negotiator computes the inverse permutation for that slot and checks whether for that time slot it has value 1. If yes, since all other negotiators also have 1 in that slot, that is the result
7. If not, each agent sends a query to the intersector, asking for another identical slot. When the intersector has received queries from all negotiators it returns another identical slot. The querying is repeated until a result is found or the set of identical slots is depleted.

Now the intersector can not tell more than the success or failure of the negotiation. Even if the protocol seems safe enough, it is time to worry about agents that collude in order to find out the secrets of the others. If a negotiator tells the intersector the mask and the permutation, the intersector is able to reverse the hiding for all negotiators' schedules. It should not be so easy for a small set of agents to collude. No negotiator should be in possession of the entire information needed for hiding the schedules.

4.2.7 Sharing the Secret

The masking lends itself to being computed in a distributed manner such that no negotiator knows the entire mask. Each negotiator must have an input into the mask. The particular way in which the distributed masking is done is described in section 4.4.

The permutation and the interaction with the intersector are done as in the previous protocol.

1. In the setup phase the negotiators agree on a permutation function, applicable to bit strings of the same length as the schedules

2. The masking is done in a distributed manner, as described in section 4.4

3. Each negotiator creates a permuted version of the masked schedule

4. Each negotiator submits the permuted version to the intersector

5. The intersector computes the set of identical slots and returns one at random
6. Each negotiator computes the inverse permutation for that slot and checks whether for that time slot it has value 1. If yes, since all other negotiators also have 1 in that slot, that is the result.

7. If not, each agent sends a query to the intersector, asking for another identical slot. When the intersector has received queries from all negotiators it returns another identical slot. The querying is repeated until a result is found or the set of identical slots is depleted.

Now the negotiator and the intersector can not collude to find out the complete schedules of the other negotiators. They still can figure out the schedules for the identical slots. The protocol from section 4.2.5 was introduced in order to hide from the negotiators part of the set of identical slots. A collusion between a negotiator and the intersector can annihilate this protection. The intersector can disclose the entire set of identical slots and the negotiator can provide their real value.

4.2.8 Bit by Bit

One way each negotiator can control the amount of information disclosed is to release one bit of the schedule at a time. This way, when the protocol stops it can be sure that none of the other bits has leaked out in whatever form.

For each bit of the schedule (in permuted order), until a result is found or the schedules are exhausted, do following:

1. Mask the bit in a distributed manner.

2. Submit the bit to intersector

3. If the intersector replies that all bits have the same value and the slot of the negotiator's own schedule is 1 a solution is found, otherwise continue with the next bit
Since the intersector's job is trivial in this case, the agents can perform it by themselves, thereby reducing the security risks even more. The entire section 4.3 is dedicated to describing this new protocol which is the main result of the project.

4.3 Proposed Negotiation Protocol

The proposed negotiation protocol has following characteristics:

1. Successive time slots are negotiated one at a time, therefore no supplementary information can be gained after a meeting has been scheduled

2. Each agent decides by itself if the meeting can be scheduled in the time slot and is not exposed to being manipulated

3. The schedule bits are never communicated to other parties. Only masked schedule bits leave the owner of the schedule

4. A distributed masking algorithm is used, which prevents all other agents from knowing the value of an agent's schedule bit during and after the masking but brings the bit of each agents

5. Identical slots of 0, for which no meeting can be scheduled introduce a breach in the privacy of the participants. The analysis of how large the breach is can be found in chapter 5

Compared to the last described protocol (see section 4.2.8) the intersector is not used and the permutation phase is unnecessary.

The algorithm is the following, for each agent:

For each bit in the schedule do:
{
    1. Participate in the distributed masking process
    2. Broadcast to the other agents all masked bits that
finished step 1 at this agent
3. Receive all masked bits broadcasted by the other agents
4. If all masked bits have the same value then:
   { 
   4.1 If the corresponding bit in this agent’s
       schedule is "1" then:
       { 
       4.1.1 remember result 
       4.1.2 stop 
       }}

4.4 Distributed Masking

The global mask must be a shared secret. Only if all agents cooperate should
it be possible to find out the global mask and retrieve the original information.

To accomplish this each negotiator must have an input into the global
mask and keep its contribution secret. Each agent will chooses a secret mask
that is called “partial mask”. The global mask will be the bitwise exclusive-or
of the partial masks.

How can be the global mask be used if no agent is allowed to know it?
The masking process must be distributed itself. The data must be brought
to the partial mask so that the partial mask never leaves the negotiator (see
figure 4.4).

This visit must be organized such that:

• Schedule bits visit all negotiators

• Schedule bits visit each negotiator exactly once

• Schedule bits can not be identified by their originating agents after they
  were masked, otherwise the originating agent would learn the global
  mask (it would know $X$ and $X \oplus M$)
Figure 4.4: Example itineraries for the schedules

The visit of all negotiators exactly once can be achieved by having the originating negotiator attach to the schedule bits a list of negotiators which shrinks as the schedule bits visit the other negotiators. After masking with its own partial mask, each negotiator would chose a successor from the list, remove it from the list and forward it the masked schedule. When the list is empty the schedule has been masked by all negotiators with their partial mask, which amounts to saying it was masked with the global mask (because $\oplus$ is associative).

The first to mask the schedule bits is always the originator negotiator, in order to ensure that the actual schedule is not known by any other entity. It also removes itself and the agent to which it sends the masked schedule bits from the list that initially contains all the agents.

The fact that the successor is chosen from the list by different agents and not by the originator makes it impossible (in most cases) for the originator and the last negotiator visited by the schedule bits to collude and find the global mask (together they know $X$ and $X \oplus M$).

Collusion is still possible between the negotiators before and after a particular negotiator in the sequence in order to find the middleman’s partial mask (since together they know the input and output of the middleman). Taking care whom the masked schedule is forwarded to is a solution accessible to the middleman. It involves trust in the next negotiator and is not always available (for the but-last negotiator in the sequence). Usage of an
anonymizer agent that makes a random forwarding without applying any processing is a better solution.

If an anonymizer agent is used, care has to be taken that this doesn’t get to know enough information to pose a security threat. In particular, a schedule received from the anonymizer agent should not be returned to the anonymizer agent after masking with the partial mask.

4.5 Protocol Set

The negotiation protocol described in section 4.3 involves a large number of communication exchanges between the negotiators. Using mobile agents partially alleviates this problem. Still, payload of one bit is wrapped in a contents level envelope, which is wrapped in a message level envelope, which is wrapped in the successive transport level envelopes every time it is transmitted between two negotiators. This introduces delays in the finalizing of the negotiation and a heavy resource usage.

Most of the time users prefer a tradeoff between security and performance. A larger set of protocols of different degrees of security and resource usage is available for accommodating the preferences of the users. The generator parameters are:

1. Intersection
   - using an intersector agent (denoted QI, from “queried intersection”)
   - every agent looks for identical slots itself (denoted DI, from “distributed intersection”)

2. Masking
   - using distributed masking (denoted DM, from “distributed masking”)
CHAPTER 4. NEGOTIATION PROTOCOLS AND ARCHITECTURE

- using common mask for protection against the intersector (denoted M)
- using no masking (no denotation)

3. Permutation

- using permutation for protection against the intersector (denoted P)
- using no permutation (no denotation)

4. Volume

- processing bit by bit (denoted B)
- processing the entire schedule at once (no denotation)

These parameters can produce 24 combinations of their values, corresponding to 24 more or less similar protocols. Out of these some were rejected:

1. The combination of M or P with DI makes no sense because M and P being a mask and a permutation known by all agents does not offer protection when distributed intersection is used. From security standpoint M_P_DI, M_DI and P_DI are equivalent to DI

2. Instead of using masking (M or DM) and permutation (P) separately, they will always be used together when QI is used

The remaining protocols are shown in boldface in the table below:

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>QI</th>
<th>DI</th>
<th>M_QI</th>
<th>M_DI</th>
<th>DM_QI</th>
<th>DM_DI</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P_QI</td>
<td>P_DI</td>
<td>M_P_QI</td>
<td>M_P_DI</td>
<td>DM_P_QI</td>
<td>DM_P_DI</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B_QI</td>
<td>B_DI</td>
<td>B_M_QI</td>
<td>B_M_DI</td>
<td>B_DM_QI</td>
<td>B_DM_DI</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B_P_QI</td>
<td>B_P_DI</td>
<td>B_M_P_QI</td>
<td>B_M_P_DI</td>
<td>B_DM_P_QI</td>
<td>B_DM_P_DI</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 4. NEGOTIATION PROTOCOLS AND ARCHITECTURE

Note: The IQ protocol uses the intersector type described in section 4.2.2.

There are ten protocols that the user can choose from. Of course, each user chooses independently. That means that the parties might want to negotiate using different (and therefore incompatible) protocols. In order to address this problem, the users will choose sets of protocols and before actually starting the negotiation their agents will need to chose a protocol from the intersection of these protocol sets. The intersection should never be empty. The user interface should not allow the user to generate a protocol set which does not contain the protocol described in section 4.3, which is the most secure out of the ten protocols.

An example protocol set selection method was developed and used in the demo implementation. It is based on asking the user questions about the security preferences and configuring the protocol set according to the answers.

In the beginning all ten protocols are considered. Any answer removes specific protocols from the set and the removals cumulate:

1. Should an intersector be used?

   (a) Yes, even if it learns my schedule. (keeps all protocols)

   (b) Yes, if my schedule stays hidden. (removes QI and B_QI)

   (c) Yes, if the success of the negotiation stays hidden, too. (removes QI, M_P.QI, DM_P.QI and B.QI)

   (d) No, do not use an intersector. (removes QI, M_P.QI, DM_P.QI, B.QI, B_M.P.QI, B_DM.P.QI)

2. Should the schedule be hidden from the other negotiators?

   (a) No (keeps all protocols)

   (b) Yes (removes DI and B_DI)

3. Should the possible moments be tried one at a time?
(a) No (keeps all protocols)
(b) Yes (remove DI and DM-DI)

4. Should negotiator-intersector collusions be prevented? (this question is asked only if the user has chosen to use an intersector but to keep the schedule hidden from both the intersector and the other negotiators)

(a) No (keeps all protocols)
(b) Yes (removes QI, M-P,DI, DI, B-QI, B-M-P-QI and B-DI; additionally, if the answer 3b was chosen, DM-P-QI is removed, too)

After getting the answers to all questions the set of remaining protocols is ordered according to the complexity estimations outlined in chapter 5. The more economical protocols are preferred to the more expensive ones.

4.6 Scheduling Phases

The previous sections evidenced the need for certain operations besides the core negotiation. These operations were grouped in four phases.

4.6.1 Invitation Phase

In this phase one user (initiator) announces the proposal of the meeting to the other intended participants. This contains the topic, the list of invitees, notes about the meeting. A set of proposed negotiation parameters is also transmitted. The recipients announce their position with respect to the original invitation. A recipient might decline the invitation, indicating that he has the intention not to attend the meeting. If a recipient intends to attend the meeting he chooses a set of negotiation parameters and responds to the invitation. There are two possible answers. If the chosen set of negotiation parameters does not include the proposed parameters then the response requests the pre-negotiation phase to take place (section 4.6.2). Otherwise
the response indicates that the recipient of the invitation agrees to use the proposed parameters.

![UML sequence diagram for the invitation phase](image)

Figure 4.5: UML sequence diagram for the invitation phase

The response messages are sent to all the invited participants in order to prevent manipulation, which would be possible if there were privileged parties. The parties that decline the invitation are not sent response messages any more. In figure 4.5 the set of invitation recipients that decline the invitation receives a needPreNegotiation message and does not receive any needNoPreNegotiation message. In practice each recipient that declines the invitation will receive only messages that were sent by the other agents before the declination was processed by them.

The invitation phase is finished when all the agents that do not decline the invitation have each other’s response. If at least one of these responses requests a pre-negotiation phase, the next phase is the pre-negotiation. Otherwise pre-negotiation is skipped and the next phase is the negotiation.
4.6.2 Pre-negotiation Phase

In this phase the parties agree on a common set of negotiation parameters. Each party sends a pre-negotiator agent to a pre-negotiation host which was indicated in the invitation phase.

The most important parameter is what protocol out of the ten possibilities will be used in the negotiation phase. The agents determine this in two steps:

1. They multicast their protocol set to the other agents

2. Each agent determines the intersection of all sets and chooses the cheapest protocol (see section 5) from the intersection set

If a pre-negotiation takes place and the resulting protocol implies the use of a common bit mask or of a permutation, the agents will negotiate these, too. Based on the order of arrival each agent chooses the value of a bit in the bit mask and the mapping of an index in the permutation. The order of arrival is known to the agents in the low-level stage of establishing contact with each other after arriving on the pre-negotiation host.

4.6.3 Negotiation Phase

In the negotiation phase the agents carry out the protocol accepted in the invitation phase or the protocol they agreed upon in the pre-negotiation phase. For details on these protocols see sections 4.2, 4.3 and 4.5.

The purpose of the negotiation phase is to obtain a time slot (its index in the schedule) in which all parties can schedule the meeting. After the negotiators obtain this result, they communicate it to their base host and are disposed of at their own request.

This phase takes place on a special host called “negotiation host” (see figure 4.6), which can be a neutral host, the same host as for the pre-negotiation or one of the hosts of the participants.
Figure 4.6: Agents migrate to the negotiation host

The security approach chosen for avoiding privacy breaches caused by the host is to avoid untrusted hosts altogether. Another, more flexible approach, without employing defensive measures is to have a mix of local and remote communication. Agents that refuse to go to an untrusted host can participate to the protocol from their base host or from another trusted host.

4.6.4 Confirmation Phase

The role of the confirmation phase is to verify the correctness of the negotiation process which was carried out in a potentially hostile environment and take post-factum measures if any irregularities are discovered. Such irregularities can be:

- The result of the negotiation is a time slot when the participant is actually not free
- During the negotiation one of the parties has not respected the protocol (fact logged by the negotiator)
- An error has occurred which might indicate a tampering attempt with the agent’s state or data (fact logged by the negotiator)

The measures that can be taken are blacklisting the offending host or owner of the offending agent if their fault can be determined. Even if it is a fault in the agent’s software it is the responsibility of the owner to use fault-free software.
If the user has no reason to suspect that anything malicious happened during the entire scheduling process she can commit to the negotiated time. This commitment can be done by telephone, by e-mail, by digitally signed e-mail (offers non-repudiation) or by agents carrying digitally signed commitments. If there was a problem then the user must let the other participants know of this, because the entire scheduling is compromised.

If all participants commit to the result of the negotiation then the meeting is scheduled.

4.7 Architecture

Following roles were identified and captured each in a type of agent (see figure 4.7):

- user interface agent, manages user interaction with the system
- dispatcher agent, remains on a safe host and manages the pending negotiations, dispatches the pre-negotiator and negotiator agents and receives their responses
- pre-negotiator agent, takes part in the pre-negotiator phase
- negotiator agent, carries out the negotiation phase

![Figure 4.7: Agents and roles](image_url)
It was chosen to implement each role at agent level in order to couple the
roles very loosely. Each agent can be developed and maintained separately
and parts of the system can be upgraded at runtime.

For starting the system only the user interface agent must be started. It
automatically starts the dispatcher agent, which creates pre-negotiator and
negotiator agents as needed. In order to ensure the consistency of meet-
ing negotiation management, there has to be only one user interface agent
and only one dispatcher agent per user. This requirement is fulfilled by im-
plementing the user interface agent as an instance of the singleton design
pattern. Since aglets override the standard lifecycle events of Java objects,
the singleton pattern had to be implemented in a different way than Java
object singletons. The code excerpts that do this are below:

```java
public class AgentGUI extends Aglet {

    // lock
    private static boolean singletonExistsAlready = false;
    // flag
    private boolean iAmTheSingleton = false;

    /* The agent equivalent of a constructor */
    public void onCreate(Object o) {
        // test and set lock (this operation should be atomic)
        if(!singletonExistsAlready) {
            // acquire lock, this should be the first thing done
            singletonExistsAlready = true;
            // set the flag used for distinguishing
            iAmTheSingleton = true;
        }
        if(iAmTheSingleton) {
            // perform normal initialization routine
```
/** The agent equivalent of a destructor */
public void onDisposing() {
  if(iAmTheSingleton) {
    // release lock
    singletonExistsAlready = false;
    // perform normal finalization routine
    // dispose of the dispatcher agent
    ...
  }
}

} // AgentaGUI

The dispatcher agent refuses to start if it was not launched by a user interface agent (i.e. it disposes). This ensures that there is only one dispatcher, too. The code which achieves this behavior is:

public class AgentaDispatcher extends Aglet {

    /* The agent equivalent of a constructor */
    public void onCreation(Object obj) {
      // When created by the U.I. agent, obj is initialized
      if(obj != null) {

// perform normal initialization routine
...
} else {
    System.err.println("error");
    this.dispose();
}

} // AgentDispatcher

The agents communicate in an application-specific language that is transmitted embedded in secure KQML performatives. The language consists of expressions, built up of name/value pairs. Each expression must have a pair that has the name "keyword".
In the implementation the Expression class extends the standard Java class java.util.Hashtable. The values can be Java objects belonging to any class. In particular classes Integer, String, Vector are used from the standard Java language. Classes BitString and ProtocolSet were added in order to provide the necessary functionality in processing them. This includes a standard conversion to and from strings of characters. The static structure diagram for class ProtocolSet is indicated in the diagram at page 59.
4.8 User Interface

The user interaction with the proof-of-concept implementation is performed through a window based point-and-click graphical user interface. This consists of two types of windows:

- The main window
- The invitation windows

![State diagram for the main window]

When the system is started the main window is opened (see figure B.1 in appendix B). Then the user can open as many invitation windows as she wants. Each invitation window exists independently of the others and does not interact any more with the main window until its action button is pressed.

The interface state transition diagram for the main window shows how the main window reacts to certain events. Some events trigger the creation of an invitation window with the role to gather user input or to display
information about a certain invitation (new, incoming, processed or stored in the lists of the main window).

In order to make the process of gathering and presenting information have a small footprint on the screen, not all pieces of information are dealt with simultaneously. They have been grouped in three classes:

- Meeting parameters (MP)
- Invitees list (IL)
- Negotiation parameters (NP)

Only one class is presented at once, the user being given the possibility to switch between them using a Next/Back linear navigation system (see figure 4.9).

![State diagram for the invitation window](image)

Figure 4.9: State diagram for the invitation window

At any time the user is allowed to cancel the navigation and destroy the window. The action button’s semantics depends on the context of the creation of the invitation window and can be to send a new invitation, or to negotiate an incoming invitation. The Cancel action also has several semantics, depending on the type of invitation window. It can mean simply to close the window, to cancel the creation of a new invitation or the declination of an incoming invitation.

There are several classes collaborating in the implementation of the invitation window. Their aggregation relationships are shown in the diagram 4.10.
Figure 4.10: Structural diagram of the classes that compose the invitation window
Chapter 5

Results

5.1 Complexity

In this section the complexity of each protocol from the negotiation phase is expressed in number of basic operations required for carrying it out.

The basic operation is considered to be the exchange of a message. It is more expensive than masking or permuting a bit string and even than searching in a bit string and intersecting two bit strings because sending a message implies three complex stages:

1. At the sender the content is constructed from basic data, it is authenticated, encrypted, embedded in a secure KQML performative and handed over to the host for transmitting it to the recipient

2. During transit the message passes perhaps through several anonymizing rerouters, if it is a remote communication it is encapsulated in the data structures specific to the network and is subjected to network delays and finally it is inserted into the recipient’s message queue

3. After the recipient retrieves the message from the message queue it decrypts it, verifies the authenticity of the sender, extracts the content
of the message, verifies that it is valid in the context of the protocol and then starts processing it.

It will also be considered that the complexity of a message exchange does not depend on the size of the message content. Since the same protocol with common masking and permutation is more expensive than the variant without, the masking and permutation will each be considered of complexity $\varepsilon$; an arbitrarily small number.

The complexity $C$ of a protocol depends on the number of negotiators and on the length of the schedule. The number of negotiators will be denoted $n$ and the number of bits in the schedule $l$.

Usually the basic operations occur as part of higher-level procedures, which form the building blocks of the protocols. Some procedures use a fixed number of messages, others use the number of messages needed to find a suitable time slot trying one bit at a time. The latter number of times is in some cases limited by the length $l$ of the schedule, other times by the number of identical slots. However, in the latter cases the number is linearly proportional to $l$ and this value will be used in the formulas.

**Queried intersection** Each negotiator sends a message to the intersector and the intersector sends messages back to each negotiator as many times as required (up to $l$). The complexity is $C = 2nl$, that is $O(nl)$ in the average case and $\Omega(nl)$ in the worst case.

**Distributed intersection** This consists of a broadcast performed by each agent sending a message to each other agent. The complexity is $C = n(n - 1) = n^2 - n$, that is $O(n^2)$ in the average case and $\Omega(n^2)$ in the worst case.

**Distributed masking** Each secret element has to visit all agents. That makes $n - 1$ messages for each of the $n$ secret elements. The complexity is $C = n(n - 1) = n^2 - n$, that is $O(n^2)$ in the average case and $\Omega(n^2)$ in the worst case.
The complexity of each protocol is explained below:

**QI**

In this protocol not procedure QI is used but a type of intersector that returns exactly one result. The complexity is $O(n), \Omega(n)$.

**DI**

The only procedure is the distributed intersection itself, the complexity is $O(n^2), \Omega(n^2)$.

**DM_DI**

A distributed intersection is done on data obtained by distributed masking. The complexity is $C = n(n - 1) + n(n - 1) = 2n^2 - 2n$, that is $O(n^2)$ in the average case and $\Omega(n^2)$ in the worst case.

**M_P_QI**

The masking and permutation being done locally, this protocol has complexity $C = 2nl + \varepsilon$, that is $O(nl), \Omega(nl)$.

**DM_P_QI**

The first stage is a distributed masking, the second stage is queried intersection. The complexity is $C = n(n - 1) + 2nl + \varepsilon = n^2 + 2nl - n + \varepsilon$, that is $O(n^2), \Omega(n^2)$.

**B_QI**

Queried intersection on data of length 1 bit is done for each bit separately until (after at most $l$ tries) the protocol finds a result or the data has run out. The complexity is $C = l \cdot 2n$, that is $O(nl), \Omega(nl)$.
CHAPTER 5. RESULTS

B_DI

Distributed intersection is performed at most $l$ times. The complexity is $C = l \cdot n(n - 1) = n^2l - nl$, that is $O(n^2l), \Omega(n^2l)$.

B_DM_DI

The protocol DM_DI is performed at most $l$ times. The complexity is $C = l(n(n - 1) + n(n - 1)) = 2n^2l + 2nl$, that is $O(n^2l),\Omega(n^2l)$.

B_M_P_QI

The masking and permutation being done locally, this protocol has complexity $C = l \cdot 2n + \varepsilon$, that is $O(nl), \Omega(nl)$.

B_DM_P_QI

This protocol performs protocol DM_P_QI at most $l$ times, on data of 1 bit. The complexity is $C = l(n(n - 1) + 2n) + \varepsilon = n^2l + nl + \varepsilon$, that is $O(n^2l),\Omega(n^2l)$.

Summary

The first observation is that the formula is the same for average case and worst case in all protocols. This implies that the protocols degrade gracefully. The fact that the negotiation does not have a positive outcome does affect the resource consumption only by a constant coefficient.

The second observation is that in practice most of the time $n$ will be small and $l$ will be larger than $n$ but will have modest values (there are only about 3000 working minutes in a week). The important factor in the asymptotic functions $O$ and $\Omega$ is $l$. The factors $n$ and $\varepsilon$ are used only for discriminating between protocols with the same asymptotic behavior in $l$.

A further analysis of the protocols is needed, along two coordinates: complexity and degree of security. A (partial) ordering of the protocols based
on their complexity should be determined for ranges of \( n \) and \( l \). For each
ordering the protocols which are more complex and less secure than another
protocol should not be taken into account.

The protocols and their complexities are given in the table below:

<table>
<thead>
<tr>
<th>protocol</th>
<th>complexity</th>
<th>average case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>QI</td>
<td>( 2n )</td>
<td>( O(n) )</td>
<td>( \Omega(n) )</td>
</tr>
<tr>
<td>M_P_QI</td>
<td>( 2nl + \varepsilon )</td>
<td>( O(nl) )</td>
<td>( \Omega(nl) )</td>
</tr>
<tr>
<td>DI</td>
<td>( n^2 - n )</td>
<td>( O(n^2) )</td>
<td>( \Omega(n^2) )</td>
</tr>
<tr>
<td>DM_P_QI</td>
<td>( n^2 + 2nl - n + \varepsilon )</td>
<td>( O(n^2) )</td>
<td>( \Omega(n^2) )</td>
</tr>
<tr>
<td>DM_DI</td>
<td>( 2n^2 - 2n )</td>
<td>( O(n^2) )</td>
<td>( \Omega(n^2) )</td>
</tr>
<tr>
<td>B_QI</td>
<td>( 2nl )</td>
<td>( O(nl) )</td>
<td>( \Omega(nl) )</td>
</tr>
<tr>
<td>B_M_P_QI</td>
<td>( 2nl + \varepsilon )</td>
<td>( O(nl) )</td>
<td>( \Omega(nl) )</td>
</tr>
<tr>
<td>B_DI</td>
<td>( n^2l - nl )</td>
<td>( O(nl) )</td>
<td>( \Omega(nl) )</td>
</tr>
<tr>
<td>B_DM_P_QI</td>
<td>( n^2l + nl + \varepsilon )</td>
<td>( O(n^2l) )</td>
<td>( \Omega(n^2l) )</td>
</tr>
<tr>
<td>B_DM_DI</td>
<td>( 2n^2l + 2nl )</td>
<td>( O(n^2l) )</td>
<td>( \Omega(n^2l) )</td>
</tr>
</tbody>
</table>

### 5.2 Privacy

This section focuses on the proposed negotiation protocol (see 4.3), which
uses distributed masking and distributed intersection. The results apply both
to the bulk and the bit-by-bit variants.

Since no studies have been performed on schedules to reveal their sta-
tistical properties, in the following it will be assumed that the bit strings
representing schedules are statistically unbiased. That means, they can not
be distinguished from a random bit string of the same length.

Recall that time slots which have the same value at all participants are
called identical slots. The number of identical slots between two schedules
is (in the average case) 50% of all slots. In the case of three schedules the
number is on the average 25%. In the case of \( n \) schedules the number will be
as low as \( \frac{1}{2n-1} \) of the total number of slots (see figure 5.1). The explanation is
that for $n$ schedules there are $2^n$ different $n$-tuples of bits, each with roughly the same number of occurrences in the schedules. Out of these, only two tuples represent identical slots: $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$. The proportion of identical slots is therefore $\frac{2}{2^n} = \frac{1}{2^{n-1}}$. In case of the bulk variant of the protocol this is the proportion of every other schedule that will be known to every agent.

![Diagram](image)

Figure 5.1: Fraction of identical slots out of the total number of slots

Of the identical slots about half are tuples of 0 and half are tuples of 1. Since we represent slots in which a meeting can be scheduled by a 1, the tuples containing only 1s will be called good slots. The tuples which contain only 0s will be called bad slots because they are identical slots and as such reveal information about everybody else’s schedule without allowing for a meeting to be scheduled.

Bad slots are a privacy breach that is part of this protocol. One must not try to attack the protocol to have access to bad slots. By using the bit-by-bit version of the protocol the breach is much reduced. After one good slot was found the protocol stops and no more bad slots are revealed.

An estimation can be made of how large the breach remains. The chances that the first try is a good slot are 50%. That means in 50% of the cases the privacy breach is zero. The chances that the good slot comes after a bad slot are 25%. This approach will be used for counting the number of bad slots that breach the privacy of the users.
CHAPTER 5. RESULTS

For a set of \( p \) identical slots there are \( 2^p \) possible sequences of good and bad slots. The number of bad slots will be counted in following way (see figure 5.2):

- After testing the first slot in half the cases we will have one bad slot revealed. The number is \( \frac{1}{2}2^p = 2^{p-1} \)
- After testing the next slot, in half of the cases with bad slots from the first point we will reveal another bad slot. The number is \( 2^{p-2} \)
- \ldots
- After testing the \( p \)-th slot in one of the two remaining cases we will have another bad slot. The number is \( 2^{p-p} = 1 \)

When the bad slots are summarized, their total number is

\[
B = \sum_{i=1}^{p} 2^{p-i} = \sum_{i=0}^{p-1} 2^i = 2^p - 1
\]

Out of the \( p2^p \) slots that exist in all \( p \) cases the number of bad slots represents:

\[
b = \frac{2^p - 1}{p2^p} = \frac{1}{p} \left( 1 - \frac{1}{2^p} \right) < \frac{1}{p}
\]

But \( \frac{1}{p} \) is exactly one slot out of a schedule. The formula can be interpreted as “on the average at most one bad slot will be revealed from each schedule”.

Figure 5.2: Number of revealed slots (only identical slots showed)
That is, the participants will know about each other that everybody is busy at that certain one time.

In the worst case no meeting can be scheduled and all $p$ slots are bad. That still represents only $\frac{1}{2^{n-1}}$ of the schedule.
Chapter 6

Conclusions

This project has fulfilled its goals. A design was developed for a meeting scheduler with mobile software agents and a demo system was written to prove the validity of the concept.

The nature and characteristics of mobile software agents were studied. The advantages and disadvantages were evaluated and it was found that using mobile software agents is justified in the case of this application. The security issues connected with agent mobility were pointed out and ways to solve the security problems were indicated where they existed.

A special purpose negotiation protocol was developed. The designed protocol focuses on not revealing during the negotiation more information than is needed for the negotiation to succeed. This goal was achieved in a very large proportion:

- in 50% of the cases no extra information is revealed
- on the average all parties will learn one time slot when everybody is busy
- in the worst case a fraction of $\frac{1}{2^n - 1}$ time slots are revealed of the total number of slots, $n$ being the number of participants
In order to give the user the possibility to achieve a tradeoff between efficiency and privacy a total set of ten protocols was developed and evaluated with respect to efficiency.

Further research ideas:

- Studying how the ten protocols achieve the tradeoff between efficiency and privacy in order to eliminate the ones who are bad at both

- The proposed protocol reveals much more than in the general case when only two parties are involved. Which of the ten protocols are safe enough in that case must be determined

- Other (and more intuitive) methods for allowing the user to specify preferences should be designed as part of the effort to improve the user interface

- The system should be implemented on another agent platform and the interoperability offered by the standardized negotiation language should be tested in real conditions
Bibliography


Appendix A

Existing Meeting Schedulers

This section provides a survey of existing meeting scheduling systems and analyzes their characteristics with respect to protecting the user’s privacy. It serves for illustrating the context in which this project was developed.

A.1 CDE Calendar

The Common Desktop Environment (CDE) is the window manager and suite of desktop tools and applications developed in collaboration by Hewlett Packard, IBM, Sun Microsystems and Novell. It includes an application called “Calendar” which allows the user to manage his agenda of events and even schedule appointments or common events with other users of “Calendar”.

“Calendar” allows the specification of meeting specific information, like the time, the topic, the author, the frequency of the event (e.g., “every two weeks”), as well as some security parameters. Each appointment possesses a privacy level with three values:

- “Others see time and text”, corresponding to “public”
- “Others see time only”, corresponding to “semi-private”
• "Others see nothing", corresponding to "private"

Besides this, a user can construct permissions matrices for each user, assigning them rights to view, insert or change appointments for each of the privacy levels independently.

Appointment negotiation occurs by consulting the users’ calendars. There exists a "Compare Calendars" menu option which highlights the common free times of two calendars, using only information available according to the permissions that the user has.

One can see that this application is privacy-aware and that it allows the user to control to some extent the degree to which information about the calendar is revealed.

### A.2 Yahoo! Calendar

As opposed to the previous desktop application, this is a web-based application. It is provided by Yahoo! Inc. to their users only.

It describes a meeting in terms of time, title, description, duration, type and repetition and also offers the user the possibility to specify for each meeting a level of privacy, having one of the values:

• "private" - others cannot see the event
• "busy" - others can see only the time, not the details of the event
• "public" - others can see the time and details of the event

The system allows one to compare one’s own calendar with that of another Yahoo! user and automatically determine the possible meeting times. This feature is limited by the visibility level described below.

A calendar can be made visible to anyone, only other Yahoo! users given in an access list or to no one else but the owner of the calendar. Of course,
none of these measures applies to Yahoo! itself. Since all data is stored on their servers, if they wish so, they can view and even edit a user’s calendar.

A.3 Proposals From the Literature

Rowan Dordick presents in [4] following scenario:

Consider the task of setting up a meeting among several busy people who work at remote locations and do not use the same software or hardware that would enable a secretary to view entries in their online calendars, let alone make any changes. One solution would be to send out a software agent, sometimes called an "intelligent agent," with the capability of scanning the participants’ calendars, choosing an optimal date and time and making an entry in their calendars. In such a scenario, data is being pushed onto the participants’ systems without their active involvement.

While it illustrates a meeting scheduler that makes use of mobile software agents, it doesn’t mention any security related measures to protect the calendars from being abused in any way.

In article [12], Bill Venner also explores the possibilities of mobile software agents in solving the scheduler problem:

Besides searching databases and files, agents can gain information by interacting with other agents. If, for example, you want to schedule a meeting with several other people, you could send a mobile agent to interact with the representative agents of each of the people you want to invite to your meeting. The agents could negotiate and establish a meeting time.
In this case, each agent contains information about its user's schedule. To agree upon a meeting time, the agents exchange information.

This approach doesn’t mention security either but it would accommodate the enforcement by the agents of security measures for protecting the schedules.
Appendix B

Manuals

B.1 Installation Manual

System requirements:

- A computer with an operating system with a graphical user interface and the TCP/IP protocol stack installed
- The Java Development Kit (JDK), version 1.1, installation directory stored in the environment variable JDK\_HOME
- The Aglets Software Development Kit (ASDK), version 1.1beta3, installation directory stored in environment variable AGLET\_HOME

Installation steps:

1. unpack the archive agenTa.tgz, directory install will be created
2. copy subdirectories lib and public into the directory indicated by AGLET\_HOME, this will add subdirectories cosic to the existing lib and public directories and not remove anything
3. start the Tahiti aglet viewer
4. press button “Create”, this will open a window labelled “Create Aglet”

5. type cosic.agenta.AgentaGUI in field “Aglet name”

6. press button “Add to list”

7. the meeting scheduler can already be launched by pressing button “Create”

8. directory install can be deleted

Note: The program being a proof-of-concept version still has some initialization information hardwired into the code. In order to change this, the code of the method AgentaDispatcher::initContactsInfo() must be edited. The AgentaDispatcher class can be recompiled with the command

```bash
```

The meeting scheduler can be restarted for the changes to take effect by exiting the running scheduler agent and starting it again (see section B.2) but pressing “Reload Class and Create” instead of just “Create”

## B.2 User’s Manual

The meeting scheduler is started by opening the “Create aglet” window of the Tahiti aglet viewer, choosing “cosic.agenta.AgentaGUI” from the list and pressing “Create” (or “Reload Class and Create” if it restarted for letting some changes take effect). This step opens the main window of the meeting scheduler (see figure B.1).

The meeting scheduler is ready to accept incoming invitations (see section B.2.3, “Answering to an incoming invitation“).
B.2.1 Operating the Main Window

For terminating the application, use button “Exit”. For creating and sending an invitation, press button “Create invitation” (see section B.2.2, “Creating a new invitation”).

After an invitation or the reply to an invitation is sent, the invitation appears in the corresponding list by its date and topic. A more detailed description of the invitation and of the answer sent can be obtained by double clicking the invitation in the list. This action pops up an informative invitation window with all interface elements disabled.

As some of the parties decline the invitation they are removed from the invitees list of the declined invitation. This is done without any notice to the user. The user will see only that the person is missing from the list when the detailed description is visible.

Occasionally an ongoing negotiation will come to a conclusion. The result of a negotiation is showed immediately by popping up an informative invitation window with all interface elements disabled. The window title shows whether the negotiation succeeded or failed.
If it failed, the invitation is removed from the list it was in.

If the negotiation succeeded, a “Time” field is present in the Meeting Parameters state of the window indicating the time which was agreed upon during the negotiation. The invitation is moved from the list it was in to the leftmost list, “Scheduled meetings”. Its details can be retrieved just as for the other two lists, by double clicking on the invitation.

### B.2.2 Creating a New Invitation

For creating and sending an invitation, press button “Create invitation” in the main window. This will open the invitation window prepared to gather input for a new invitation. The window shows the first class of parameters: Meeting Parameters (see figure B.2). After filling out the fields (all are optional), press the “Next” button.
Figure B.3: Screenshot of the IL state of the invitation window

This will make the invitation window show the interface elements that enable the Invitees List to be specified (see figure B.3). The left side list contains all contacts with which the meeting scheduler was initialized. Multiple contacts can be selected from the list and placed in the invitees list on the right side by pressing button “— >”. For removing contacts from the invitees list, select them and press button “< — ”.

Pressing button “Next” will bring the Negotiation Parameters class of interface elements in view (see figure B.4). The array of checkboxes along the left edge of the window allows the introduction of the set of times when the meeting can be scheduled. The right side contains questions about the way the negotiation process should be carried out. Only valid combinations of questions are allowed, that is why sometimes question 4 will be disabled.

Pressing “Cancel” at any time before sending the invitation aborts the process. When the invitation window is in this state the “Send” button
Figure B.4: Screenshot of the NP state of the invitation window

becomes active. Pressing it will send the invitation to the contacts on the invitees list. If there are no contacts on the invitees list, an instruction window pops up. The two options are to specify at least a contact in the invitees list or to press “Cancel”.

After the invitation is sent, an entry is created for it in the “Sent invitations” list of the main window. For operating the main window refer to section B.2.1.

B.2.3 Answering to an Incoming Invitation

When the meeting scheduler receives an incoming invitation it pops up a window presenting informations about the invitation and requesting input for the reply.

This window should be closed by either pressing button “Decline invita-
tion” to announce unwillingness to participate to the meeting or by pressing “Negotiate” to participate in scheduling the meeting.

The latter is possible only after the information about the meeting (topic, date, notes and invitees list) was read. This information is presented in uneditable interface elements. In the Invitees List state of the window there are not two lists to allow choosing but only one, for displaying the invitees list chosen by the sender of the invitation.

In the Negotiation Parameters state, the suitable times can be specified and answers to the questions regarding the negotiation can be given. Initially the answers are set in such a way that the negotiation is performed in the most efficient way allowed by the sender of the invitation. The radio buttons for answers are initially disabled. If another set of answers is desired the buttons can be enabled by checking the “Change” checkbox.

After “negotiate” was pressed, an entry is created for it in the “Received invitations” list of the main window. For operating the main window refer to section B.2.1.
Appendix C

Code