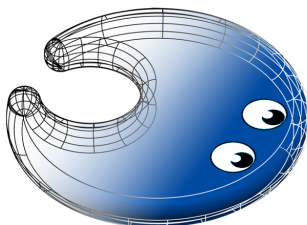


Agent Technology for Personalised Unified Messaging



Hi there,
Just wanted to introduce myself, after
moving around here for a while.
greetings,
your agent



Remco Schaar

 Tryllian  TU Delft

Agent Technology
for
Personalised Unified Messaging

An agent-based unified messaging system with intelligent user support



Tryllian B.V.
Amsterdam, The Netherlands



Delft University of Technology
Delft, The Netherlands

Master's thesis project

Remco Schaar

June, 2002

Graduation Committee:

Drs. dr. L. J. M. Rothkrantz

Prof. dr. ir. E. J. H. Kerckhoffs

Drs. M. V. Jonkers (Tryllian B.V., Amsterdam, The Netherlands)

Prof. dr. H. Koppelaar (chairman)

Schaar, Remco M. (remco@ch.tudelft.nl)

Master's thesis, June 2002

“Agent Technology for Personalised Unified Messaging

An agent-based unified messaging system with intelligent user support”

Delft University of Technology, The Netherlands

Faculty of Information Technology and Systems

Department of Mediamatics

Knowledge Based Systems Group

Keywords: Agent technology, unified messaging, user profiling, mobile agents,
multi-agent system, intelligent agents, communication overload,
cross-media profiling, Tryllian ADK

Copyright © 2002 Tryllian Holding N.V. All rights reserved. No part of this document shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with the respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this document, the publisher and author assume no responsibility for errors or omissions. Neither is any liability assumed or damages resulting from the use of the information contained herein.

All product names referenced are trademarked by their respective companies.

Preface

The last requirement for the completion of the study Computer Science at Delft University of Technology, is to conduct a research project. I chose to fulfil this project at the Knowledge Based Systems group, headed by Prof. dr. H. Koppelaar. After gathering the necessary motivation to commit myself for nearly a year, I had to choose a location and topic. I ended up to look for a place where I could combine network- and agent-technology with knowledge based systems. This led me to ask Tryllian whether they had an opening for a graduation internship. A few meetings later, we agreed I would start my internship at Tryllian half October 2001.

About Tryllian Tryllian is a Dutch high-tech company focusing on mobile agent technology. It was founded in March 1998 by Christine Karman, supported with venture capital. The first product, *Gossip*, was an application of mobile agents to search the Internet and share with other users. Further development led to the *Agent Development Kit* (ADK), an industrial strength software development kit for the construction of mobile agent application. Focus areas are remote management, embedded systems and especially business integration. In 2001, Tryllian was awarded as *ICT company of the year* by several leading Dutch ICT organisations. At the moment, Tryllian employs about 40 people, forming a young and dynamic company.

Project The project indeed became a combination of network-, knowledge- and agent-technology. As a person often annoyed by mobile phones and lack of access to e-mail, the subject of unified messaging came up. Hence, the project was formulated to combine unified messaging with user profiling. Important element was the role (mobile) agent technology can play herein. After some study in related literature, a design was developed and described in detail. Together with the implementation and evaluation of a prototype, the project became a complete challenge. The results of this work are described in this report.

PREFACE

Acknowledgements First of all, I would like to thank all the people at Tryllian for granting me an internship with the company. It is a great thing that a company supports research and students, even in times when the economy is less favourable. I am convinced mobile agent technology is the next step forward in computer science. The Tryllian ADK is a very good product, I can recommend it to everybody who needs anything beyond a simple desktop application.

Furthermore, I specially want to thank Mark Lassche, Menno Jonkers and Leon Rothkrantz for their support and time. Without their feedback, comments, advices and criticism, I probably never achieved a coherent and finished thesis. Mark, for taking the ungrateful job of forcing me to make a planning once in a while. Menno, for his enthusiasm at each new pile of paper to review. Leon, for reminding me on the academic nature of my research. All three are thanked as well for coauthoring my paper.

Of course I am very thankful to everybody at Tryllian; for their support with my struggles with Java and Swing, the seminars and drinks, the LAN-party and carting, even accepting (some of) my music, throwing penguins to me and . . . Mark, Menno, Wiebe, Norbert, Mark, Otto, Peter, Anthony, Simon, Hes, Yigal, Boudewijn, Mimi, Bram, Eduard, René, A3aan, Serge, Justin and all the others. . . Thanks!

There are many more people who helped me forward as well and I want to thank them therefore. Both professor Koppelaar and professor Kerckhoffs for their efforts as member of the graduation committee. Ralph, Google and Citeseer for finding lots of decent literature and Taco for proof-reading my paper. All those not mentioned here as well and last, but certainly not least, my family and friends for their support throughout my study.

Remco Schaar
Delft, June 2002

Abstract

Now that telecommunications and Internet are merging to one global communication network, new possibilities and their related problems arise. Unified messaging as used in practise nowadays only provides for the exchange of data. It is much more complicated if one wishes to receive the information rather than the data. More and more people start to complain about the large amounts of e-mail they receive every day. User profiling has often been applied to e-mail as text, but with unified messaging one uses various media. The combination of unified messaging and user profiling with agent technology is a last element of this study. Backgrounds for these areas are provided in this document.

To handle this problem, an architecture has been developed. This architecture is based on multi- and mobile agent technology. Intelligent user profiling is directly incorporated in the design. Due to the separation of profiling in two phases, it can be achieved across different media. This is accomplished by using meta-data to describe a message, rather than using its content directly. The usage of agents supports a high level of personalisation of the system.

The proposed architecture has been implemented in a prototype version. This prototype was used for an evaluation of the concept. This evaluation shows that agents are great facilitators for personalised unified messaging. Overcoming a communication overload is handled by an intelligent agent that is capable of learning an user profile. This learning problem is clearly defined, and a possible solution is found in the k -nearest-neighbour algorithm. Further benefits of the usage of agents are shown as well. The system is dynamically extensible and highly adaptable, with robustness in a distributed environment.

Contents

Preface	iii
Abstract	v
Contents	vii
1 Introduction	1
I Project Definition	3
2 Background	5
3 Problem	7
3.1 Unified messaging	8
3.2 Communication overload	9
3.3 Agent technology	11
4 Goals	13
4.1 Research	14
4.2 Design	14
4.3 Prototype	15
4.4 Deliverables	16
5 Scope	19
5.1 Issues	19
5.2 Included	23
5.3 Excluded	25

II	Backgrounds	27
6	Unified Messaging	29
6.1	Research	29
6.1.1	Unified messaging	29
6.1.2	Media processing	31
6.1.3	Standards	32
6.1.4	Related	33
6.2	Practise	33
6.3	Developments	34
7	User Profiling	37
7.1	Motivation	37
7.2	Information filtering	38
7.2.1	Global technique	38
7.2.2	E-mail filtering	39
7.3	Machine learning	40
7.3.1	Nearest neighbour	41
7.3.2	Decision tree	41
7.3.3	Rule-based generated by clustering	42
7.3.4	Bayesian learning	42
8	Agents	43
8.1	Perspective	43
8.1.1	Properties	43
8.1.2	Classification	44
8.1.3	Agency	46
8.1.4	Practise	46
8.2	Agent communication	47
8.2.1	Languages	47
8.2.2	Architectures	48
8.3	In telecommunication	49
8.3.1	UMTS/VHE	49
8.3.2	Intelligent Network	50
8.3.3	IMPAX	50
8.4	Alternative technologies	51
III	Concept	53
9	Requirements	55

CONTENTS

9.1	Must have	55
9.2	Should have	56
9.3	Could have	57
9.4	Assumptions	57
10	Func. Design	59
10.1	Offered functionality	59
10.2	Use cases	61
10.3	Internal functions	62
11	Selection	65
11.1	Multi-agent	65
11.1.1	Why?	65
11.1.2	Compartment	66
11.2	Peripherals	67
11.2.1	Motivation	67
11.2.2	Input	69
11.2.3	User Interface	69
11.2.4	Storage	70
11.2.5	Transformation	70
11.3	Routing	71
11.4	Profiling	73
11.5	Miscellaneous	75
12	Architecture	79
12.1	Overview	80
12.2	Roles	81
12.3	Agents	82
12.3.1	Message Factory Agent	83
12.3.2	Router Agent	84
12.3.3	Extractor Agent	86
12.3.4	Profile Agent	87
12.3.5	Transformer Agent	89
12.3.6	UI Agent	90
12.3.7	Storage Agent	92
12.4	Communication	92
12.5	Other components	93
12.6	Work- and dataflow	95
12.6.1	Important new message	96
12.6.2	Unimportant new message	98
12.6.3	User situation change	100

CONTENTS

12.6.4	User feedback	102
12.6.5	Store message	104
12.6.6	Request message	106
12.6.7	Present message	108
13	Anticipations	111
13.1	Profile learning	111
13.1.1	Learning problem	111
13.1.2	Examples	113
13.2	Extension	115
13.2.1	Automated translations	115
13.2.2	Sending messages	115
13.2.3	Synchronous communication	116
13.2.4	Enhanced dynamic extensibility	117
13.2.5	Other agents from the user	117
IV	Prototype	119
14	Technical Design	121
14.1	Task model	121
14.1.1	Factory Agent	122
14.1.2	Router Agent	122
14.1.3	Extractor Agent	126
14.1.4	Profile Agent	127
14.1.5	Transformer Agent	128
14.1.6	UI Agent	129
14.1.7	Storage Agent	130
14.2	Object model	133
14.2.1	Router Agent	133
15	Implementation	135
15.1	Environment	135
15.2	Messages	135
15.2.1	Envelope	135
15.2.2	Content	137
15.3	Agent interaction	137
15.4	Factory Agent	137
15.4.1	E-mail Factory	137
15.4.2	RSS Factory	138
15.4.3	SMS Factory	138

CONTENTS

15.5 Router Agent	139
15.5.1 Task model	139
15.5.2 Locating Service Agents	139
15.6 Extractor Agent	139
15.7 Profile Agent	141
15.7.1 Dummy Profile	141
15.7.2 Random Profile	141
15.7.3 Rule-based Profile	141
15.7.4 KNN Profile	143
15.8 Transformer Agent	144
15.9 UI Agent	145
15.9.1 Swing UI	145
15.9.2 Phone UI	146
15.10 Storage Agent	146
15.11 Runtime examples	146
15.11.1 Example: important	146
15.11.2 Example: unimportant	150
15.12 Deviations	151
V Evaluation	153
16 Functional Tests	155
16.1 Objectives	155
16.2 Approach	158
16.3 Experiments	161
16.3.1 Extensibility	161
16.3.2 Adaptability	163
16.3.3 Personalisation	165
16.3.4 Cross-media profiling	166
16.3.5 Robustness	167
16.3.6 Queueing	170
16.3.7 Learning	171
16.3.8 Mixed	175
17 Interpretation	177
17.1 Tests	177
17.1.1 Extensibility	178
17.1.2 Adaptability	179
17.1.3 Personalisation	179
17.1.4 Cross-media	180

CONTENTS

17.1.5 Robustness	180
17.1.6 Queueing	181
17.1.7 Learning	182
17.2 Prototype	182
17.3 Design	184
VI Conclusions & Recommendations	189
18 Overview	191
18.1 Backgrounds	191
18.2 Design	192
18.3 Prototype	194
18.4 Evaluation	195
19 Discussion	197
19.1 Non-mobile Router	197
19.2 Overhead	198
19.3 Scalability	199
19.4 Communication overload	200
19.5 Commercial realisation	200
19.6 Agent negotiation	201
19.7 Out of scope	202
20 Conclusions	205
20.1 Unified messaging	205
20.2 Communication overload	206
20.3 Agent technology	206
21 Future Work	209
21.1 User profiling	209
21.2 Security	210
21.3 Performance	210
21.4 Virtual secretary	210
21.5 Related work	211
21.6 Tryllian ADK	212
Bibliography	215
List of Figures	223
List of Tables	225

CONTENTS

VII Appendices	227
A Tryllian ADK	229
A.1 Architecture	229
A.2 API highlights	230
A.3 More	231
B Diagram Legend	233
B.1 Workflow	233
B.2 Dataflow	234
B.3 Task hierarchy	235
B.4 Task-state transition	235
C Abbreviations	237
D Paper	239

Chapter 1

Introduction

“You have thirteen new e-mails, six new SMS-messages, three people tried to ICQ with you, there are four new news-items, seventeen new Usenet-discussions in your favourite groups and you have two faxes and one voice-mail.”

With the forthcoming merger of telephone-networks and Internet, it becomes possible to create an integrated communication-portal for users. This will allow users to receive all their messages using a single application. Included are all kinds of message-based communications, from e-mail to voice-mail and CNN-headlines to Usenet-discussions. But would it not be more usable if the above was stated as follows:

“You have seven new messages that are currently important.”

The last statement is more informative for the user, a lot more comprehensible and shorter as additional gain. An user would have a great benefit if it received the information rather than the raw messages. In this era of telecommunication one receives more and more messages, and the challenge becomes selecting the real information in this endless stream. An user has specific interests for these messages, and thus prefers to receive certain messages as important. Users can be in various situations and each situation will have related interests. Not many people wish to be interrupted at home by a customer, while this customer would receive immediate attention at work. The most interesting messages should thus be selected based on the user's situation.

Although users could explicitly express their interests, this is not always that simple. Creating computable rules for their interests even complicates this task. The best solution would thus be a system that can automatically

deduce these interests. An intelligent computer system can help the user to selected the interesting new messages.

A last important aspect is the fact the interests of the user are personal. Everybody has his or her own preferences for certain information. How this information is received can also be bound to the person receiving it. An automated solution will therefore have to be personal and possible even further adaptable for the user.

The idea induced from the above is to create a personal assistant, that can help the user to select new important messages. In combination with modern telecommunication, this assistant would travel with the user wherever it goes. This not necessarily means the user has to carry a device for this assistant everywhere, it should follow in any way.

A possible solution can be found with agents. Intelligent agents can help solve complex problems, and personal agents have represented people before. Other agents can move around networks and operate all over their virtually world. Agents thus seem a viable approach to create an intelligent assistant that helps the user everywhere. Today's technology is expected to be able to form an assistant as suggested.

A study to establish how this can be achieved has been performed. Part of this study was a proof-of-concept implementation to verify whether it can be realised. This report describes both the study and the prototype in detail.

This document is divided in six parts. The first describes the problem that is investigated and defines the project. Part **II** gives some background information of related areas of research and concepts. The next part introduces the developed architecture. On page **121** and further the created implementation is explained, including the technical design. Further, part **V** has some tests that were performed on the prototype. Last follows a discussion and the conclusions and recommendations, in part **VI**.

Part I

Project Definition

This is a full description of the project. It includes a description of the background and the problem to be dealt with (respectively chapter 2 and 3). Chapter 4 will define the goals of the project. Further, it states all issues involved and a determination of the scope and direction of the project in its last chapter, number 5.

Chapter 2

Background

Always, everywhere, anyhow and certainly lots of it... that seems to be the future of telecommunications. People will be able to communicate wherever they are, whenever they want and how they want to. When you are at work you can read on your desktop the voice-mail just received, while driving home you can listen to your e-mail and when arriving home you have a summary of today's news of your interest among your e-mail. The future of telecommunication looks very promising. People will be able to communicate in such large amounts, it even becomes a burden.

In the last decade, the usage of telecommunication has grown beyond many expectations. The mass acceptance of second generation mobile telephone systems (GSM) and the Internet have led to an enormous growth in electronic communication. Recent introduction of i-mode and GPRS provide improved data-rates and allow one to be continuously on-line. With the announcement of the third generation (3G) of mobile telecommunication (UMTS), reachability of people improves further. Along with acceptance of broadband, ADSL and similar high-bandwidth Internet connections, available means of communication grow even further. More and more people start complaining they cannot keep up with the rapid growth of these new means of communication. This counts for not being able to access new messages whenever the user finds it suitable, and receiving loads of (unimportant) messages.

With the integration of phone- and data-networks (like Internet), new opportunities become available. Electronic messages¹ are no longer limited to a single electronic network. Connections between, or the merger of, both networks allow us to send a voice-mail to an e-mail account, or send an e-mail to a fax. This concept is often referred to as *unified messaging* [79].

¹E.g. fax, e-mail, SMS, voice-mail and instant messages.

In the age of dot-com, several companies stood up that started with (free) public available unified messaging. They allow you to receive your fax and voice-mail through your e-mail, or forward your e-mail to your fax (e.g. [86]). Since access to e-mail is now available through mobile Internet connections and WAP and the nearby introduction of better bandwidth-enabled UMTS, all kind of messages can be available always and everywhere.

But from the use of e-mail we already know communication can be a burden as well. Many people who open their e-mail inbox after their vacation, often first notice the large amount of new mail messages. More and more people even experience this every day. You would not be the first to find dozens of new messages first thing in the morning, and even more throughout the day. Estimates and forecasts up to four hours a day for handling e-mail alone have been made [13].

The problem of spending lots of time for handling all mail is a known problem. In community and literature this has been identified as *e-mail overload* [17, 44, 82]. This refers to e-mail alone however. With unified messaging, one deals with more media, and this applies to a broader spectrum of communication. One could say we face a *communication overload*.

Chapter 3

Problem

The future of telecommunication allows us to receive new messages anywhere, anytime¹. Due to mobile telecommunication equipment, people can communicate everywhere. With the next generation of mobile networks (UMTS) better mobile data-connections will become available [77]. People will therefore be able to be continuously reachable for all kinds of communication. Several issues have to be expected however, which are addressed in their respective sections:

1. When continuous reachable, one prefers the use of only a single device at one time. A problem is a way to deal with the large variety of interconnected devices of many people. On the lowest level, most modern communication equipment can be connected with each other. These devices can exchange data this way, but it remains to be seen whether both devices can deal with the information contained in the data [56]. It is not realistic to ask a recipient to acquire a similar device when a sender acquires a new type. The sender's new device might lack full potential however, when it has to be backward compatible with all equipment of the sender's contacts. Connecting these devices to exchange the information rather than the raw data, seems a more logical gain for new devices, but is not fully met yet. Most important herein is the idea to exchange semantics, a thought, rather than a message. Throughout the used media, these semantics should be maintained.
2. Although most people prefer receiving their messages rather now than later, this might lead to less productivity due to constant interruption.

¹Although this can be accomplished with known technology, this is economically and practically not realistic.

Another, maybe even more important, effect could be overlooking important information faded in the flood of less important communication. With the volume of electronic messages send nowadays, everybody faces a vast amount of messages. Published predictions commonly show even further increase of these numbers [13, 30]. Dealing with all messages in time, without losing too much time for other activities becomes a challenge.

3. Third element is the relation with (mobile) agent technology. Quite a few designs are proposed for the application of agent technology in these communication systems (e.g. [22, 37]). Most of these system are not known as (commercial) usable systems, or as systems with many users. Most projects appear to stop after a design [20, 37] or a prototype [12, 36]. Although agent technology has many promises [35], evaluation of agents applied to (personal) communication has not received much attention. Further research for the opportunities and drawbacks in this context seems appropriate.

3.1 Unified messaging

Many people will recognise the following scenario: You have an electronic article, which you want to send to a colleague today. You know however, that he will only be reachable on his cellular phone since he is out of the office. Using a fax is not an option, since he has no laptop connected to his phone. Sending by SMS (Short Message Service) will not do either, the article is too long, even for a dozen of messages. Using your colleague's voice-mail has the same limitation. The only solution left is reading the entire article to your colleague directly.

People can communicate all over the world at the speed of light nowadays². We even have a broad choice of media we can use. The essence of communication remains the same for all forms. Whether it is an e-mail, a fax or a voice-mail, they are all used for the exchange of thoughts. The end-users express their thoughts through some media, be it speech, text or image. Normally they choose whatever they find convenient, since the medium³ is not what they want to communicate. Lacking telepathic capabilities, humans

²Almost literally in the case of optic fiber.

³Medium, media-types and format will be used as equivalent in this document. They are used for both the used medium (i.e. image, audio, text) as well as specific document formats. One can best compare these with a MIME-type [19], consisting of a major and a minor type (e.g. `text/plain`).

have to exchange their ideas through some medium. The medium functions as an intermediate carrier between the communicating parties.

Until recently, the sending party commonly choose the medium to be used, thereby forcing the receiving end to use the same medium. When this poses a problem (e.g. a letter for illiterate or blind people), it is often up to the recipient to solve this, for example by using an intermediate person. The sender also determined the location where the other would receive the message. This implied restrictions to the receiver as well.

The choice of communication channels seems to have grown on a regular basis. New devices and technologies allow a broader choice of media. Every new medium has to be integrated with existing systems, or require new ones on the receiving side as well. Many people cannot keep up with all new possibilities, and a general solution is needed.

With modern communication equipment, it becomes possible to have a broader choice of media to use. As a result of this progress, the sender can have even more influence on the receiver. But modern technology also enables the receiving side to have a choice. With electronic communication, automated transformation becomes possible. This allows the receiver the choice of medium as well. Receiving a (electronic) letter nowadays no longer forces you to read the letter, but you can listen to it with text-to-speech software. Since the medium is only an intermediate, the communicated thought is still delivered.

The idea to have your incoming communication anyway and everywhere you want it, is referred to as *unified messaging* [79]. With unified messaging, the receiver determines where and how his incoming messages are delivered. This allows you to have a choice rather independent from the sender. Receive your messages where, when and how you want them is the ultimate goal of unified messaging.

Most current commercial solutions offer only a limited range of possibilities (e.g. [27, 71, 78, 86]). A complete integration of all devices and sources into one acceptable channel for the user is not achieved. Adding new devices to these systems is limited to whatever these providers integrate in their systems. A more generic solution to integrate devices and media for the user is desired.

3.2 Communication overload

“You have 84 new messages”, is the announcement of the e-mail client. This is an actual number from the authors inbox for only 1 mailing-list over a three day period. The number of regular users of e-mail is still increasing, as well

as the amount of messages send per day [30]. With the more recent growth in popularity of SMS [21] more piles of messages rise. People communicate more and more since the first electronic message.

Although people themselves choose to communicate more, it has its drawbacks as well. People spend increasingly more time handling all their phone calls, e-mail and SMS. Besides, normal work is interrupted more often. With the introduction of e-mail notification programs, people often read their e-mail right after arrival. This causes interruption of the work in progress, requiring task changes and loss of concentration. Since people can respond faster to new messages than ever before, even more communication is the result.

Aside from this interruption lies the fact that electronic communication costs money. In the case of SMS, each message is charged to the sender⁴. If one reads ones e-mail, you often have to pay for the amount of transferred data (either for the required time of connection, or the size of the information). Specifically when using mobile data-connections, these costs can increase rapidly when large or unwanted messages are received [81]. Listening to ones voice-mail is a billed service at several providers as well.

The advantage of being able to respond immediately is the ability to take timely action. The disadvantage is the interruption of current activities with communication for other tasks as well. Context switches are needed continuously in this setting, causing the same tasks to require more time. Further, changing ones current thoughts, can lead to more errors in work. One can conclude that timely notification of arriving messages is wanted, but should be limited to messages that are either important or related to the receiver's current tasks.

Previous research brought up the name *e-mail overload* [82, 89]. This is the phenomenon of receiving e-mail in such large amounts, the availability of e-mail becomes a burden instead of a blessing. Already numbers have been estimated and predicted accounting for up to 4 hours a day for handling e-mail [13]. This can become a real problem; when people start spending more time on communication, they lack time for other activities. With more means of communication tied together, this can increase even further. This problem might even be a broader *communication overload* [26].

An ideal solution would be to only receive all *wanted* incoming communication immediately. Of course a secretary can fulfil this function, but not everybody can afford one. An automated system could solve the problem for many people. Such a system should be a perfect assistant, showing exactly those messages that are relevant or important to its user at that moment.

⁴Some requested information services are charged upon the requester, thus the recipient.

Since the interests of a typical user changes on both short and long term, the system will have to consider this [42, 62]. It should track the user's interests, and evaluate the activities the user employs at that moment. A decision to notify the user of the arrival of new communication should then occur when the user really wants to receive the incoming communication. In this way, interruption can be minimised, while timely delivery of important messages can be relied on.

Previous work mainly targets the problem of e-mail overload. Several solutions for automated e-mail filtering have been proposed (e.g. [10, 62]). These solution commonly take advantage of the fact that these messages consist of text. In an unified messaging system this cannot be assumed. A way to build a cross-media filtering mechanism, preferably a self-learning one, is needed.

3.3 Agent technology

Agent technology has been applied in many areas, among which telecommunication. Many efforts however, have been directed to telecommunication network management [24], which is of little visible value to end-users. Personal assistant agents for e-mail-handling have also gotten a lot of attention. Usage of (mobile) agent technology on the application level of telecommunication have been proposed [22]. Operational prototypes or implementations are less known however.

What is to be examined is the feasibility of using agent technology to solve the problems found in unified messaging and communication overload. Application of (mobile) agent technology has proven to be beneficial in several areas. Agents have been used for handling e-mail, as intelligent assistants. Other agents have been used to help tackle the problem of information overload [39]. Application of agent technology to the problem at hand seems possible, since it has been applied to several subproblems.

Remains another couple of questions. It is likely that agent technology can be applied, but what will be the benefits. Application of agent technology should be compared to regular techniques and other alternatives. Second, the question is how to construct such a system, in a way that maximises the benefits. Several possibilities (e.g. mobile, multi- or interface agents) can be utilised, but which combination will maximise the benefits.

Chapter 4

Goals

The goals of the project are defined here. Research, design and a prototype implementation are part of it. A full (commercial) deployable system is not a target however. The entire project will be approached from an *engineering* point of view, rather than as proving a *hypothesis*. An important remark is that receiving ones messages has the main emphasis throughout this study, for sending existing channels will be utilised.

Overall purpose is a system that enables unified messaging, while preventing the communication overload for the user. Unified messaging is fully universal in this context, allowing users everywhere and anyhow access to their communication. The prevention of communication overload is to be dealt with specifically *per user*, and should not be generalised¹. A fully automated solution is preferred, relieving the user of the management of the prevention system as well. As a last target, this has to be a solution based on (mobile) agent technology. Reality poses its limits on the proposed however, and has to be considered at any time. More detailed requirements needed to achieve these goals can be found in Chapter 9. To summarise along the lines of the defined problems, the goals include:

1. A unified messaging system, allowing any device, anywhere, anytime.
2. Prevent user from communication overload by means of
 - (a) Having a filtering system across different types of media.
 - (b) Deploying machine learning in this filtering to improve its accuracy over time.
3. Investigation of the applicability and benefits of agent technology herein.

¹Although collaborative (spam-)filtering and similar techniques should not be excluded.

4.1 Research

A research of literature on related topics has to be performed. This should form a foundation for the design, and prevent pitfalls already encountered by others. Incorporation of similar projects might also uncover other issues or opportunities. Last but not least, the benefits of applying agent technology to the particular problems have to be shown.

Architecture for unified messaging Architectures used for unified messaging in research and commerce can be utilised as a basis for the system at hand. Their coverage and maturity should be examined, in order to profit from previous research.

Benefits of agent technology Agent technology has been proven to be beneficial for several other problems. The benefits of the application of agent-based technology to the problem at hand has to be proven.

Algorithms for user profiling Quite an amount of research projects aimed for user profiling, many with handling information overload as goal. This led to quite some algorithms and systems, which can be useful here for automated — per user — solutions. An investigation of algorithms that can be used in this case is needed.

4.2 Design

Since systems of this size are rarely successful build from scratch, a proper design should be made first. This design should be based on agent-technology, and be generic. Being generic includes the possibility to apply future opportunities and means of communication. The design should thus be modular and extensible.

Agent-based architecture An agent-based architecture that enables unified messaging has to be designed. One important component of this design should be the capability to provide per user prevention of communication overload. This component should contemplate the user's situation and the priority new communication has towards that situation. Extension with new devices and media shall be another major component of the architecture. The incorporation of these new media and devices should not pose a problem for the prioritiser. Provision of data for the prioritiser shall therefore be taken into consideration throughout the architecture.

User profiling To accomplish the prioritiser mentioned with the previous item, a profile or preferences of the user have to be established. A proper algorithm or system to enable this functionality has to be included in the design. An important consideration for this profiler shall be the handling of unknown data. The prioritiser needs to operate under conditions unknown in advance, allowing new media and devices to be added transparently. Ideally, this profile should learn the profile from the user's interests itself. At least the learning problem needs to be defined, and possible algorithms should be investigated.

4.3 Prototype

Finally, a prototype based on the designed system should be implemented. The purpose of this prototype is to verify and evaluated the design and prove its feasibility. To limit the requirements for human-resources, a full implementation shall not be made. Some elements should be included however.

Architecture The overall architecture has to be prototyped for verification and evaluation. This does not state that the entire system should be implemented. Some of the elements can be simulated, or could be realised offering only partial functionality. This prototype has to account for several media and a couple of devices at least. Implementation of highly specialised software for certain media shall be avoided, but some non-trivial systems have a preference as well.

Underlying tools Some of the underlying tools should be implemented, to have a realistic meaning for evaluation. In order to prevent a similar evaluation to a theoretical one, the prototype should have some real life functionalities implemented. At least several media types and a couple of different devices should be usable, thus tools therefore will be needed.

User profiling The prototype should involve user profiling, since this is a likely candidate to attack one of the main problems. Implementation however can be proof-of-concept, rather than fully deployable. An algorithm has to be found that allows for heterogeneous data, that will not always be known in advance. This is required to allow new media and devices to be added without adapting the prioritiser. Unless easy applicable systems can be found, implementation, comparison and evaluation of several systems is estimated to be too time-consuming. Optimisation is left for further research and development for the same reason.

Use cases Examples of use cases that are likely to be covered:

- A fax will be send to the recipient's e-mail inbox when the recipient is in a meeting.
- An ICQ message ends up on SMS of an user while travelling.
- An attachment of an e-mail in an unusual format (e.g. a vendor-specific type) is converted to a format that is available on a PDA, and displayed on this PDA when applicable.

Due to practical issues, not all examples can be realised. Examples of scenarios not likely to be implemented:

- A voice-mail will not be send to a fax, due to unavailability of speaker-independent continuous speech recognition.
- An attached movie will not be displayed on a PDA, since these have currently only limited capacities, combined with high costs when used with a mobile data-connection.

4.4 Deliverables

The end result of the project will consist of two products. A report of the full project has to be delivered. Second, an implementation of a prototype is set as target.

Report First of all, a total report will be presented. This report will contain all results from the performed research, design and evaluation. Included will be:

- Research, whether agents are appropriate and a description of their (dis)advantages.
- Research, which user profiling method can be best applied for prioritising, or at least clearly delimited requirements have to be defined.
- Backgrounds, on related topics and decisions made throughout the project.
- Design, an agent-based architecture for handling unified messaging, including per user prioritising.
- Evaluation, of design and implementation.

- And of course discussion, conclusions and recommendations.

Excerpts of this document can be delivered when appropriate.

Implementation A prototype will be implemented, using Java and the Tryllian ADK as main foundations. This prototype will exist of an operational system, existing of usable components. It shall not be deployable for a large market of end-users, but usable nonetheless. Although it will be operational, it is not to be expected to be a full implementation of the design. It will contain most of the main elements at least, like the main components of the architecture and the (user profiling based) prioritiser. Functionalities with too heavy requirements will be left out or are simulated, to assure overall practicability. The entire prototype should be extensible, and the components can be reusable by themselves as well.

Chapter 5

Scope

From a practical point of view, it is not realistic to examine all of the issues that could be involved. A limitation of the scope of this project is necessary, and is made in this chapter. Not all choices can be made here however, since they might be unavoidably met during the project after all when excluded. Therefore, not all issues will be considered, but those that are likely to be usefully included in this limitation will be. Sending messages is left out of scope, since the emphasis is on receiving messages, as stated under “goals” before.

First, the issues will be briefly described in section 5.1. All issues that are explicitly included in the research are stated next. Last follow all issues that are explicitly *not* dealt with, in section 5.3. All issues which are mentioned, but not covered in the inclusion or exclusion, are implicit. These implicit issues are briefly discussed when encountered, or more thoroughly examined when needed for issues within the scope. Note that issues that are excluded still might be addressed when appropriate.

5.1 Issues

This section describes the elements that could be an issue in the system. The list is not exhaustive, but covers many of the main issues involved. It is intended to serve as a basis to determine the scope of the project. Below follows a brief list of issues that arise, afterwards some further explanation:

- Types of input.
- Meta-data.
- Interconnectivity.

- Retrieval.
- Media transformation.
- Centralised versus distributed.
- Prioritising.
- Situation tracking.
- Preference learning.
- Platforms and devices.
- User interface.
- Agents.
- Extensibility.
- Security.
- Billing.
- Relations with other projects / systems.
- Performance and scalability.
- Testing and verification.

Types of input Media are either synchronous or asynchronous, meaning direct (like telephone) respectively buffered (messages, like e-mail). Which of these types and their instances can be handled.

Meta-data Meta-data are descriptive data of a document or item. It should be closely related to the semantics and the value of a message with regard to the recipient. They do not represent the content itself, but give some characteristics. Examples are the sender's address, the size, a time-stamp, etc. . . Other examples include less obvious items, such as keywords, context or urgency.

This issue includes the way these meta-data should be gained and the relevance to the system. Some are easily acquirable (the sender has an e-mail-address or telephone number¹). Others require some processing (e.g. voice stress analysis to determine the urgency of a voice-mail).

¹available through CLID; Calling Line IDentification.

Interconnectivity Many devices can be interconnected, but in practise this causes some problems however. Not each device can be connected to any device, so there can be limitations to a certain group. Interaction between groups can be possible, but might be bounded by commercial policies.

Retrieval Some messages can automatically be forwarded to other systems, but some have to be retrieved. An example of active retrieval is the fetching of ones voice-mail by interaction. Passive retrieval happens for instance when e-mail is automatically forwarded to other addresses.

Media transformation Devices and media often have a close relation. Certain devices can therefore not display all media, and a transformation is thus required. Conversion can also have benefits for further automated processing. On the other hand can transformation mean a loss of informativeness (e.g. loss of non-verbal communication with speech-recognition of speech to text). When using automated transformation one should closely watch to maintain the same semantics of the message.

Centralised versus distributed A centralised system has a fixed point of connection and access. The system depends entirely on this central component. A distributed system might provide better availability, but requires synchronisation [64]. This can be a functional (from the user's experience) requirement, or a technical issue (on the level of implementation). Specifically the former could have implications on the latter.

Prioritising Messages are to be prioritised depending on the user's current situation, considering his or her preferences. For each message an appropriated action should be assigned [42]. These could include:

- Forward immediately to the user's current location.
- Inform the user about the arrival of the message.
- Store the message.
- Delay the message till a change in situation occurs.
- Summarise a collection of new messages.

Situation tracking In order to support the user with notification of messages that are important to the user's situation, this situation must be known. This situation could be provided for by the user itself, or automatically determined, e.g. by determining the location with GPS [40] and deriving the situation.

Preference learning Ideally, the prioritiser should be obtained by an automated adaptive profile of the user. In this way, the distributor should adapt to changing and unknown preferences of the user. An appropriate algorithm is required in order to automate a learning process. Greatest impact will have the data in this case, since messages can have great variety in content.

Platforms and devices What platforms and devices could be utilised. This indicates telephone, desktop, PDA, laptop, etc. . . Since certain systems have only limited resources, this results in boundaries for the system. Boundaries include limitation on available computational capacity and available media types.

User interface What kind of user interface(s) should be used. These are tied to the used platforms and devices. Different media might require different functionalities of the interface as well. The interface should incorporate all actions, but has to maintain its user-friendliness.

Agents The first question is whether agent-technology is applicable at all. Secondly, is a single agent the best solution, or is a collective of agents a better solution. One heavy-weight agent could have too big a footprint for proper mobility or light-weighted devices. Collaborative agents might solve this problem. Another choice is the usage of mobility. Mobile agents can offer great benefits, but can also form an unnecessary nicety. Advantages and disadvantages have to be weighted.

Extensibility Can the system be extensible, adapt for future possibilities and changing environments. The design should reasonably be adaptive to new emerging technologies and possibilities. Transformation between different media might be cumbersome and lacking nowadays, future research could enable several new opportunities. Advances in technology could offer several new ways of messaging and communication. A robust design can incorporate such advances in a fast and simple way. It would be preferable if these can be added dynamically, without interruption or modifications to existing parts

of the system. This will certainly be a requirement for a system with many users.

Security What are the consequences for security. When one receives all or nearly all communication through one channel, this forms an increased vulnerability. Legal issues may require some guarantees for the protection of privacy [5].

Billing What issues are involved on the commercial side, such as billing related to network- and computer-usage.

Relations with other projects / systems What other systems might profit from or interfere with this project. Other systems could be incorporated, or interoperability can be maintained.

Performance and scalability This includes computational as well as functional performance. The footprint of the system should be considered regarding mobility and resource-limited devices. Deploying a system for many users requires scalability [54, 56].

Testing and verification How to test the system on a functional level. This can involve some need for training and training data as well when the system has learning capabilities. A fully implemented and operational system can verify a proper design, but requires a lot of efforts beforehand.

5.2 Included

The items below are those that are explicitly part of the project. On completion, these items have to be evaluated or contained in the design.

Architecture and everything that is somehow related. The architecture is one of the most important elements of this project. It forms the basis in which to operate. This is a somewhat broad issue, since it involves many other sub problems, among which:

- Single versus multi-agent.
- Static versus mobile agent.
- Extensibility.

- Interconnectivity.
- Multiple source / provider.
- Media transformation.

Prioritising This is a major element given the problem of communication overload. Only those messages that are timely (high priority) should be presented to the user immediately. Therefore this component has to be embedded in the architecture. Prioritising should form the intelligent core of the system, allowing each user to have his or her own adaptive preferences. Therefore, the prioritiser is very likely to be realised with user profiling.

User profiling Given the requirement that prioritising has to be done *per user*, user profiling can certainly have a large impact, and should be investigated. At least the problem should be clearly defined, and an algorithm has to be found which can deal with heterogeneous data when possible. A way to initialise and improve this profile based on data is needed to be able to adapt to changing user's interest.

Agents The system has — in principle — to be agent-based. It is not a hype-founded requirement, but an item of research. Their benefits and drawbacks have to be shown, which can be best evaluated through their usage in the first place. In the architecture a choice for single, multi- or mobile agents has to be made.

Meta-data The usage of meta-data is considered important, since it is unlikely to deal generically with full message content. Handling full content is likely to disable the possibility for a generic prioritiser. Extracting meta-data in advance allows the prioritiser to handle a more generic type of data. Dealing with meta-data is therefore the preferred basis for adaptive prioritising. Extraction of these descriptive data should be implemented only when available, since they should not be required *per se*. Creating extractions themselves is not a goal of the project.

Platform device implementation Some media and devices shall be implemented to prove the extensibility of the design. Note that specific software for devices is avoided when possible, but feasible implementations shall be made. Since Java is the preferred language for implementation, devices supporting the Java Virtual Machine are likely candidates. Connected and controllable equipment can be utilise as well when available.

5.3 Excluded

All items below are considered out of scope. They are either too strongly related to the level of implementation, off topic or too time-consuming given their influence.

User interface Designing an user interface for an application of this size, could be a project on itself. The (necessary) interaction with the user is important however, given the fact this project is aimed at supporting the user.

User location tracking/situation determination Fully automated determination of an user's situation is a hard task, likely to require specific hardware sensors [60]. Since this project focuses on an user application, such hardware is avoided where possible. The user's situation will be used in the prioritiser, so has to be provided by the user explicitly.

Synchronous communication Direct communications (i.e. telephone) are left out of scope. The reason to do so is that they might impose real-time restraints. This unnecessarily complicates the system, given the fact that many heavy processing (e.g. speech recognition) can be involved.

Billing Billing is a complex issue, since it involves many parties and systems. Since a commercial system is not a target, commercial aspects will not be addressed.

Security No special arrangements for security will be made. Where possible, underlying security mechanisms can be utilised.

Performance Although resource limited mobile devices have to be considered, no special tuning or performance measurements will be made. Optimisation of code is left for commercial deployment.

Scalability Since a state of full operational deployment is currently not assumed, scalability for deployment with many users will not be tested.

Part II

Backgrounds

A survey of related literature has been conducted. Three areas are covered in three chapters. The first will describe unified messaging. Chapter 7 will describe user profiling. Last, chapter 8 describes agent technology.

Chapter 6

Unified Messaging

The concept of unified messaging is relative simple. As stated on the website of UnifiedMessaging.com [79]:

Access voice, fax, and e-mail from anywhere, using any device,
at any time.

Unified messaging is a common denominator for the combination of message based communications. These are often seen as fax, voice- and e-mail, but in this document any kind of message will be used. Thus news, Usenet, stock quotes, SMS, instant messaging and many others are included as well. A description of research, practise and developments in related topics will be given in this chapter.

6.1 Research

Since unified messaging is mainly a practical application, not many research efforts directly target it. Some projects exist however, and quite a few related areas that do have lots of active academical interest exist. A few related research projects and areas of interest will briefly be addressed in this section.

6.1.1 Unified messaging

Although research in unified messaging is scarce, a few (related) publications are available. Five of these will be briefly discussed:

Universal inbox As part of the ICEBERG-project at Berkeley University, the universal inbox was developed [56]. This project mainly had three goals; separate functionalities, allow any device on any network and give the callee

control. Included to enable this are any-to-any data transformations, user preferences and name mapping for any device. The data transformation are created by combining smaller conversions using what is called *automatic path creation*. User preferences are created through a simple rule-based system, where the rules are assembled with a graphical user interface. The naming service is hierarchical and closely related to DNS [49].

Mobile People Architecture Another project aimed to develop personal reachability. At Stanford University the Mobile People Architecture [58] was developed, to make *people* reachable, rather than devices. Important alternate target was to maintain the user's privacy, by not revealing its location. The proposed solution is an extra layer with an addressing scheme on top of existing network models. Each user gets its own address, which can be translated by the system to device-addresses. A *personal proxy* tracks the user's current device, and connects the incoming communication to the actual device. Conversions of media and a rule-based system are included in this design as well.

Active Messenger Marti [41] has studied the use of multiple devices to maintain reachability. Basis of the design is measuring the user's reaction. The device used by the user's last interaction is registered. If new e-mail arrives for the user, the user is notified on the most recent used device. A personal schedule can be incorporated, for more specific treatment of messages. Included in this schedule can be alternative devices, that are automatically used if the user does not respond within a certain interval.

Universal Messaging Agent Lauff et al. [36] describe another system. Their architecture includes device specific protocols, to formulate requests for information. This is mainly used by the user to actively request information. Users can include in their request how they want to receive this information (e-mail, phone, fax, ...).

UniMail Last, Yeo et al. [87] describe a system for unified messaging. In this design, each device has its own handling module. Aside from text-to-speech, there are no conversions described in the design. They provide a quantitative comparison with some commercial alternatives as well.

6.1.2 Media processing

Other areas of research that are related to unified messaging are those used for media-processing. Some of the related are:

- Speech recognition.
- Speech synthesis.
- Optical character recognition (OCR).
- Human expression recognition.

Below they will be described, including the reason why they are of interest.

Speech recognition Recording speech and digitising it allows computers to process it. Although speech is a very natural way of communication for humans, it is a complex task for computers. Most speech recognition aims to recognise commands or the actual text. Several problems still exist, which are actively researched [28].

- Speaker independent. Creating speech recognition that can be used for any user is a hard task. Different people have different pronunciations, complicating recognition.
- Language and dialect. Many languages and even more variants thereof, dialects, exist. Each has its own characteristics, requiring adaptation or even entirely different methods.
- Continuous recognition. Recognition of normal speech includes an enormous possible dictionary. This is also known as free speech, recognition not bound to a certain vocabulary or scenario. If this is to be used interactively with the speaker, real-time problems become involved as well.
- Noise and disturbances. Environmental sounds, static in recordings and other people speaking in the background can further complicate recognition.

Speech recognition is relevant for unified messaging, because it can allow users to read a spoken message (e.g. voice-mail). Partial speech recognition can also be used, for example to determine important keywords.

Speech synthesis Also known as text-to-speech (TTS), speech synthesis is commonly the reverse of speech recognition [28]. For various languages, computers can rather believable pronounce texts. Several different speakers (“voices”) are available as well for most of these languages [3]. Speech synthesis is important for unified messaging, because it can allow users to listen to typed (or written) messages.

Optical character recognition OCR is the process used the distill text out of images. It is commonly used after a document has been scanned. Another application is often found in a Personal Digital Assistant (PDA, also known as handheld). PDA’s equipped with a touch sensitive display or special pen can be written on. The commands or text entered this way are recognised. OCR can allow images, such as from a fax, to be transformed to text, usable for non-graphical devices, such as SMS.

Human expression recognition Another technique still subjected to research is recognition of human expressions. This is applied to video, mainly for facial expression recognition [53]. It can also be applied to audio, based on differences in speech. Although not directly related to unified messaging, it will become clear later how this can be useful.

6.1.3 Standards

There is no standard way to achieve unified messaging. Neither are there any standards how a unified message is to be represented. There are some standards that are often used for unified messaging in practise. Several standards that are often used¹ in relation with unified messaging will be described. Note that their is work in progress, but at the time of writing many of these efforts are not yet completed or have hardly any implementation [31].

SMTP Although e-mail itself is not a standard, there is very common standard to facilitate the concept of e-mail. This is the Simple Mail Transfer Protocol (SMTP) [55]. It is a client-server system based on a separation of the application for the user, and a network of servers for delivery and routing of messages (often called user-agent and mail-transport-agent resp.). Exchanged messages are normally of the format defined by [11].

¹Aside from protocol specific standards, such as those used for the telephone-networks, fax and SMS.

MIME The Multipurpose Internet Messaging Extension (MIME) [18, 19] is an extension to the old type messages [11]. The main limitation of these initial type of messages was their strong orientation towards plain text only. One of the main goals of the extensions is therefore to allow other type of messages (e.g. image, audio) to be send as well. Several media can be combined as well, the resulting parts of a message are commonly named *attachments*.

Specialisations Further specialised variants exist of the above, for various means. The Internet Fax is a file format for the exchange of faxes through e-mail. It is mostly a variant of the TIFF format [43]. VPIM [80] is a restriction of MIME, specialised for the exchange of voice-mail messages between different voice-mail systems. Particular the Voice Profile for Internet Mail is still work in progress and quite experimental however.

6.1.4 Related

Some other areas than those mentioned earlier in this section have a relation with unified messaging. How, will be explained below.

Ubiquitous computing Research for ubiquitous computing aims to provide continuous access to computing facilities. This is mainly achieved by creating networks of small points of access, in many places. Another approach is that of wearable computing, where computer elements are interwoven with clothes or are wearable as artifact [60].

Natural language processing Another one — perhaps the holy grail of artificial intelligence — is natural language processing (NLP). This is the area of interest to create computer programs that can process, or even understand, human language. It is highly complicated for computers, since humans often use a lot of implicit context. Natural language is very ambiguous as another complication.

6.2 Practise

Most implementations of unified messaging can be found in practise nowadays. In the era of *dot-coms*, several companies started to offer free unified messaging. After the collapse of everything freely available on the Internet, the *dot-bomb*, most of these services have to be paid for (e.g. [86]). Other services are specific to a certain provider, and are combined with a subscription.

Another trend that has been identified is the offering of unified messaging as a service to companies, rather than individual customers [32].

Most of the companies that provide unified messaging, only limit these to specific media (e.g. [27, 86]). Commonly used are at least the first three items, the latter are more rare.

- Fax.
- Voice-mail.
- E-mail.
- SMS.
- Instant messaging.
- Paging.

One can access these mostly by:

- Receiving them as e-mail.
- Receiving them as fax.
- Using a website.

Note however, that not all combinations of the above are supported. Many providers only offer restricted possibilities, not full interconnectivity (e.g. [46]).

Other companies provide only elements of the required infrastructure. Companies like Cisco, Siemens, Alcatel, Ericsson, Lucent and many other large and small companies are involved [79]. They often provide infrastructure ranging from network equipment to establish the required connections to complete applications.

6.3 Developments

Aside from developments in academic research, several developments take place in practical areas as well. Some important developments, as they are related to this research, shall be listed below.

Networking technologies Two important developments can be found in networking technology. Both mobile and fixed access to Internet become faster, and can be continuously connected without additional costs. Another item is wireless networks, where speeds start to approximate those of fixed in-house networks.

In the field of mobile networks, both i-mode and GPRS are available nowadays. I-mode is an improvement to the mobile phone, allowing multimedia content from specially adapted websites. General Packet Radio Service (GPRS) is an improvement of GSM, improving to “always-on” networking at circa 5 times the data-rate of the original GSM-standard. Both are (becoming) available in production level provider networks. Operators of mobile networks are implementing UMTS-networks, the third generation (3G) mobile communication.

For continuous broadband Internet access, the consumer-market has shown progress as well. Cable operators provide 24-hours a day access over the same cable used for television. Telecom-operators offer Digital Subscriber Lines (DSL)², also for continuous access. Both can not only operate around the clock, but also at speeds significantly faster than traditional dial-in connections.

A last development is found in the wireless networking market. New standards, like the IEEE 802.11 specification, allow wireless networking between computers. Speeds approaching those of fixed in-house networks are available without the use of any cable. These allow full multi-media access around your house or company. Wireless networks should not be mistaken for mobile networks. A mobile network is provided by a public operator, normally in a large area. Wireless networks are used in private networks, using a base-station and normally limited to an operational area of a few hundred meters.

Communication devices Not only network connections have shown improvements lately. The devices — particular the mobile and wireless ones — people use to communicate have shown a lot of progress as well.

First of all, mobile phones are available on the market nowadays with an integrated digital camera (e.g. Nokia 7650 [51]). This allows people to send images directly to be published on the Internet or to other users.

Another development can be found for the Personal Digital Assistant (PDA). These handheld devices now come with mobile data access integrated. GPRS capabilities (e.g. Siemens SX45 [63]) are integrated, allowing one to

²Asymmetric Digital Subscriber Line (ADSL) is currently the most used. Improved versions like Symmetric DSL (SDSL) are still under development.

read e-mail and browse the web.

A last item described here, is the availability of phones with an integrated Java Virtual Machine. These devices are equipped with Java, so new functionalities can be added. Mobile phones and other small mobile devices can thus be extended with nearly any (small) application.

Chapter 7

User Profiling

User profiling is the attempt to create a profile of an user. This profile can be used to select information that can be of interest, or withhold information an user wishes not to receive. These fields are known as respectively information retrieval and information filtering (IR and IF). In this section a motivation for and an explanation of user profiling is given. Some related areas will be discussed as well.

7.1 Motivation

Most people who have searched for information on the Internet have a shared experience. When using one of the search-engines available on the web, you can type a few keywords and all kinds of related websites and -pages are listed. With the increasing popularity of Internet, the number of available pages grew as well, resulting nowadays in enormous amounts of *hits* for any query with a few common keywords. This problem is called an information overload. As earlier stated, receiving lots of e-mail and other messages can lead to a communication overload [26].

Nowadays there are many research efforts in the field of Information Retrieval and Information Filtering (IR & IF) [23] to attack the information overload. The first is used to find relevant information in a vast amount of information, such as a collection of documents or a database. The latter is used to reduce a stream of information (like a news service or e-mail) to relevant items, or classify these into related groups. Another important difference is that with retrieval an user applies a specific query to a relative fixed database, while filtering is based on a long-term *profile* of the user and used on more dynamic information. More commonalities and differences are described by Hanani et al. [23].

Particular interest in this study goes to information filtering. In order to prevent an user from being overwhelmed with messages, messages either have to be removed or classified whether they are important. Messages that are not important can either be stored without disturbing the user or just deleted¹. Since ones (personal) communication is a long-term issue, creating an user profile does not seem an obstacle. The fact that e-mail is often used as example confirms this assumption.

7.2 Information filtering

Information filtering is often applied to classify e-mail. Another well-known application is to filter web-pages, after a simple search resulted in a large amount of results. This section will emphasise the former, for obvious reasons.

7.2.1 Global technique

The generic idea behind information filtering and user profiling is a relative simple one. All it takes is some documents already classified by the user to compare new data against. As described by Hanani et al. [23] in their overview, a generic system to accomplish information filtering contains a few components:

- Data analyser.
- User model.
- Filter.
- Learning component.

The data analyser creates the items to be filtered, and ensures it is in a normalised, processable format. A user model is the actual profile of the user, representing the user's interests. The filter compares a new item against the model and classifies the information. At last, the learning component is used to improve the user's profile, based on feedback provided by the user. This has to deal with changing interests of users over time as well.

How all these components are actually filled in depends on the application. Some data analysers are created for a particular domain, and can thus include some preliminary knowledge. Other systems have a more open

¹Which is essentially the same as storing them separately, and never look back to them.

area of application, and need more generic data analysis. The same applies for the learning component. Aside from various inputs (the data), it can have different required results and expressiveness of feedback. The latter can for instance be a simple wrong or right, or can include the exactly defined preferred result.

7.2.2 E-mail filtering

Communication overload is effectively a generalisation of a well-known problem, *e-mail overload* [82]. An application of information filtering that has received a lot of attention over the last few years, is e-mail filtering. People receive so much mail, directly from other people, mailing-lists, unsolicited (also known as spam or junk-mail) and viruses as well. This poses two main problems:

- Important messages get lost in the flood of less important ones, quickly reading ones e-mail becomes impossible [70].
- Organising ones mail for later usage (archiving) takes a lot of time [62].

Both of these problems have been addressed by several people. Not surprisingly, these problems are often handled in a very similar, if not equal, way. The main task to fulfil is after all classifying messages. An important difference is commonly the result; message are important or unimportant, or message belong to a certain subject (and the related mail-folder). A few of the regular manners are discussed below, although it is emphasised nearly all of these only consider (plain) text e-mail. Very common words in normal natural language are frequently removed, to avoid classifying non-informative words (common words like *the*, *an*, *is*, ...) that are mainly used for linguistic purposes.

Static rules One of the first and, at least in production-level programs, most wide-spread solutions is not all that fancy. It simply consists of a set of rules, that determine the faith of the message. The user just creates a rule that needs to match certain criteria (e.g. from `mailinglist@company.com`), and a target (e.g. delete, save in a folder named work or forward to another e-mail address). A classical example is `procmail`, where an user has precise control by writing rules according a specified syntax [2]. E-mail applications with rule creation tools, integrated in the graphical user interface, exist as well.

TF-IDF Another, in literature often encountered, solution is a self-learning one [4, 62]. Term-Frequency Inverse-Document-Frequency (TF-IDF) and its many variations are based on the usage of keywords. A new document is compared to existing categories of documents, classified by the user (commonly existing e-mail folders). It can be summarised as follows:

1. Select the most frequent used words in the document to be classified.
2. For each of these words, the relative frequency is determined.
3. Each target category has its own characteristic series of keywords and corresponding frequencies.
4. Now select the category with the keywords that match best, this is the resulting category.

Bayesian approach Other learning mail-classifiers are based on a statistical principle founded by Bayes [38, 57, 70]. This is based on the probability that a word appears in a text. Existing categories have such a probability for frequently used words. Now these probabilities are combined using Bayes law, under the assumption the words are used independent of each other.

Others Other algorithms are used as well. Rule learning systems have been developed, based on rule-learning algorithms [10]. Another approach was used to automatically create rules as well. CLUES attempts to create rules from ones calender, reply-headers in e-mail, area codes of telephone-numbers and domain-names [42]. The well-known nearest-neighbour algorithm has also been applied [48].

A last project worth mentioning is the application of Takkinen and Shahmehri [70]. Although mentioned before as a Bayesian system, this is just one of the possible algorithms. The application actually has three modes (busy, cool and curious), all using a different algorithm. Depending on the user's state, another algorithm is applied. The actual filtering mechanism thus depends on the user's state of mind.

7.3 Machine learning

Ideally, an user profiling system is self-learning. Therefore, some basic algorithms used for machine learning will be briefly described. This should give an impression of their working and properties. Note that this summary is not exhaustive, but an introduction to algorithms that can be useful for the

problem discussed in 13.1. Many other algorithms exist, but they are either too advanced for this section, or are considered inapplicable for the problem.

The essence of a learning problem normally is the classification of an object based on its attributes. Previous instances or user classified objects form a set of known samples. The learning algorithm determines the classification of an object based on a direct or indirect comparison with these known samples. A strong relation of machine learning exists with statistics and pattern recognition.

7.3.1 Nearest neighbour

A very well-known algorithm is the nearest neighbour method. In a standard nearest neighbour method, some way to measure a *distance* between objects is required. Now, the distance between the object under evaluation and all available examples is calculated. In this setting, the k nearest objects are chosen. The classifications of those samples determine the applicable classification for the current object. In case these classifications are not unanimous, the most dominant classification is used.

Many other variations of this algorithm exist or can be made and are used for various purposes. A further description can be found in [47, chapter 8].

7.3.2 Decision tree

A decision tree is a classification based on a hierarchy of the attributes of the objects. Each node in the tree can have branches, for all objects that have a certain value for the attribute. The leaves contain the classification of the objects that have all attributes along the established path.

Decision trees are often used to model human knowledge. One of the advances is that decision are established hierarchical. This allows a strict distinct behaviour between different values of a specific field. Another advantage is that the most distinctive fields can be evaluated first. It also allows for easy explanation of a decision afterwards. The downside of these algorithms is the possibility of enormous trees that are hardly readable anymore.

Algorithms like ID3 and C4.5 establish this kind of learning [47, chapter 3]. These algorithms normally work based on statistic grounds. They try to establish a decision tree top-down. Therefore, they determine the attribute with the highest information gain, that is the one that creates the best separation within the set. This attribute forms the top node of the tree. For all branches, this is repeated on the resulting subset. Now iterate over the set of samples until the tree covers the entire set.

7.3.3 Rule-based generated by clustering

A rule-based system is based on a sequence of rules. A rule consists of one or more specific attribute-value combinations. Classification happens by trying rules, the first one to match determines the result. Although this is closely related to decision trees, there are a few differences. Most important is increased expressiveness, and deduction of new rules [47].

RIPPER is an example of a rule-learning algorithm [10]. More and various algorithms exist, that operate based on various approaches. A broader description can be found in for instance [47, chapter 10].

7.3.4 Bayesian learning

A last approach discussed here is the one based on mathematical probabilities. In the simplest variant, it creates a probability for each attribute. When a new object has to be classified, all its attributes are examined. The result consists of a probability for each possible classification based solely on that attribute. These probabilities are combined over all attributes according statistical principles. The result is a probability for each possible classification.

Normally the most probable result is used as classification. Variants exist that result in multiple classifications, each with a probability. Another type are the Bayesian belief networks. These networks include a representation of prior knowledge in the calculation, instead of assuming that all attributes are independent of each other. Further descriptions of all of the above can be found in [47, chapter 6] and various others.

Chapter 8

Agent Technology

One of the first questions many people ask, will be what an agent is. In this section a brief introduction in some aspects of agents will be given. It mainly presents an overview of literature and practise. Included are some examples of agents as they are applied to, or suggested for, (tele-)communication.

8.1 Perspective

The question “What is an agent?” does not have a simple answer. There is no generally accepted definition, and among different groups, different ideas exist. As Nwana [52] said, agreeing to a definition for agent is as impossible as reaching one for artificial intelligence. Bradshaw [6] points out that almost all views of agents at least imply the use of the term agent as a metaphor. This is the personification of agents.

Several people have proposed different classifications as well, for the different types of agents. Some of these will be discussed below, and used as a basis for the perspective used throughout this document. Make clear however, that an agent as discussed here is a *software* agent. Explicitly *not* mentioned are agents as they are used in chemistry, biology, human society or any other discipline.

8.1.1 Properties

Several researchers have proposed to delimit what can be called agents, and how different types of agents can be distinguished. A few distinctive features are commonly used in these classifications. The classifications themselves are discussed in 8.1.2. Among the characteristics of agents are:

Intelligence Agents can have a certain degree of intelligence. Most commonly this is directly related to techniques from artificial intelligence (AI). Included are elements like expert systems and machine learning [59].

Autonomy Another important characteristic is the autonomy of an agent. Agents have autonomy if they can act without external guidance of an user or operator. They do have to act on their experience and perceptions [59]. Another aspect of autonomy is their ability for self-regulation. Where an object is commanded to perform an action, an agent is requested and may refuse, so it has control over its own state [83].

Reactive An agent can react to its environment and other agents. An agent therefore needs to perceive its environment. This can be done through physical sensors or a software equivalent if the agent has no direct physical representation. Reactive behaviour is often combined with the sense-reason-act loop [73].

Multiplicity An agent does not have to be alone, systems with multiple agents exist as well. These agents then form the entire application.

Cooperative and social When multiple agents are used, they can operate in a collaborative manner. In this way all agents can work together to reach a common goal [52]. The opposite would be for instance a game, where agents work against each other. Closely related, but not necessarily compulsory, to the above, is the ability to communicate with other agents. Aside from manipulating and perceiving its environment, agents can be able to communicate with each other [84].

Mobility A software agent is not necessarily tied to a single computer system. When an agent is capable of transportation to another system — this transport includes code, data as well as its current state — the agent is called *mobile* [35].

8.1.2 Classification

Different classification schemes of agents exist. Most are based on one or more of the previous declared properties. A few of the most important are given below.

Multi-Agent Systems Systems with multiple agents that cooperate and communicate, form multi-agent systems (MAS). In these systems, multiple agents exist, all with their own *role* in the system [85]. Based on these role(s), an agent has permissions and responsibilities. These define what its behaviour needs to be.

Mobile Agent Technology Another important field is that of mobile agent technology (MAT). Mobile agents are agents that are capable of migrating themselves [35]. They can move including their data, program and current state. This allows mobile agents to be independent of the hosting system, but can be anywhere “in the network”. There are several reasons to use mobile agents [9, 35], some of which will be interesting during the research at hand:

- They reduce network load. Mobile agents can interact at the remote location, they only travel once with the complete results.
- They are robust. They can recover from errors to find alternatives when and where they are needed.
- They encapsulate protocols.
- They are heterogeneous. They can be run throughout the network, independent of the hosting platform.
- They execute autonomously and asynchronously.
- They can operate when their initial host is off-line.
- They can introduce and use new code on a system¹.

Although there is an alternative solution for almost each benefit, it is the combination of benefits that is unique for mobile agents [9]. Mobile agents can have various other agent properties as well.

Multi-dimensional typologies Other classifications go further than one specific property. Bradshaw [6] describes two typologies both based on three properties of the previous list. The first is by Gilbert et al. who see agents characterised in three dimensions:

- Mobility, from static to mobile object.

¹It is said this is done by computer viruses as well, these can be seen as malicious mobile agents, and systems can be protected against these.

- Intelligence, from preferences to learning.
- Interactivity towards its environment, from asynchrony (non-interactive) to service interactivity.

The second is given by Nwana [52] and uses different characteristics. The (main) properties used here are:

- Cooperation.
- Learning.
- Autonomous.

Various types of agents combine several of these characteristics.

8.1.3 Agency

In this document a description of agent similar to those used by the multi-dimensional classifications will be used. Rather than using a single definition, the term agency is applied (partially after [6, 84]). Software with a combination of the listed properties is said to be agent-based. The agent or agents herein have a certain “degree of agency”. More or stronger presence of properties increase this degree².

8.1.4 Practise

A last note in this section will be with regard to agents in practise. Not only various applications and systems have been created as based on agents. Several academic as well as commercial implementations for agent *platforms* are available. These can be used for the development of agent based applications. They provide developers with some or most of the basic elements needed to build agent-based systems.

Most of the existing platforms emphasise on one or more specific agent characteristics. The *Agent Development Kit* (ADK) offered by Tryllian is one of the commercial platforms. The Tryllian ADK emphasises on mobile agents, but supports communication for cooperative multi-agents as well. A further description can be found in appendix A or on <http://www.tryllian.com/>.

²“All agents are equal, but some agents are more equal than others.”

8.2 Agent communication

Agents in a multi-agent system that have social capabilities can communicate with each other. There are several ways agents can communicate. Communication principles consist of two major components: language and architecture.

8.2.1 Languages

The language is a definition of the *way* agents communicate (what they “say”). Two languages will be discussed:

- KQML.
- FIPA ACL.

KQML From the ARPA Knowledge Sharing Effort (KSE) came forth the Knowledge Query and Manipulation Language (KQML). To enable agents to exchange their knowledge and intends, KQML was designed [14]. The communication of agents is strongly based on the communication of humans. Derived from human conversation are *performatives*, the actions an agent can attempt in communicating. KQML messages are carriers of the content, they do not represent any knowledge themselves. Other definitions are needed to define the contents of the message.

KQML has evolved from a language to exchange information to an agent communication language. It was developed from a theoretical approach in which an agent was just an entity, not a technological concept as used nowadays. The definition of KQML lacks a clearly defined semantics, since it its was initially developed as *syntactic sugar* [34].

FIPA ACL The Foundation for Physical and Intelligent Agents (FIPA) has developed the FIPA Agent Communication Language (ACL) [16]. FIPA started from the agent technology point of view, rather than that of knowledge. One of the main goals is to create standards for agent technology. These standards are much broader than the communication alone, ACL is one of them. FIPA ACL uses some of the basic ideas of KQML, but has better defined semantics to overcome differences in interpretation.

The syntax of KQML and FIPA ACL are almost identical. Their differences can mainly be found in the semantics. An extensive comparison of both can be found in [34]. FIPA emphasises a more pragmatic ACL, in combination with various other standards. These include for instance standards for agent management as well.

```

(request
  :sender      MyAgent
  :receiver    ServiceAgent
  :content
    (process, headers, message)
  :language    simple-request
  :reply-with  processed
)

```

Figure 8.1: Sample FIPA message (fictive)

8.2.2 Architectures

An architecture defines the *method* agents exchange their messages (the means they use). Two mainstream architectures will be described here:

- Blackboard.
- Messaging.

Blackboard In a blackboard system, agents communicate through a shared medium where they can deposit their message, a so called blackboard [25]. Each agent can place a message on this blackboard, and read the messages placed by others³. Blackboard systems are particularly useful if agents need to maintain a shared state. Every agent can read or update this state when it needs to. There are no restrictions on the shared information and central control is required [29].

Messaging Another approach for agent communication is the use of messaging [25]. In these architectures agents send messages to each other in an asynchronous fashion. A message has to be addressed by the sending agents to the recipient agent. If the agent expects an answer, it has to include its own address as well. An infrastructure to support messaging is needed of course.

Messaging is asynchronous, so an agent is not blocked by communication. After a message is sent, it can continue with other jobs it has to do. Receiving agents are reactive to the incoming message. Agents can set up a conversation, if they intend to reply to each other. Variations exist where groups of agents communicate, or an agent can broadcast a message. The messages are most commonly based on either FIPA or KQML.

³Unless some form of specific access control has been added.

8.3 Agents in (tele)communication

Agents have been applied, or suggested, to problems in telecommunication [24]. Various types of agents are included in these applications. Many (mobile) agent solutions are applied for network-management and -monitoring. These will not be described, since they are not of interest in the current project. A few applications will be described briefly:

8.3.1 UMTS/VHE

The development of mobile telecommunication has received a lot of attention during the last decade. One of the recent developments is that of Universal Mobile Telecommunications System (UMTS) [77]. An important element of this third generation of mobile telecommunication is the Virtual Home Environment (VHE). With such a Virtual Home Environment, users will be able to access their mobile communication on any device, always receiving (almost) the same services and look-and-feel. This VHE should be customisable by the user in all aspects. Several researchers have proposals for using agent technology to accomplish this VHE.

Mobile agent One solution encountered to solve some of the problems involved in the VHE-concept is based on mobile agents. Important reasons to use mobile agents are to allow off-line operations and dynamic use of new software [12]. Some of these solutions basically come down to a collection of agents [12, 22]:

Terminal Agent A terminal agent represents the capabilities of a device. It also provides initial access to the system for the user.

Service Agent Services available in the network are offered by service agents.

Provider Agent A provider agent forms the centralised coordinator. It offers and combines the different available services.

VHE Agent The mobile agent in these settings is the VHE agent. This agent represents the user, and migrates to the device upon request. It adapts to the terminal agents capabilities, and communicates with the provider agent.

Multi-agent Another solution is mainly based on (negotiating) multi-agent systems. Lloyd et al. [37] propose an architecture where the VHE is represented by a set of agents. A controller agent has central command over

other agents providing services, routing and emulation (for look-and-feel). Network and terminal agents encapsulate devices and network.

Fujino et al. [20] describe a multi-agent system where agents are used to separate communication and personalisation. A personal agent is used for customisation towards the user. Each location can have a clone of such an agent, which are to be synchronised with a central instance. A network agent is provided to create transparent access to the network.

8.3.2 Intelligent Network

Agents are also proposed to support Intelligent Network services [33]. Kerr et al. describe a system called PANI, to handle intelligent networks based on agents. An intelligent network allows users to use services, rather than network operations. In PANI, three main types of agents are used.

Event agents Event agents are providers of events, pieces of new information, that are available in the network. Their prototype includes news, stock quote and e-mail.

Action agents PANI's action agents form the outlets for this information. They deliver the information to the user by e.g. e-mail, telephone, -fax or SMS.

Personal rule agents The third type of agent is the one connecting the former. A rule agent combines events with actions, possible with extra conditions. One of their examples is informing the user by SMS, when a specific stock quote exceeds a predefined limit.

8.3.3 IMPAX

A last application that will be mentioned, is IMPAX. IMPAX stands for Intelligent MESSaging with Personal Agents and XML. It is described by Meech et al. in [45].

IMPAX is a multi-agent based approach to handle unified messaging. It is based on XML and KQML. It consists of a central controller agent, along with several others:

Service Adapters Specific forms of telecommunication, available through *gateways*, are handled by what are called service adapters. These act as two-way gateway, feeding messages to the system and as delivery channel as well.

Converter Agent A converter agent provides conversion of content from one format into another.

Personal Communications Agent The personal communications agent represents the user. It coordinates with the central *messaging manager* the capabilities of the user's devices, and preferences through rules.

8.4 Alternative technologies

Agent technology is not the only approach to the problem at hand. Several other popular approaches exist as well. A few alternatives are proposed and discussed below.

Client-Server A wide-spread practise nowadays is that of client-server. In this concept a client system connects to a central server system, using a specific, predefined, way of communication. The functionality required in this case can be implemented using conventional client-server. Although client-server is well-known, it has a couple of disadvantage however:

- Centralised. Client-server is in general a centralised approach. This makes a client-server setup vulnerable for network failure, dependant on continuous connections and repeated transmission of the same data. Further details on these subjects can be found in section [11.3](#).
- Less personalised. Using centralised servers shared among multiple users, it is much harder to let these behave differently per user. Although servers can easily be adjusted to differentiate between users, behaving very differently for separate users is a hard task. Either many servers are required, or very heavy server-applications need to be used.
- Less extensible. When dynamically extending the system across multiple networks, client-server requires a direct connection. This can be resolved by using intermediate servers (proxies), but this requires specialised intermediates for each application at each connection between networks.
- Less adaptable. The process in a client-server setting is very restricted to that offered by the server. Supporting various alternatives requires all servers to be altered, and is thus very unpractical.

All these arguments are generic for comparing client-server with (mobile) agents, but they apply to the particular case at hand as well. One should

keep in mind that agents are a concept, that can be implemented using some technology. Client-server on the other hand, is more a technical architecture. Many additional measures upon this initial client-server concept have been developed. Agents can be implemented upon a client-server system, thereby creating a large set of such additions itself. Evolution of these concepts and technologies cause the existence of vague boundaries. As with the term agent itself, a clear distinction is thus often hard to make.

Web based Although it is a specialisation of client-server, many interest goes to “web-enable” existing applications nowadays. Using standard web-browsers and related products seems a reasonable approach at first. It suffers from the same disadvantages as a client-server based system, and:

- Media dependant. Although web browsers support all kinds of media, most web based applications are very graphical oriented.
- Continuous requests. Web based systems are primarily designed to be user initiated. A communication application should thus continuously reload a web-page to check for new messages.

Part III

Concept

In this part, a description of the developed concept is given. It starts with stating the requirements in chapter 9. Next, chapter 10 describes a functional design. Based on these, chapter 11 motivates the selections and choices made. This selection leads to the architectural design of the proposed system, in chapter 12. Last chapter of this part (13), includes some anticipations towards implementation and future developments.

Chapter 9

Requirements

Before a design can be made, the domain of the design should be given by requirements. Those functional requirements will be described below. They are divided in three categories, representing whether the requirement must, should or could be present in the resulting system. Last follow a few assumptions that were used for the design.

9.1 Must have

Some issues need to be present in the system. These are the elements that *must* be included. Below are given those that cannot be neglected:

Media transformations An unavoidable feature for a system with unified messaging is of course transformation of different media. For many users this will be quite useless if it cannot be done automatically.

Message prioritising In order to prevent an user from an overload of communication, incoming messages need to be prioritised. This should be done in accordance with the user's current situation. Implied hereby is that this situation is known, so this should be tracked in some manner.

Distributed / mobile The promise to be reachable anywhere, anytime, can only be accomplished if users can connect from many places. What follows is the implication that users must be able to use mobile devices, and / or can use multiple points of access. Therefore the system must have a distributed nature, or at least appear so to the user.

Multiple device To be able to provide unified messaging, all sorts of devices must be enabled and connected. These devices can be found at both sides of the system. Input devices provide messages for the system, like receiving a fax or e-mail. On the other side, a device such as a desktop or phone provides access for the user to ones messages. Multiple instances of both, and of several types, have to be possible.

9.2 Should have

While the previous issues cannot be loosened at all, the following requirements are less hard. These items *should* be confined in the system, with a high preference to indeed do so. When unavoidable, a compromise between some of the following might be needed:

Storage and retrieval Similar to traditional e-mail and other systems, the system should be able to store messages for the user. These should be accessible later, both as an archive or e.g. to mark follow-ups on the message.

Preference learning All messages must be prioritised for the user, based on its interests. Although the user could formulate these interests itself, it would be very beneficial if the system automatically can deduct these.

Robustness Because communication networks are relative open and unreliable, unpredictable input can occur (e.g. due to malfunctions or malicious use) or connections may fail. Therefore, a robust system which can cope with many types of incidents will have strong benefits.

Multi-user Although each single user can be equipped with its own system, a system for multiple users can save useless redundancy. Sharing parts of the system becomes a possibility as well, reducing hardware-requirements and allowing to use other people's devices. This particularly applies for media that are very often shared already (e.g. a facsimile device). Furthermore, it will save many users from involved maintenance and upgrades.

Cross-media All of the requirements made apply across all included media. Wherever possible, all features need to be independent of the used medium. Of course this is impossible and illogical for the media-specific input- and output-devices, but applies to all other components.

9.3 Could have

A last category of requirements are those that are optional. These requirements are not essential for the targeted functionality, but are very welcome nonetheless.

Platform independent The more portable a system is, the easier it is to develop and use. This means that new devices require less efforts to be embedded in the system, and appear more similar towards the user. With platform both the hardware and the software are mentioned.

Adaptable For more advanced usage of the system, additional measures can be taken to allow easy modification. Parts of the system could be altered, without a complete renewal of the other existing parts. When a multi-user system is created, different users could even alter a specific part to their own needs in this way.

Secure Since communication can be confidential, and should therefore not be exposed to others than the involved parties. Another specific implication arises from the law, specially in a multi-user system, towards the privacy of the user [5].

9.4 Assumptions

Of course a system cannot be only limited by must haves and should do's. A couple of assumptions are made here, to allow some degree of freedom. Another goal of these assumptions is to give a reasonable basis for the design.

Network The first assumption made is the presence of an interconnected network. Note that this not necessarily means *one* common network, only the presence of access to *a* network, that is *somehow* connected with the rest. Parts of the entire network may be heterogeneous, and a decent backbone (no noteworthy delay or bandwidth limitations) is expected. Devices are connected to this network as endpoints, but these may be so through limited connections.

User A second assumption applies to the user. This user is assumed to be willing to cooperate¹, for its own benefit (avoiding communication overload with availability of communication). The user is thus assumed to be willing to provide some necessary information.

Capacity The last assumption is made on the capacity of devices. They are assumed to have a reasonable capacity available, thus avoiding the presence of very hard constraints on available resources. It does not state the absence of limitations, light-weighted devices *do* exist. For a task that requires more resources, a proper machine should be available, thus the end-device should avoid possible heavy tasks.

¹This implies another requirement; the user must be *able* to cooperate, through e.g. interaction.

Chapter 10

Functional Design

This section will describe the functions needed in the system, to fulfil the requirements. Included are the functionalities offered to the user in section 10.1. Based on these, some use cases are formulated, which are given from page 61 on. Section 10.3 describes the functionality that is needed within the system, derived from the above and the requirements. The last is a listing of all the actual, although abstract, elements that are needed in the system to fulfil the functionality and requirements.

10.1 Offered functionality

Some things have to be offered to the user, so he or she can use it. These are the functionalities that can be observed and controlled by the user. An inventory of these features follows:

User interface First of all, the user will need a way to interact with the system. This normally works in both directions, information from the system to the user as well as commands from the user for the system. Although many people will first identify a desktop-GUI as an user interface, many other variants exist. For example one can use the remote control of a television, interactive voice response (IVR) systems for telephone services, etc. . .

Messaging Probably the most important element of the system for the user will be the possibility to receive¹ all types of messages. These should be possible in any way the user wants (either as speech, text, image and so on), but also from any source available to the system (fax, e-mail, news, etc. . .).

¹ Sending messages can be included, although “traditional” means can be used as well. This project emphasises on receiving ones messages.

Most critical hereof is that the previous stated user interface supports a way to let the user perceive the message. The media supported by the used device should not imply any restriction. Message received in another medium should appear in an available medium, but certain formats may have preference above others, based on the actual message.

Ubiquitous From the user's point of view, the system should be accessible in any place. This could be solved by an approximation of ubiquitous computing, by supporting the usage of mobile devices. Access from available non-mobile devices should be included, when the device can be connected to the network.

Persistency Messages that have been received by the user, should optionally be archived. This means they can, even when the entire system has been shut down in the meanwhile, be requested at a later time. Three implications are derived hereof:

- The user needs a way to retrieve the messages. Stored messages must be retrievable, to allow the user access to the message.
- The user must be able to view and manipulate the archive. This includes receiving a list of the messages present, deleting messages and others like moving.
- The user should be able to organise or structure the archive, i.e. by the usage of a folder-concept. Great resemblance can be found in most e-mail applications, as well as in physical archives. Logically, this increases the number of available operations as given under the previous item.

Message prioritising Although this will not necessarily be an explicit feature for the user, it is an important one though. New messages for the user are only send directly to the user whenever it confirms to the current interests² of the user. Ideally, the user will not have to configure or state its interests, but the system will derive these automatically. This might force the user to (explicitly) provide feedback, in order for the system to perform better. Since the system has to consider the user's *current* interests, the user's situation must be known.

²Although this might actually be the system's representation thereof.

Personal For the user the system is personal, and should be adaptable for the user³. Not only does this apply to the messages received, but also the system's behaviour. The prioritising mentioned before should happen on a per user basis. In a more advanced setting, the user can replace each separate part of the system to fulfil its specific needs even better.

10.2 Use cases

Below are the most important scenarios the system has to fulfil. These form the most basic behaviour the architecture should be able to handle. Scenarios not mentioned here are no target of the system, but many other scenarios can possibly be handled or integrated. Most of these come from regular practise in usage of e-mail and related system. Although these use cases are still quite abstract, filling them in with applicable devices, messages and situations should not be too hard.

Incoming important message A new message arrives, which is important to the user in its current situation. The message should therefore be forwarded to the user's device immediately. A notification or presentation of the newly arrived message should be done in the best (from the user's point of view) available format. A notification will only alert the user of the arrival of the message, while a presentation will let the user perceive⁴ the message.

Currently unimportant message Another new message arrives, which will not require the user's attention at the moment. It should be hold back until a more suitable moment comes forth.

User situation change The user's current situation changes, and (s)he updates this at the system⁵. All messages that were not relevant in the previous situation, but are relevant for this new situation, have to be shown now. The user's situation comprises two elements:

- The user's current interests.
- The user's device and user interface.

³What is not meant here is the customisation of colours and sounds.

⁴Perceiving a message depends on the effective medium *on the device*: a text is read, an image is viewed, sound is heard, etc.

⁵Although this implies a deliberate action, this might be automated (see for instance [60]).

The latter may include an entirely new device. In this case it should be incorporated into the rest of the system automatically. Both of them have to be considered, since they can influence each other. This for two reasons:

1. When the message cannot be viewed at all (e.g. video to SMS), one might not want to see the message at all, while others do want a notification so they can switch devices to receive the message. This may apply to the network connection as well. When using a (expensive) mobile connection, one may wish to receive a short notification, while on a dedicated connection, one wishes to receive the entire message.
2. One may wish to view the same message in different ways on the same device in different situations. For example one wishes to receive a message as text (SMS) during a meeting, but as audio while driving, both using the same mobile phone.

User feedback on message An user has received and read a message, and provides feedback⁶ on the decision made for this message. The system should learn from such feedback, to improve classification accuracy for future messages. An (re-)evaluation of already arrived, but yet unread messages, may (optionally) be triggered hereby. Feedback can be provided on the decision regarding importance of the message and the format it is presented in.

User stores message After an user receives a message, (s)he may want to store the message. In this way, messages are filed for later access.

User requests (stored) message The user should have the possibility to access a stored message. Other manipulations on archived messages should be possible as well. Note that to make a request, a list of stored messages is needed first. This list can be a request itself, for a list of available messages.

10.3 Internal functions

Derived from the previous stated requirements (chapter 9), and the previous sections (10.1 and 10.2), a list of needed functions internally in the system is compiled. This list will briefly highlight those elements that are needed in the system, in order to achieve the above.

⁶Either explicitly, or measured by the system.

User interface For an usable system, several elements have to be supported. These are listed below, and are either *Essential*, *Important* or *Optional* (which are derived from the requirements). Although they are all to be supported, certain functions are hard to realise for certain devices.

E Alert for a newly arrived message.

E Perceiving a message.

E Setting the current situation⁷.

I Providing feedback⁷.

I Storing a message.

O Retrieving stored messages⁸.

O Listing stored messages.

O Manipulating the storage.

Input device handling Since messages are to appear from (physical) devices, these devices need to communicate with the system. As important is the fact the system needs to be able to receive and package the messages for the system.

Automated transformation In order to receive messages in the preferred format rather than the send format, automated transformations between different media formats are needed in the system.

Discovery and locating Using and adding any device transparently to the system requires a mechanism to automatically advertise, find and identify devices.

Persistency Storing messages for later access requires some form of background storage. This background storage needs the capabilities to store, list, remove and retrieve messages. Creating and controlling further structures within this storage is preferred.

⁷The user interface might automate this.

⁸Storing needs to be handled directly after perceiving the message on a device, but retrieving can be handled on other devices as well.

Priority decision Messages have to be judged for their relevance to the user's current situation. Maintaining this state is a necessary feature therefore.

Format decision Besides establishing the priority of a message, the format to use for the user to perceive the message has to be decided. This decision depends on the user's situation as well (as discussed in 10.2).

Situation tracking The situation of the user needs to be known, as stated at the previous two items. A way to keep this state of the user up-to-date is therefore required.

Feedback When the system is required to improve its accuracy of judgement, the system needs feedback from the user.

Personalisation When the system is multi-user, a way to differentiate between the users is needed. This applies to the judgement of a message in respect to the user's interest, as well as the locating mechanism described above and other personal features.

Recovery To fulfil the requirement to be robust, the system needs to handle unpredicted behaviour. Additional scenarios to handle these exceptional cases need to be embedded.

Media independence Handling different media in the same way needs extra care. Since creating a single unified format is considered unachievable, an extra level of abstraction is needed. This layer of meta-data will describe the actual content in a more unified manner.

Flexibility As a last item, adaptability is an optional feature. This requires flexibility in the design, in order to allow variations in performed operations.

Chapter 11

Selection

This is an evaluation of the choices made based on literature and reason. It should only contain high-level choices, with impact on architecture, which is described on page 79 and further. Items specific to implementation should be avoided, and left open till the start of the prototype.

One of the main thoughts used is the reduction of central components where possible. This in order to achieve a highly distributed but reliable system. The main aspect in this design is functionality. Issues directly related to machines and implementations are left out as much as possible.

11.1 Multi-agent

The most important choice made, was the one in favour of a multi-agent system. Below is an explanation why this decision was made in the first place. Next is a brief overview of the functions that have to be divided over these agents.

11.1.1 Why?

Although many agent-based applications that function as “software secretaries” are composed of a single agent (from [83] with regard to [39]), chosen was to use a multi-agent basis. The following advantages of multi-agent over a single agent can be identified:

Size Although the total size of a multi-agent system is likely to be larger, a single agent can become too large for small devices.

Modification Each modification in a single agent system requires the entire

system to be replaced¹. In a multi-agent system, a certain agent can be replaced transparently by another agent with altered functionalities.

Connectivity A single agent system is normally restricted to one machine at a time, limiting the system in its capacity to simultaneously span multiple devices, providers, sources and networks.

Data traffic A single agent also requires all messages to be transferred to the user's device, causing large amounts of data traffic to end-devices one wishes to avoid, or a continuous connection between the user's device and the agent.

Modular Using separate agents for distinct functionality creates a modular system. Although this is common practise in software engineering, using agents allows for more autonomy, providing more flexibility [83].

Personalisation A single agent requires the agent to be generic, so it is harder to personalise. Otherwise it is personal and therefore will omit public improvements made to a particular part of the system without updating the user's system. With multiple agents, partial personalised behaviour can be achieved more easily.

For the communication between these agents, asynchronous messaging is the best candidate. A blackboard system will introduce a central component, implying a single point of failure in a networked system. As will become clear, there is no need for data shared among all agents. Besides, a blackboard architecture implies that all agents need to check the blackboard for messages. This requires continuously active requests from the user's device, something wished to be avoided in the first place. Asynchronous messaging triggers another agent to react, effectively causing *push*-technology.

11.1.2 Compartment

Since the previous choice was to have a multi-agent system, the next choice is how to divide these agents. This separation is based on the needed functionalities as given in section 10.3. Following these functionalities, the next functions are identified:

- Input.
- User interface.

¹This may ignore the fact that dynamic loading of code is possible, but this is considered to have less capabilities.

- Storage.
- Transformation.
- Routing.
- Extraction.
- Priority profiling.
- Format profiling.

The motivations for this particular partitioning are given next. Most of these are based on a survey of relevant literature [61]. Several iterations were made over the design choices and related literature. Due to these iterations, most choices cannot be pinpointed to specific sources. The resulting reasoning is therefore included as motivation.

Furthermore, details of other specific characteristics are discussed. The first four items are grouped together and will be described in 11.2. Routing is explained on page 71 and further. The last three are handled in section 11.4. Reasons to have this grouping will become clear in the respective sections.

Please note that the actual agents are defined in chapter 12. A detailed description of each agent can be found there. On page 80 you can find an overview of these agents (Figure 12.1).

11.2 Peripherals

The first group of agents to be discussed is quite obvious. These are easily derived from most common e-mail applications, along with the first items² on the list of included functions (see page 62). Covered by these four elements are almost all the functionalities needed for an unified messaging system. They are split up according the presented functionalities. Aside from the transformation, these simply cover the peripherals found in a most elementary (e-mail) system. Note that user interaction and output are combined.

11.2.1 Motivation

The separation of the control of peripheral devices is not without reason. Each of the parts is motivated:

²Actually the first five items without the fourth (the item on discovery and locating).

Input Due to the fact that different input devices must be added easily to the system, they are separated as well. Note that this input only refers to new messages that are to be used in the system. The user's input will be handled by the user interface.

Output and User Interface On the other side is the output. The choice was made to integrate the functionality of the user interface with this output. Motive to do so is the general necessity for the output to have an user interface itself. One could think of scrolling functionalities in case of text, zooming for images or forward and rewind for audio. Since these need to be supported anyway, some additional features should be possible. An additional advantage is the fact that specialised libraries for hardware can be shared when needed³.

Storage Reasons to have a separate storage are trivial. Since storage has no direct relation with receiving, viewing or transforming a message, there is no need to integrate this functionality together. When an entirely different method of storage (in the implementation) is changed, this should have no impact on the other elements. Therefore, the storage is separated.

Transformation A separate step of transformation is used. This is done in order to avoid an universal media-format. One format is considered impossible, without huge loss of information⁴ and an unneeded overhead of conversions.

Aiming for transparently cross-connecting "any" device, all inputs and user interfaces will need to be connected⁵. When each of these device must be able to process messages for or from the other, this leads to a full-connected network. Drawback hereof follows from graph-theory. Full connected networks require each node to have a connection with all the others. When connecting all inputs with all user interfaces, this means that each new input device will need to support all output devices. Even worse is probably the fact that adding a new output device (user interface) could require a modification to all inputs. Since this strokes heavily with the requirement of easy and dynamically extending the system, another solution is needed.

This leads to the following solution: Instead of having each device connecting to all others, use a single point of connection. This single connector will then need to take care of all connections, but this can be handled in

³Although this might be valid for input as well, the reasons to separate are stronger.

⁴How can one express a video in simple text?

⁵Connected with regard to information exchange, rather than data.

another way. How this can be solved is shown later, when the transformer is described in more detail (page 70).

11.2.2 Input

Input devices will be encapsulated by an agent that creates input for the system. Such an agent needs to be connected with the device that forms the source of the message for the system. A logical choice for these is on a computer directly connected with the device. A device not always directly resembles a physical device in this context however. E-mail for the user for instance, can arrive on a remote system as well. In this case, the agent will have to connect over the network to fetch this e-mail like any other user-application can do. The term input device is thus used as an abstraction for the source of a message, rather than the underlying piece of equipment.

The encapsulation of the input device by an agent is useful to shield the system from specific and proprietary handling. It saves other parts from handling or accessing any specific characteristics of the device (such as the format *on the wire*). On the other side it does not create messages in a universal format. It merely packages the messages in its specific format in a way the system can handle it. What can be done is some minor preparation of the format. Formats that apply only to very specific devices can be changed to more common standards. An image from a fax can be converted to the more generic TIFF format for instance. Another example is re-sampling audio-messages, to use a frequency applied in common practise. The agent that handles input is described in 12.3.1, and examples are shown on the far left of Figure 12.1.

11.2.3 User Interface

Similar to the input, the user interface will wrap an (interactive) output device. As with the input devices, the device is an abstraction. The output hides the implementation details from the rest of the system. Towards the system, it represents a generic output, accepting standardised packages. The contents of these packages has to be in a format that can be handled by the output device. The user interface should thus advertise these capabilities.

The output and interaction can be handled in many ways. This should all be handled by the agent handling this output. If multiple physical devices are needed, the agent should handle these, e.g. a phone combined with a fax, using the phone as a control channel, while messages can be printed using the

fax⁶. The user interaction has to account for any state the device requires as well. When a message requires specific handling, the agent should do this. For example when using SMS, the agent could send longer message in several pieces. Agents that fulfil this are described in 12.3.6, and can be found on the right side of Figure 12.1.

11.2.4 Storage

A storage agent is provided as transparent storage for messages. Although storage can be pretty straightforward, a separation is provided to split the implementation from other parts of the system. This allows to replace the storage with a completely different type of persistent background storage. The storage needs to implement the capabilities listed earlier in section 10.1 under “persistence”.

11.2.5 Transformation

As was stated before, having many different kinds of devices connecting to each other is not realistic. It was motivated before to have a single point of connection. On first sight this might look like moving the problem. When all connectivity is confined in one place, alternative ways of reaching a transformation are possible. Instead of having all possible combinations of transformation (full connected), one can build chains of transformations. This can decrease the number of required connections dramatically. Although one could argue this can be used for the separate inputs and outputs as well, this still involves all these nodes. Removing one node may have severe impact on the entire system.

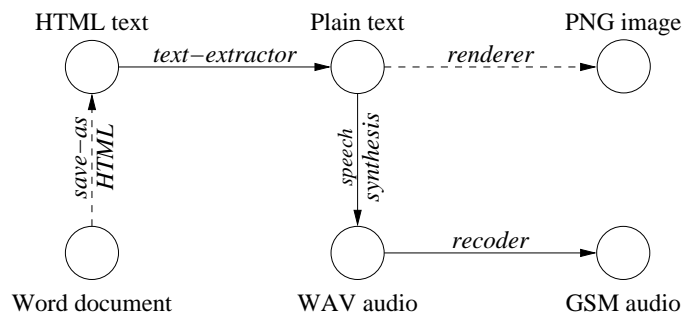


Figure 11.1: Chains of transformations

⁶Most modern faxes do have phone-capabilities embedded.

The idea behind the chains of transformations is quite simple. Instead of converting one format to another, one uses several steps to achieve the transformation. This idea is very similar to the “automatic path creation” used in the *universal inbox* [56]. An illustration is given in Figure 11.1. This example shows how an established transformation chain from HTML-text to GSM-audio can be extended to accept Word-documents. As alternative, it can be adapted to create PNG-images as well. A transformation in this context is a conversion from one media or application format to another, not a transportation bridge (like FTP to HTTP).

Advantage of this approach is the reduction of required conversions, together with the easy addition of new devices. When this new device produces (input) or accepts (user interface) a known format, this can be automatically supported. A disadvantage occurs when a new device is added that does not work with a known format. A modification to the transformation is then required as well. The quality of the transformations has to be closely guarded, to assure no information is lost unnoticed during a conversion.

11.3 Routing

Although all elements from the preceding group can be used to create an unified messaging system, no explanation of them connecting together is given yet. For this connection, a routing facility is used. This routing is a separated functionality that handles transportation and controls the flow of a message through the system.

The main advantage of using a separate router, is that only one (type of) node has to have knowledge of routing. The flow of a message through the different phases of the system and parts of the network can therefore be adapted by only updating the routers. Other nodes should be able to continue without noticing any difference. Effectively, the router is the only node that does the entire job, it only transfers all of its work to several other nodes.

A central router may pose limits to the capabilities to span a network, connecting many devices. Furthermore it requires passing large amounts of data through the same network connection multiple times. Therefore, multiple routers are chosen. These routers are (in principle) mobile, thus carry the message through the network, although they can alternatively be (partial) static and hand over messages as if they were moving. Conceptually, one message is handled by one router. In this way a router handles the different states of a message very naturally.

Using multiple of these separated routers has advantages:

Connectivity Connections between different networks can be supported, even through other network-systems (when appropriate interfaces are available).

Diversification Possibility for different routing capabilities per message (personalised / message-dependant routing), or different per location (when using routers that are static and pass messages).

Minimise network-traffic Avoid double traffic of a central router by using routers that are distributed in a network. A central agent would have to use remote communication several times, before the results can be passed to the next step. A mobile router only has to make one move, communicate locally, and move to the next phase. Consider Figure 11.2, where each arrow represents communication (or a move) over the network, and each circle a node in the process.

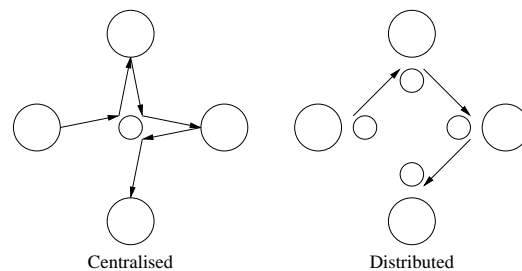


Figure 11.2: Centralised versus distributed routing over network

As a drawback, new steps in the processing cannot be introduced without adjusting the router agents. One does not only have to write the new functionality for a new step, but adjust the routers to use it as well. Such a modification is necessary for most (distributed) solutions however.

Except for this router, each node only needs limited knowledge of messages. Therefore, additional functions can be added, without interfering with existing elements of the system. Note that routers do not form just a transportation “backbone”, but are functional routers. They decide which action to take next, in the process of steps a message has to go through.

Why move the data to the calculation, opposite to regular practise in mobile agent technology of moving the computation to the data:

- First of all, several parts of the code of the intended are expected to be larger than many of the pieces of data, while the use of many and widespread sources requires large amounts of movement. Several types

of messages are expected to be rather short (Short (SMS) and instant messages (ICQ / AIM / MSM), a share of e-mail, voice-mail, news-ticker service). A storage with all previous data cannot seriously be considered to be (fully) mobile.

- The data has to be moved anyway (from the source to the user).
- Next, the profile⁷ requires continuous reachability and adaptability (learning from feedback), if (ideally) updated instantly after feedback. When moving a single profile around, a delay in communication or movement will hold up all new messages. A distributed approach, using a profile agent per message, will require synchronisation and a distributed user-profile, or will suffer from bad (profiling) performance when updated less often. This can apply to the extractor⁷ as well when it needs data of previous and send messages.
- A message has to go through several operations. Moving the message around to these operations is still some sort of code along with the data. Only the operations are delegated to other nodes. Transitions between several of these operations, the different phases, are transitions between different states as well. While data does not naturally keep a state of its own, agents do, therefore wrapping a message in an agent *upgrades* the message to a state-keeping one.
- As a last added benefit is the improved flexibility. When each message is enhanced with its own code, the processing of each message can be varied. Each message can theoretically⁸ have be processed in another way. This variation can be based on the origin of the message, as well as the addressee, type of message or other criteria.

11.4 Profiling

All the elements discussed in the previous sections only relate to the ability to receive ones messages in a unified way. None of these parts considers the interest the user might have in the particular message. It is an important part of the project however, to save the user from a communication overload. For this reason three additional elements were identified in 11.1:

- Extraction.

⁷Profile and extractor are both introduced in the next section.

⁸Performance reasons will likely restrict the degree of freedom to a set of more generic possibilities.

- Priority profiling.
- Format profiling.

Since we are dealing with messages of any kind of medium, many solutions previously used cannot easily be applied. Many approaches have been proposed to deal with the e-mail overload. Most of these are based on the fact that most e-mail consist of plain text. In an environment where many other formats can be encountered, this will be rendered rather useless. As a solution a decision can be reached in two steps, first a phase where meta-data will be extracted and afterwards a decision based on this meta-data.

Two steps Separating a phase of extraction before the actual decision is based on previous research in e-mail filtering, where extraction is a step made before profiling (e.g. [4]). One could compare with data analysis found in information filtering as well [23]. The first step is extracting meta-data, descriptive information, from the actual message. Meta-data can be seen as information about other data, describing important characteristics, i.e. the author, the size, the topic, urgency, etc... The second step is the effective decision made based on these meta-data. This separation is made for several reasons that are beneficial:

- Profile is independent of the used formats, adding new formats without modification. The profiler can manage to judge new formats without an addition to the user profile. A first occurrence of a format can then be handled based on the extracted meta-data, compared to similar meta-data of messages in other formats.
- Additional extractions can offer possible improvements without adaption of the profiler. When new technology provides new or better ways to evaluate certain media formats, they can be added without revising the profiler.
- Cross-media profiling is possible. Using uniform ways to express meta-data, independent of the type of media used, similar decisions for different media can be made. This allows for knowledge gained with a certain media-type to be applied transparently to another media-type.

Splitted decision The second decision was to split the profiling based on this extracted meta-data. First of all, a decision whether the message is important needs to be made. When a message needs to be shown to the user, the format to present it has to be decided as well. These are two separate

steps, since this not only applies to messages that arrive as new ones, but also for the messages that are requested by the user from the storage. These last need to be adjusted to the user's current device as well. Therefore, both a step to apply a priority profile and one for the format profiling are needed.

Priority: piles The main task of the priority profile is to assign applicable situations to each message. The user defines those situations itself, allowing the user to have as much variation as needed. A simple example would be a request from a customer, which will be shown only at work, when working at home or travelling to work, but not when at home, travelling home or during a meeting.

Multiple situations can be assigned to each message. One can regard this as putting documents on *piles*, where each pile represents a situation. Note that due to the use of virtual documents (electronic messages), removing a message from one pile implies removing the message from another pile as well. This concept can be realised simply by maintaining references in these piles, instead of copies of the entire document. The use of electronic documents ensures fast and easy access to the actual documents. Assigning a situation to messages can thus be seen as putting a message on a virtual pile.

11.5 Miscellaneous

Now that all elements that are needed have been identified, only a few minor items are left. A few other things must be exchanged between the different agents. This is required for storage manipulation and the user needs to be able to set a situation. In a similar way (s)he needs to be able to provide feedback for the learning profile.

Control as messages Except for the messages themselves, there are a few more internal required communications. Requests and notifications are special instances of a message, in order to utilise the router and simplify the system.

- Requests (to retrieve a stored message) are special messages. They start at a different phase, but are largely equal.
- Requesting lists or multiple messages can be done with special requests, requests marked with slightly different content (requests with wildcard id or selection and a headers-only field for example). The same can apply for other storage operations, like delete or operations on folders.

- A notification is an internal message, send from the user interface to the profile. Notifications form the carrier of feedback and situation changes. A notification will thus require special treatment from the router.

Feedback A necessity for the system to learn the user's interest, is knowledge thereof through feedback. Feedback can be either explicit or implicit. Explicit feedback is active participation of the user, while an implicit method measures user responses to establish feedback. Chosen is to use explicit feedback as primary choice in this design for the following reasons:

More reliable When measuring the user's responses, a certain amount of uncertainty is introduced. One cannot measure the reason of (e.g.) the reaction time of a user.

More expressive Except for providing positive or negative feedback, one can add additional possibilities to provide a reason or other adjustments. In this way more detailed feedback can be gathered, allowing for more directed learning capabilities.

Easier to realise Adding a few buttons (or other "commands", depending the type of user interface), is easier than building and tuning a measurement system.

This does require explicit user action. Although this places a slight burden on the user, this is regarded a lighter one than the communication overload.

Since explicit feedback allows more expressive feedback, a brief discussion of this feedback is appropriate. Although the exact feedback depends on the possible situations, the capabilities of the profile and even the user interface, one should think along the following lines: Feedback for learning algorithms normally exists of positive or negative feedback. In this way a decision could only have been either good or bad. However, at least two decisions are made. Next, a decision could not only have been good or bad, with explicit feedback an alternate decision can be provided. In this way, one does not only provide a negative example for the decision that was made, but a positive example as well for the decision that should have been made. The result looks like the following examples:

- Thanks (positive for situation and format).
- Wait till *home* (negative given situation, positive other situation).

- Rather as *SMS* (positive given situation, negative given format, positive other format).
- Do not bother me (negative situation).

Chapter 12

Architecture

After having all the required functions identified, these still need to be combined together to form the system. In this section a description of the architecture as designed will be presented. The form in which the architecture is presented is based on the Tryllian method *SmartAgent* [72].

Methodology As identified by Tveit [76], several methodologies exist for designing agent-based systems. The Gaia [85] methodology emphasises the relation between agents, defining their roles, responsibilities and permissions. It further incorporates assigning instances of agents, aiming to be directly implementable. This method is not used, as it is considered too broad, but the methodology matches best with the Tryllian SmartAgent methodology of those discussed by Tveit. Other methodologies discussed by Wooldridge and Ciancarini [83] do not seem applicable either. Those methodologies are either similar, or have other target areas like belief-desire-intention, which are suited for single agent systems and not the case at hand.

Description The used methodology works *top-down*. First all roles that have to be fulfilled are identified. Next, the total system is described, defining the different agents and their relations to each other. The communication between agents themselves is given as well. Later in this chapter follow other, non-agent, components encountered in the architecture. Note that further details are included in the method, but these are given as part of the description of the prototype. One can find these parts in chapter 14 starting on page 121.

But before all definitions and other formal representations, an overview of the system will be given. This will give the reader a first impression of the architecture, and can be used as a guideline throughout the remainder of this chapter.

12.1 Overview

First a brief description of what a message¹ is within the system. A message within the system actually exists of two parts. The first part is, naturally, the message itself. The other part is the information that guides the message through the system, the meta-data. This is data that describes the message. It can include the elements as size, sender and addressee² which normally exist in messaging systems as well. Furthermore it contains additional information generated within the system. These are the meta-data extracted by the system from the contents.

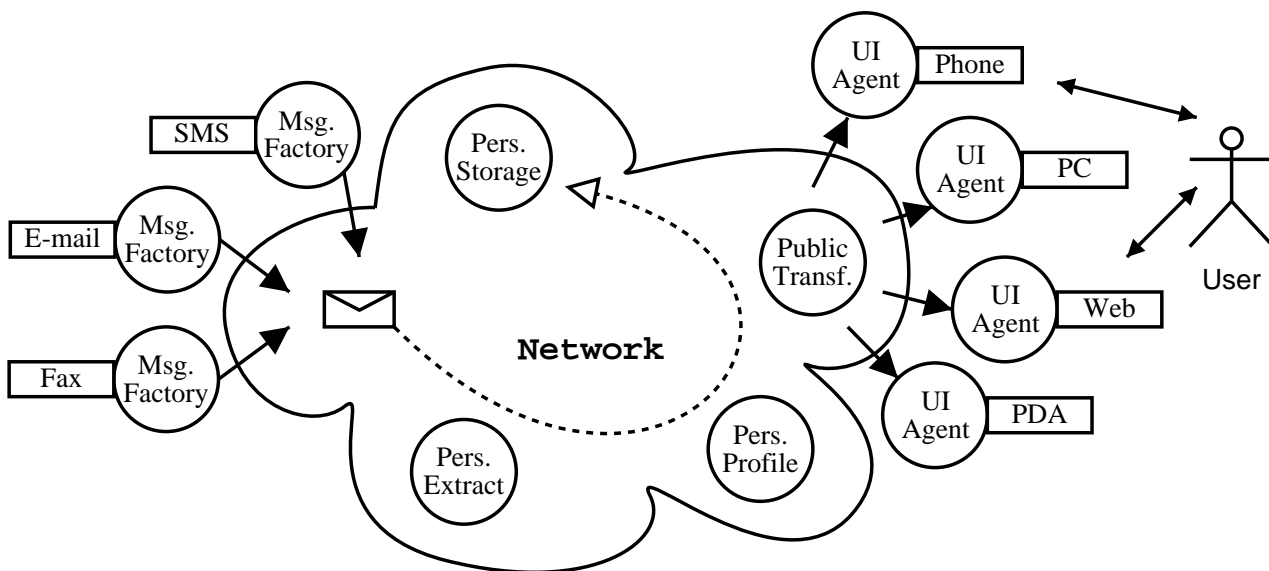


Figure 12.1: Overview of the architecture

One can compare a message with a letter, similar to the metaphor often used for e-mail [11]: The message is the letter, contained in an envelope. On the envelope is the address and (sometimes) the sender. Now this applies to the messages used within this system as well, where the letter is certain data (be it an e-mail, or a recorded voice-mail, etc. . .). This data is packed within the envelope, which has the meta-data *written* on it. Furthermore,

¹This is the “unified” message send from user to user, not messages as used internal in the system, those used between different agents.

²This is not always equal, compare your home and work e-mail-address, phone-number or mailing-lists you are subscribed to.

the system can make additional notes, like you would write or stick a *post-it* on the envelope, which contain the meta-data generated in the system.

The Message Factory Agent packs the data in the envelope, addresses it and start a Router Agent with the package. An Extractor will take out the data, and attach extra notes to the envelope, which further describe the message (urgency, user's interest topics, user's relation to sender, etc.). Next, a Profile Agent will decide whether / when this message is important, based on the information found *on* the envelope. The same Profile Agent will attach an extra note with the (preferred) format in which the message is to be presented. A Transformer Agent will examine the contents of the envelope, and convert it into the desired format. Finally, a User Interface Agent will show this to the user. As a bonus, there is a Storage Agent to archive messages.

12.2 Roles

Before we can establish all required agents, their functions need to be known. For this purpose, roles are defined. A role is a function an agent can fulfil, similar to roles humans can have in society. Agents and roles are not one-on-one connected to each other. An agent can have multiple roles, and a role can be fulfilled by multiple agents. In the metaphor of a human-society one can be a customer at the bank, but the banker is a customer at the grocery himself as well.

In the system the roles are easily derived from the functions identified in [11.1](#). Each role that needs to be present is thus only briefly described below:

Input When an agent acts as input, it receives messages for the system. It packages a new message for the system, and initiates the processing of the message.

User Interface An agent that performs the function of user interface, will handle all of the interaction with the user. This includes letting the user perceive the message, thus fulfilling the function of output.

Storage A storage handles messages that need to be archived. This includes storing, retrieving and organisation of the collection of messages.

Transformer The role of transformer involves the conversion from the original medium-type of receipt, to the preferred format for acceptance by the user.

Router Message have to be transported, and the sequence of operations needs to be determined. The agent handling this has the role of router. It connects and coordinates the relation of messages with the other agents in the system.

Extractor During the extraction, information present in the message is made explicit. The extractor can be said to *write* additional meta-data on the envelope, as processable information for the rest of the system.

Priority profiler A filtering mechanism decides the destiny of a message. Messages that are important have to be shown to the user instantly, other messages are hold off until they do become important.

Format profiler A message that is to be given to the user, needs to be usable on the user's current device, in the best available format. A format profiler does so.

12.3 Agents

Now that the roles in the system are known, the agents that fulfil them can be determined. In this section all classes of agents that appear in the system and their view upon their environment are described. These are explicitly *classes*, since the actual instances that are needed in an implementation may vary per user, providers and available equipment.

First a small summary of the intended classes of agents and the mapping of the roles on these. After this, each agent will be described in more detail. The agents are listed in the order of the intended process, not that of the previously given list of roles. This process will be discussed in further detail in the rest of this chapter and the following part.

The last five types of agents have another common denominator. Each of these agents provide a certain kind of service. Although this is not entirely clear right now, it will become later on. For now it is mentioned they are also referred to as Service Agents. Any of the five types is mentioned then.

Message Factory Agent A Message Factory Agent handles the input. It is named factory since it creates a Router Agent with the packaged message.

Router Agent The Router Agent handles the routing of messages. It is directly mapped to a single message.

Extractor Agent An Extractor Agent adds meta-data to the message.

Profiling Agent A Profile Agent is the only agent which fulfil two roles. It decides both the applicable priority and the preferred format for the message.

Transformer Agent An Transformer Agent ensures the message is transformed to the preferred format.

User Interface Agent The User Interface Agent, or UI Agent for short, handles the interaction with the user.

Storage Agent The Storage Agent acts in the role of storage, which hardly needs further explanation anymore.

12.3.1 Message Factory Agent

The Factory is the generator of messages for the system. It annotates the message with protocol specific meta-data before delivering the message to the system. These protocol specific (primary) meta-data are e.g. time of arrival, the sender's address (e-mail, ICQ) or a telephone number (fax, voice-mail) etc. In metaphor this agent can be seen as the one who fills and posts the envelope into the system. First some characteristics of the agent, then a few examples.

Properties The properties of a common Message Factory are listed in Table 12.1. Since this is the first such listing, the non-obvious items might need a little explanation. A component is a non-agent element of the system, that is utilised by the agent. The partners are the other agents the particular agent communicates with.

When an agent is located "anywhere", this indicates it is located somewhere in the network, preferably on a central server. With accessibility, the restrictions for a multi-user system are meant; agents are either personal, or generic (shared among multiple users). Agents are either of a static or a mobile type.

Examples A facsimile device can be used as an input device for messages. Whenever a fax arrives, the agent will buffer the image. When the entire fax has arrived, the fax is packaged as a system-message. The Fax Factory Agent can use CLID to add the sender's original phone-number as meta-data.

Another example would be an Usenet Factory Agent. This type of agent will monitor news- and discussion-groups as found in Usenet, and feed the

Table 12.1: Properties of a Factory Agent

Roles	Input
Data	Creates the message-content and initial meta-data
Components	Message receiving devices
Partners	Initiates a Router Agent
Accessibility	Personal
Location	At input device
Type	Static
Instances	Multiple per user One per input device
Remarks	Can be specific for a certain input medium Point of contact with external environment

interesting groups³ as messages to the system. The meta-data that can be added are the name of the group and the address of the poster.

12.3.2 Router Agent

Takes care of the routing of a message through all processing in the system. It manages the status of a message, and acts accordingly, delivering it at its next destination. When the other agent is not local, the Router Agent tries to locate the needed agent, and moves to the remote location. A Router Agent is (conceptually) directly connected with a message, forming its “conscious”, or the courier of the envelope. One could say an *autonomous message* is created.

Properties The Router Agent has the characteristics found in its table of properties (12.2).

Process A Router Agent is the one that takes a message through the system. It manages the sequence of operations that have to be applied to a message. This process will be described here. Note that this is the standard process to be applied to an *important* message, any deviation hereof will be discussed later, specifically in 12.6 for the whole system and in more detail per agent in chapter 14. The steps taken below will be commented on afterwards:

³Or even single messages with *preliminary* filtering.

Table 12.2: Properties of the Router Agent

Roles	Router
Data	Transports the content and the meta-data
Components	None
Partners	Initiated by a Factory Agent, contacts all other agents
Accessibility	Personal
Location	Anywhere, depends on step in the process
Type	Mobile
Instances	Multiple per user One per message
Remarks	A message packaged with some active code

1. Factory starts Router.
2. Router consults Extractor for meta-data.
3. Router asks Profile for priority.
4. Router requests Profile for preferred format.
5. Message is transformed by Transformer.
6. Router delivers message at User Interface.
7. Router hands message to Storage.

First of all, the message will be packed in the Router Agent by the Factory. This is where the life-cycle of the Router Agent starts as well. Now logically, the message is to be judged whether it is important. Some further information (the meta-data) is needed first as a basis for that decision. Therefore the Router will go to the Extractor first, and then the Profile. Since the message is important, the format to present it in is acquired of the Profile as well. Now the Transformer is requested to change the message to this format. It can now be delivered to the user. The user finally decides it to be stored, so the message is taken to the Storage.

Other activities As was mentioned in 11.5, the Router needs to help in a few other events. When the user changes his or her situation, the user does so at the UI Agent. This data is needed at the Profile Agent however, and the same applies to feedback. Since the Router handles the connection between these agents, it can be used to transport these notifications as well.

The concept of using only one agent which handles connectivity is kept intact as another effect.

12.3.3 Extractor Agent

A Personal Meta-data Extraction Agent handles the extraction of meta-data for a message. This Extractor adds descriptive meta-data to a message, as a basis for the Profile (described later) to assign situations and / or preferred formats. One could say it sticks more notes on the envelope, describing (properties of the) the contents.

Other agents with usefully information, like the user's calender and travel-agent can be integrated. This will be discussed in [13.2.4](#) and [13.2.5](#).

Table 12.3: Properties of the Extractor Agent

Roles	Extractor
Data	Adds meta-data to a message
Components	None
Partners	Router Agent
Accessibility	Personal
Location	Anywhere
Type	Static
Instances	One per user
Remarks	Can perform media-specific operations

Properties An Extractor Agent can be recognised as given in [Table 12.3](#).

Example extractions Examples of meta-data that could be generated include:

- Address unification with a virtual address book (numbers or addresses specific for the originating device to real-life name).
- Urgency determination based on voice stress analysis of spoken audio.
- Topic classification based on TF-IDF (Term Frequency - Inverse Document Frequency) ratio of previous, user-classified messages (as often applied to e-mail [[38](#), [62](#)]).
- Spam rating, in collaboration with other people's spam filters.

- Proximity based marker, that marks the messages of people that are in the same area as the user, e.g. for appointments, based on telephone-area codes, known location from an address-book, combined with a calendar or GPS (see CLUES [42]).
- Thread recogniser, that marks groups of messages that are followups on previous messages, as often used in mailing-lists and discussion-groups. Algorithms with practical accuracy can be found in current production-level implementations for this classification [88].
- Relationship determination, adds a field with the relation a sender has to the user, for instance using an address-book. Results can be for example family, friend, colleague and customer.
- Due date, if the relevance of the message will expire after an amount of time.
- Language determination, to established the used language in the message. This can be accomplished using e.g. elements of speech recognition for voice-mail or linguistic analysis for texts.

12.3.4 Profile Agent

The Personal Profile Agent assigns applicable situations to each message. It is the part of the system that should prevent the user from being overwhelmed with messages. Each message is assigned zero or more situations, in which the user would be interested in receiving those messages. These could be seen as virtual piles, with each pile containing references to the actual messages. These messages are stored until the user changes its situation, thereby (implicitly) requesting a certain pile. References to messages are then removed from other piles as well. When a message is assigned to a pile that represents the user's current situation, the user can be notified immediately when the priority requires so. In order to know the current situation, the Profile has to maintain a state of the user's situation.

The other task the Profile has is the determination of the format the message is to be presented in. This agent should therefore keep track of both the capacities the UI Agent has, as well as the preferences the user has regarding receiving its messages.

Two roles combined In 11.4 it was argued that both of the described decisions have to be separated. Earlier on page 62 the relation between both assignments was made. Although this might seem contradictory, both

argumentations are still valid. Both decisions are indeed *different* phases in the *process*. They do share some *data*, in order to achieve their decision. The most important part of this data is the user's current situation and feedback. To prevent this information to be known by several agents, it can be shared in a single agent. The agent thus fulfils two roles.

Properties A Profile Agent can be summarised as shown in Table 12.4. It is intended to act fully autonomous for the user. When a Profile Agent does require the user to configure it, an (separate) user interface should be arranged for.

Table 12.4: Properties of the Profile Agent

Roles	Priority profiler Format profiler
Data	Decisions are based on the available meta-data The user's current situation and device
Components	None
Partners	Router Agent
Accessibility	Personal
Location	Anywhere
Type	Static
Instances	One per user
Remarks	Assigns a priority and a desired format to all messages

Profiling characteristics Aside from the properties of the agent itself, an important aspect of the Profile Agent is the profiling itself. The Profiler is intended to have machine learning capabilities for the user's interest. Although this is simply said, it has many implications. A way of learning these interests, an algorithm for creating a profile, is needed. Before an implementation can be created, the characteristics of the learning problem must be known. The most important of these are described here. Further discussion follows in 13.1.

Heterogeneous fields present Different messages can come from different sources, and have a different content. As a result hereof, the available meta-data can differ per message as well. This is due to the message itself, but also depends on the source and the implemented extractors. A source like a fax will likely only provide a device-address and a message-size,

whereas e-mail can provide the sender's priority and relations to previous messages (`reply-to`). The implemented extractors can be medium-specific (per design), and different media can thus have different meta-data as well.

Heterogeneous field types The meta-data that are present with a message, can be heterogeneous among the separate fields as well. A header representing the size of a message is numerical, scalable and sortable, while a related topic of a message (due to an extractor) can only be used as a discrete textual classifier. The relation between different meta-data fields is thus harder to compute. Although this heterogeneity can be used within meta-data with the same semantics (field name), this usage is strongly discouraged for its even higher complexity.

Dynamic Over a certain period of time, things on both ends of the profile might change. The result must represent the user's interests, which can change over time (e.g. [42]). On the input, the messages themselves, new meta-data can become available or might be abandoned. This can be due to newly added or removed extractors, new media used or other improved capabilities of the implementation.

Unforeseeable The system is designed to be highly dynamic and extensible, and to be used over a long period of time. As a result hereof, both the input and the situations of the user cannot be foreseen in advance. While the profile remains valid, the user for instance might modify the used situations. The values used in the field of e.g. a topic classifier (extractor) can alter over time as well, due to changes in the user's occupation. Most of these cannot be established at the time of design, implementation or even deployment.

Noise As common in machine learning problems, noise will be present as well. In this case the user can make mistakes in the feedback. Another important element to consider is inconsistent behaviour of the user. Between different messages, the user may have the same interests, but provide other feedback. The system should thus allow for incorrect classification in the set of known samples.

12.3.5 Transformer Agent

A Data Transformer Agent, or Transformer for short, is generically available to all users. It provides for transformation of data, from one application for-

mat to another. These include transformations between different kind of media, when available. For this reason, advanced features like optical character recognition (OCR), speech-recognition or synthesis will be necessary. Conversions within the same format apply as well, for example down-sampling audio or reducing the resolution of an image. Note that handling of network-specific formats and protocols (e.g. SMTP) are handled by the input and output agents and the corresponding devices. Further improvements to this agent are presented in [13.2.4](#).

Properties The most important property of the Transformer Agent is the fact it is shared by different users. Other aspects are given in [Table 12.5](#).

Table 12.5: Properties of the Transformer Agent

Roles	Transformer
Data	Message content
Components	None
Partners	Router Agent
Accessibility	Generic
Location	Anywhere
Type	Static
Instances	Redundant, each for multiple users
Remarks	Not restricted to messaging, just provides media-conversion

12.3.6 UI Agent

The User Interface Agent is the point of contact with the user. It forms an integral unit with an interface device, allowing an user to be notified of, or access to, its messages. An user has the ability to inform the Profile of a change in its situation, triggering messages for the new situation to be given or adjusted to another device. Feedback to a learning mechanism in the Profile also takes place here. Multiple User Interaction Agents can exist, connected to different Interface Devices at different places to the network. Contrary to the Transformer, an UI Agent might need to transform the transport protocol, to be able to place a message *on the wire*.

Properties Since the user only has access to the system through this agent, it handles many functionalities. It does not implement all these functionali-

ties itself, though. An important detail is the fact that this agent is located at the user's device. This means it is the point of access for the user to the network as well. Since unified messaging has to provide access *anywhere*, this can indicate this agent is hosted on a mobile device. An alternative is hosting the agent as a service handling the last part of the connection, e.g. as a telephone service. Located at the user's device is thus a relative term. Further details can be found in Table 12.6.

Table 12.6: Properties of the UI Agent

Roles	User Interface
Data	Content of the message
Components	User's interactive device, "display" for the user
Partners	Router Agent
Accessibility	Personal
Location	At user's device
Type	Static
Instances	Multiple per user, one active at a time One per interface device
Remarks	Actual delivery of message to user Point of access to system for user

Examples Since the target is to enable "any" device as a message receiving facility, many examples can be given. Only a few will be mentioned here.

The most obvious is probably a desktop application. Similar to any e-mail application, it runs on a personal computer, completely integrated with the agent. The user has a broad variety of available media here, including text, images and audio. Other applications can be used here as well, allowing even more types of media to be received here.

A telephone can be used as well. The agent now will be located on a system where the telephone service resides. An user can call the service with any telephone, to activate the particular device. When a new message arrives, the service rings the user's telephone, and the message can be played audibly to the user. The dials of the phone can be used to give commands, based on an Interactive Voice Response program.

A little unusual will probably be the use of a television. This requires a programmable television with a network connection of course. The agent now runs in the television system, and messages can both be shown or played to the user. The user can use the remote control to interact with the system.

12.3.7 Storage Agent

The Personal Storage Agent takes care of persistence of messages. It stores all messages in order for the user to retrieve them later.

Properties Although the Storage Agent is described to be personal in Table 12.7, the used resources can be shared. Whenever messages are stored or retrieved, the agent must take care to only access messages of the specific user.

Table 12.7: Properties of the Storage Agent

Roles	Storage
Data	Both content and meta-data
Components	Background storage (device or service)
Partners	Router Agent
Accessibility	Personal
Location	Anywhere
Type	Static
Instances	One per user
Remarks	Handles persistency

Examples Examples of background storage can be straightforward. A Storage Agent can be a simple wrapper for an external database. Another option would be to store messages as regular files on a system, and controlling those. Other solutions can be created of course, such as traditional UNIX-mail-spools.

12.4 Communication

In a multi-agent system, agents commonly need to communicate to be able to cooperate. In this section, an overview of the communication needed is given (see Table 12.8, grouped by conversation). The type of a message⁴ is based on FIPA-performatives [15]. To reduce duplicity, only combinations of `request` and `inform` are given. Agents are allowed to reply to a `request` with a `refuse` or `failure` as well. These are other FIPA-performatives,

⁴In this section, these are *not* regular unified messages, but messages between the agents themselves in the system (although these may contain the unified message).

and differ from their corresponding `inform` in purpose and contain a reason instead. Replies that can require longer processing may send a similar `agree` first, to prevent time-outs.

Table 12.8: Agent communication

Message	Type	From	To	Contents
<code>reqMetaData</code>	Request	Router	Extractor	Message content, input-device generated meta-data
<code>infMetaData</code>	Inform	Extractor	Router	Meta-data for the message
<code>reqPile</code>	Request	Router	Profile	Meta-data
<code>infPile</code>	Inform	Profile	Router	Pile(s) and priority for the message
<code>reqFormat</code>	Request	Router	Profile	Meta-data
<code>infFormat</code>	Inform	Profile	Router	Preferred format for the message
<code>reqTransform</code>	Request	Router	Transformer	Format and message content
<code>infTransform</code>	Inform	Transformer	Router	Transformed content
<code>reqPresent</code>	Request	Router	UI	Full message
<code>infPresent</code>	Inform	UI	Router	Storage instruction for message
<code>reqStore</code>	Request	Router	Storage	Full message and storage instructions
<code>infStore</code>	Inform	Storage	Router	Acknowledgement of success
<code>reqRestore</code>	Request	Router	Storage	Message identifier
<code>infRestore</code>	Inform	Storage	Router	Full retrieved message

12.5 Other components

Aside from agents, the system will need other elements to function properly. This section will present a description of these components, most have been described before as part of their related agent.

Input device An input device is a message receiving device. This can be any type of hardware, or an abstract “device” such as a network-service of some kind. Examples of common devices are facsimile machines, voice-mail recorders, e-mail- and news-ticker-services, etc. . . These devices must be accompanied with an appropriate Application Programmers Interface (API) to be usable. They can either provide events for the agent when messages are received (push), or the agent needs to check them at regular intervals for new arrivals (pull).

Interactive output device Interactive devices are very common nowadays. Each reasonable piece of equipment can be used, ranging from a personal computer, phone, fax, PDA and so on. . . Of course these devices can be

accomplished by means of services, like SMS, themselves. An output device needs an API as well.

Background storage Persistency is commonly supported by means of background storage. In the used context here however, it is abstracted as well. A database providing persistency is thus considered a background storage device as well. Others can be used as well, such as files on hard-disks, tapes or even hard-copy on paper, although the latter would be very impractical.

User A component that should most certainly not be neglected, is the user. Normally it is an human entity, who is the reason the system is designed in the first place. He or she is the one that is supposed to use the system. The user in the context of the system is the addressee of a message. The sender will not be addressed with the term user in this document.

12.6 Work- and dataflow

Now that all the required elements of the architecture are in place, their coherency can be explained. This will be done along the use cases defined before (see 10.2 starting at page 61). Each scenario given there will here be examined with a workflow- and a dataflow-diagram. These should explain how the work is divided among the different types of agents, and which agent gets what kind of information. The used diagram methods are explained in appendix B. Notice the last scenario is extra compared to the use cases. This scenario is for the presentation of a message, which is shared among three use cases. To reduce redundancy, this scenario is separated.

12.6.1 Important new message

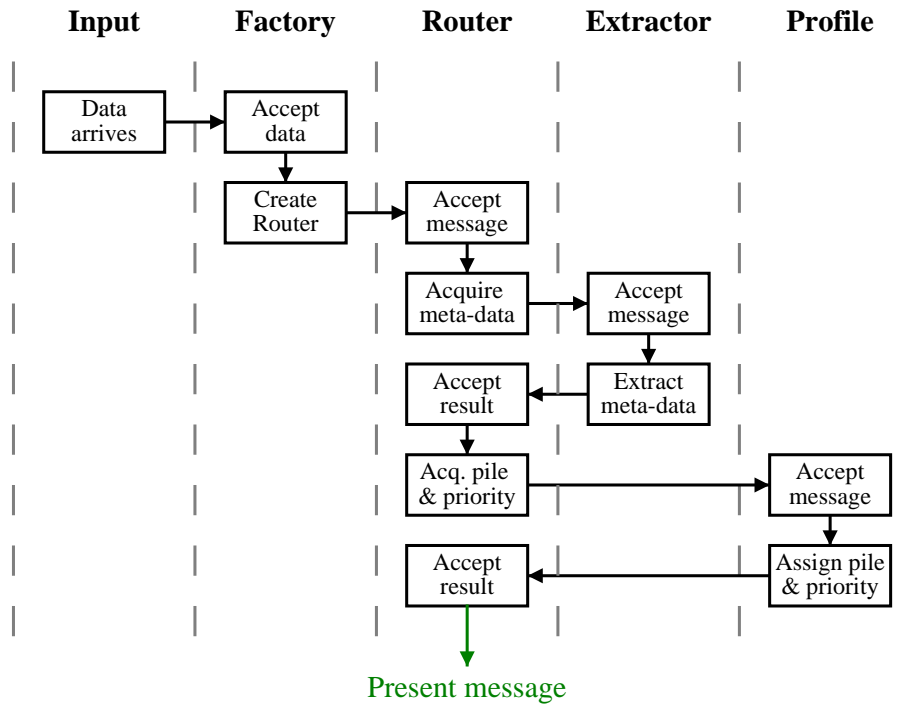


Figure 12.2: Workflow for an important new message

When a new message arrives, the workflow behaves as stated in Figure 12.2 with the dataflow in Figure 12.3. The last part of this workflow is the presentation of the message to the user, as explained later on. A brief description of this workflow:

1. Data arrives at the Message Input Device.
2. The Factory Agent gets the data, and creates a corresponding Router with the message.
3. The Router Agent starts of with the message, and will take the message to the Personal Meta-data Extractor Agent.
4. This instance will annotate the message with meta-data, and return it.
5. Now the Router Agent will take the message to the Personal Profile Agent.

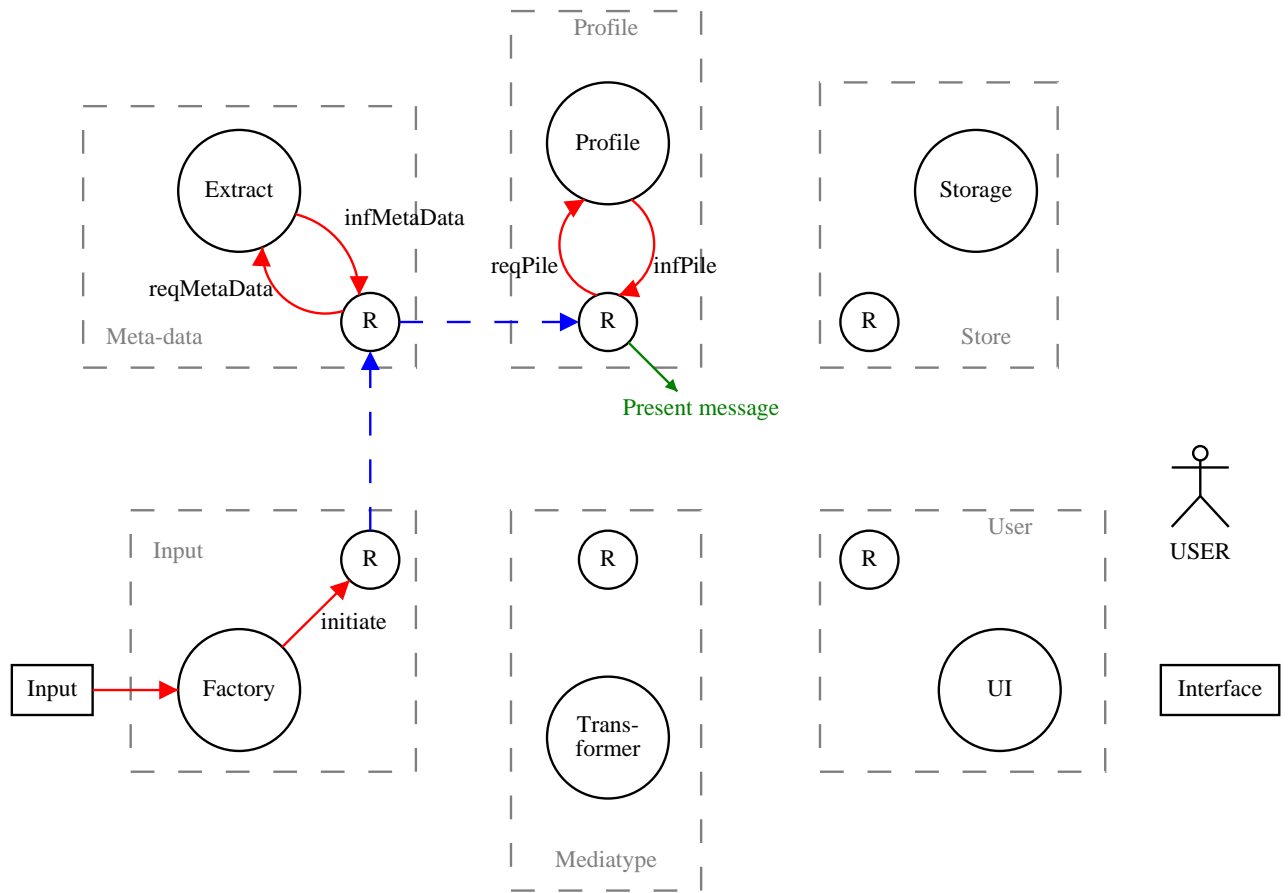


Figure 12.3: Dataflow for an important new message

6. Here, the message will be assigned a priority and the situation(s) it belongs, and is handed back again.
7. The message travels further to be presented to the user.

12.6.2 Unimportant new message

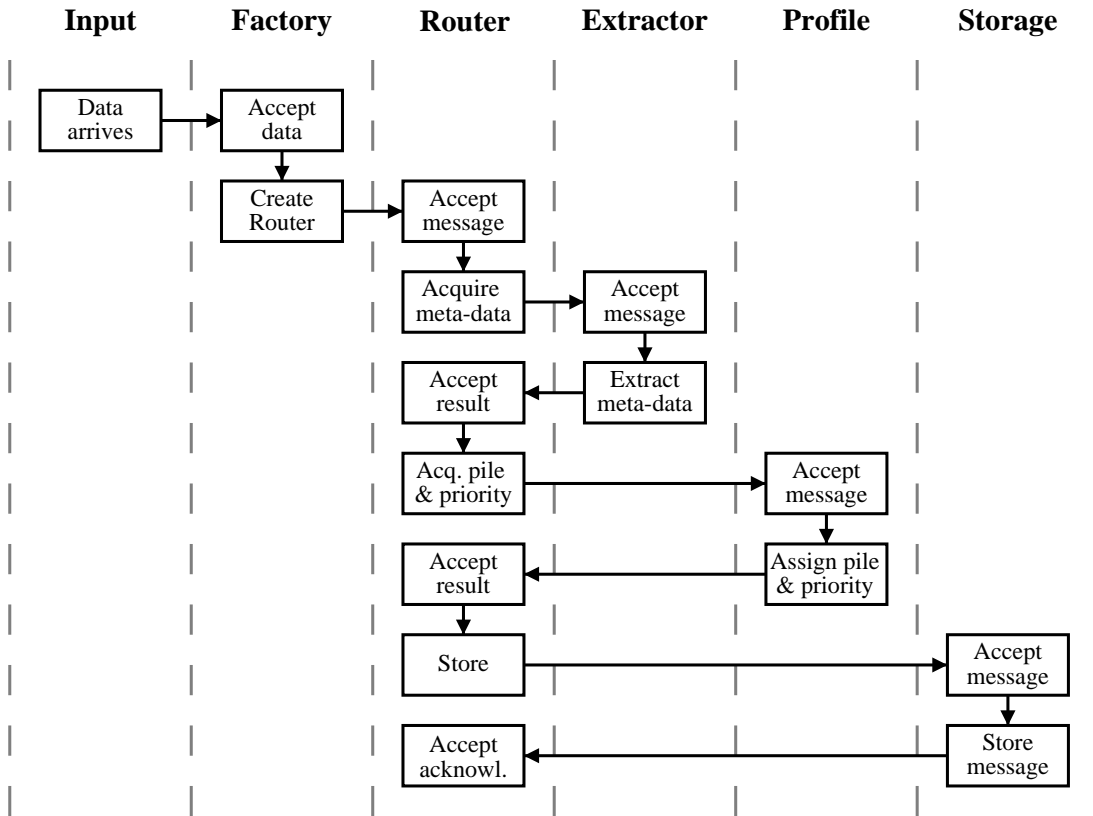


Figure 12.4: Workflow for an unimportant new message

Another scenario is when the incoming message is *not* important to the user at the moment (see figures 12.4 and 12.5). This workflow is almost identical to that of an important message. The only difference is the last part, where the message will not follow the way to be presented to the user, but instead is taken to the Personal Storage Agent.

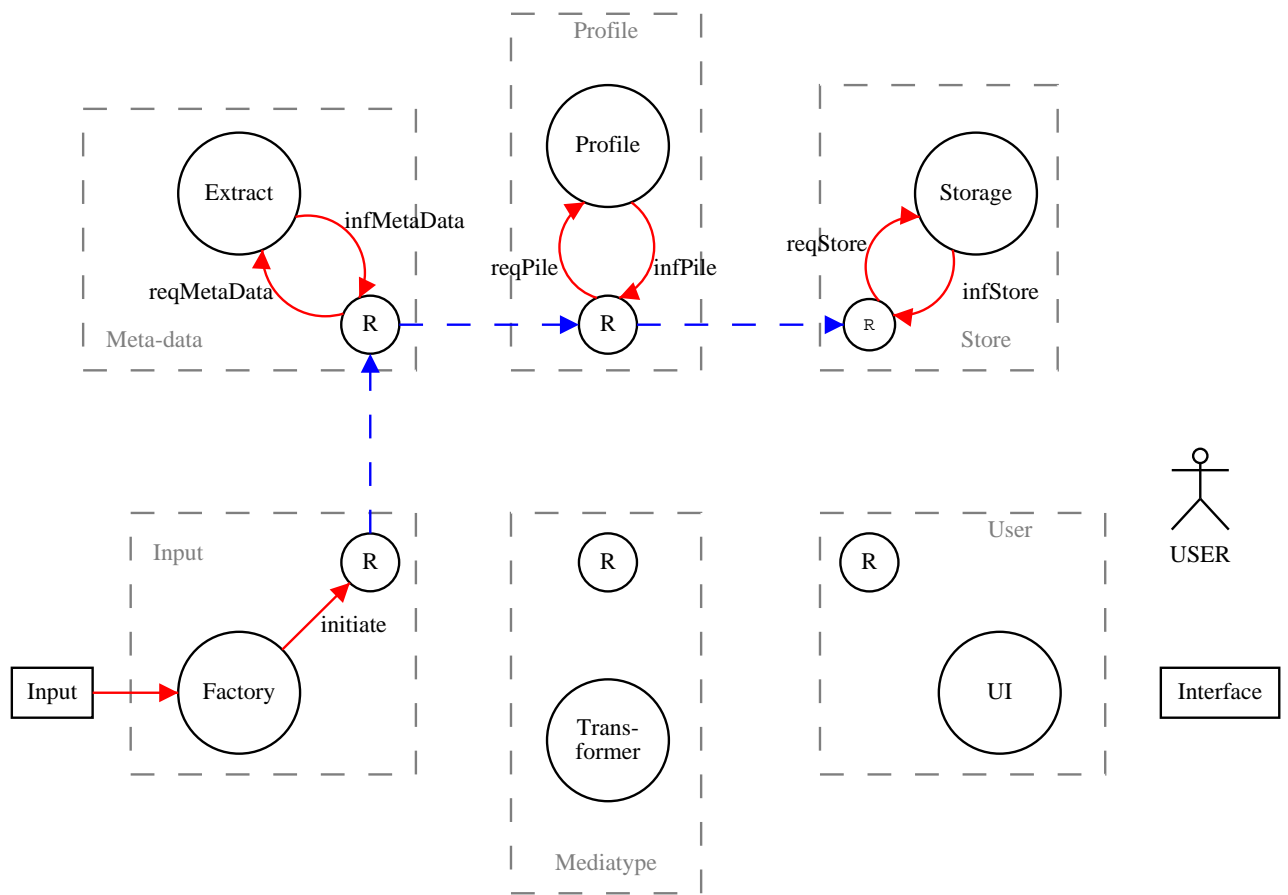


Figure 12.5: Dataflow for an unimportant new message

12.6.3 User situation change

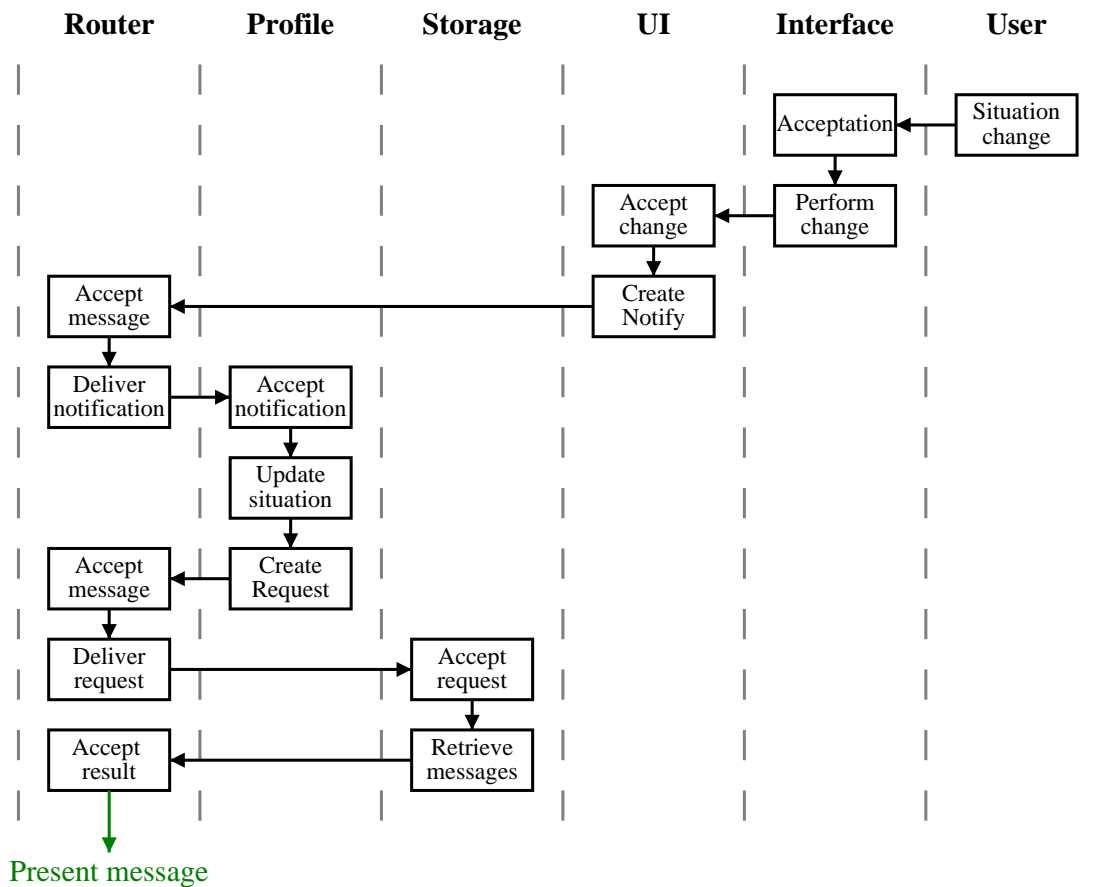


Figure 12.6: Workflow user situation change

Figure 12.6 presents the workflow for the case where the user changes its current situation. When the user changes its situation, the messages that have become important due to this new situation, should be retrieved. This could cause multiple messages to be presented to the user.

1. The user changes its situation.
2. Its interface triggers the User Interface Agent.
3. This User Interface Agent sends a notification,
4. which is routed by the Router Agent,
5. to the Profile Agent.

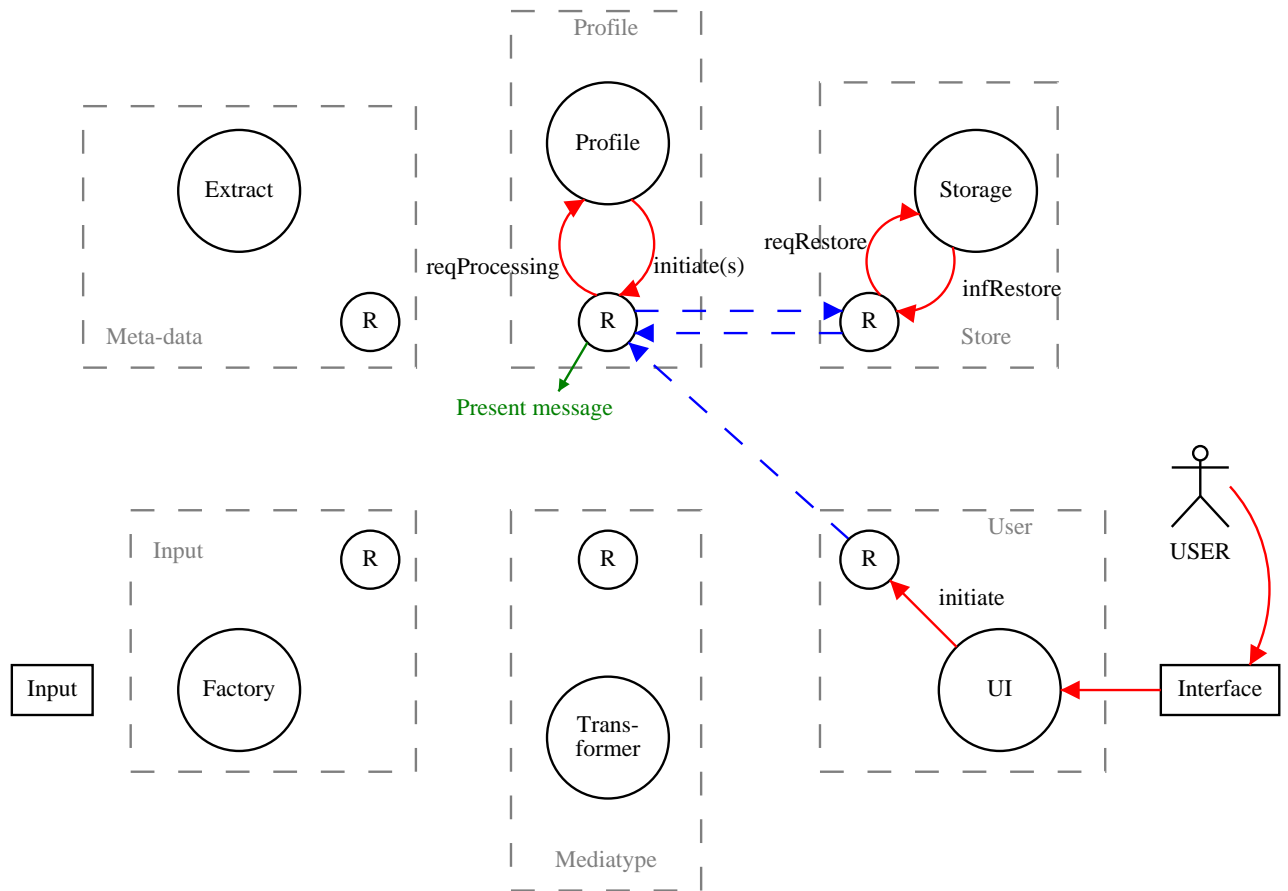


Figure 12.7: Dataflow user situation change

6. Who will update the situation.
7. Furthermore, it will create Router Agents as requests for all messages that have become important now.
8. These Router Agents place the requests,
9. at the Storage Agent.
10. The latter will return all messages with the Router Agents, where they are handled to be presented to the user. They behave similar to messages requested directly by the user.

12.6.4 User feedback

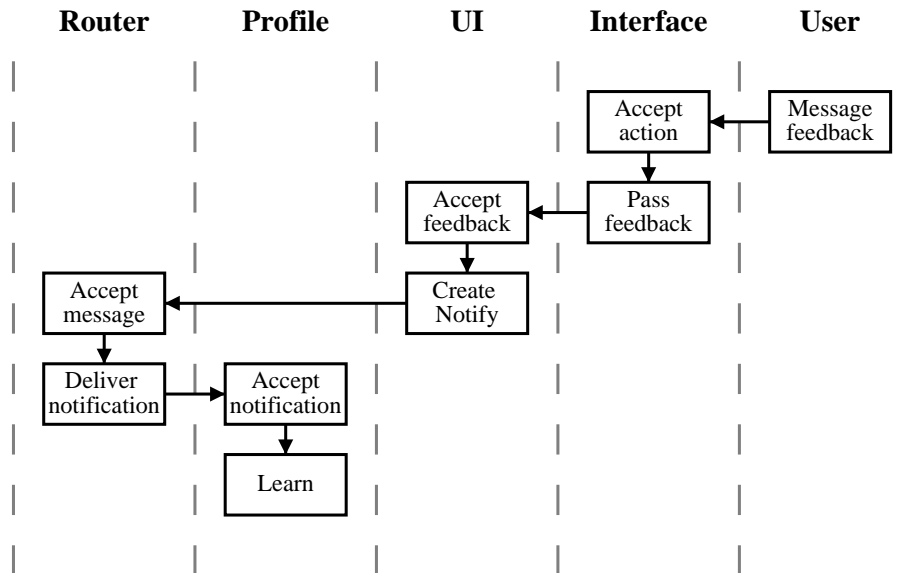


Figure 12.8: Workflow user feedback

A user can provide feedback as well (figures 12.8 and 12.9).

1. An user provides feedback after receiving a message.
2. The user's Interface will trigger the User Interface Agent,
3. to send the feedback.
4. This is routed by the Router Agent to the Profile Agent.
5. This will learn from the feedback, to improve its performance.

Although the diagram shows the Router Agent to move with the feedback, there is no statement regarding the *moment* of movement. Since the Router Agent is autonomous, it has some freedom of its movements. The Router agent therefore can wait with its actual movement until favourable conditions exist, for instance until an open (or less expensive) network connection is already established.

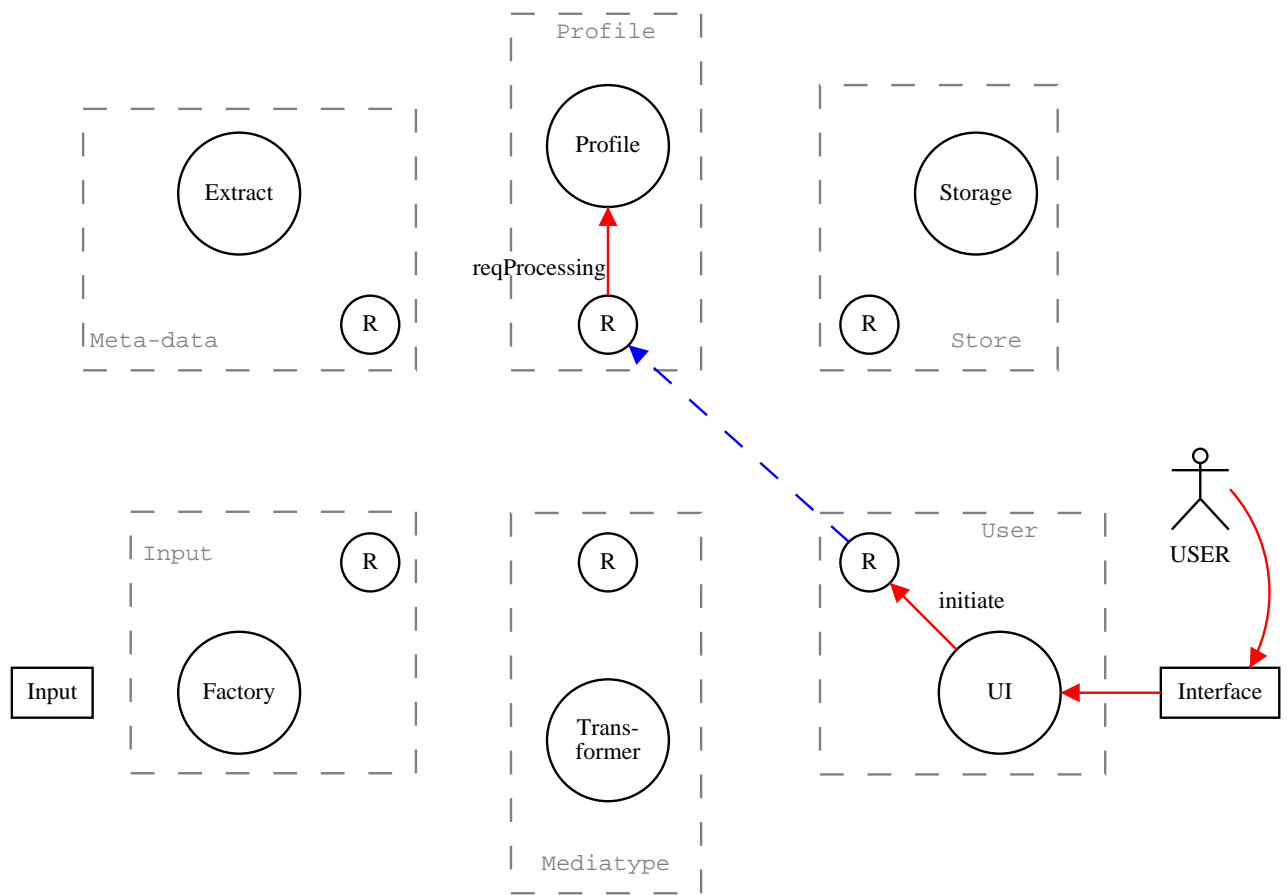


Figure 12.9: Dataflow user feedback

12.6.5 Store message

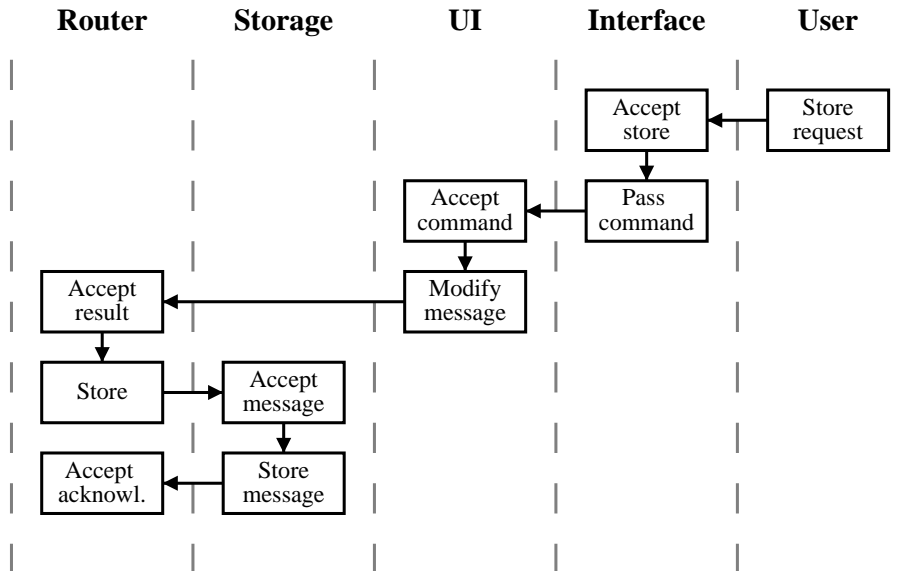


Figure 12.10: Workflow store message

The figures 12.10 and 12.11 represent the situation where a user wants to store a message. Similar to the handling of feedback, the move of the Router is not necessarily immediately. Note that storage can only happen after a message is received or requested, and is thus a reaction to the presentation of the message (see 12.6.7).

1. The User wants to store a message.
2. The user's Interface will trigger the User Interface Agent,
3. to annotate the message with storage information.
4. Next, it is routed by the Router Agent,
5. to the Storage Agent, where it is stored.

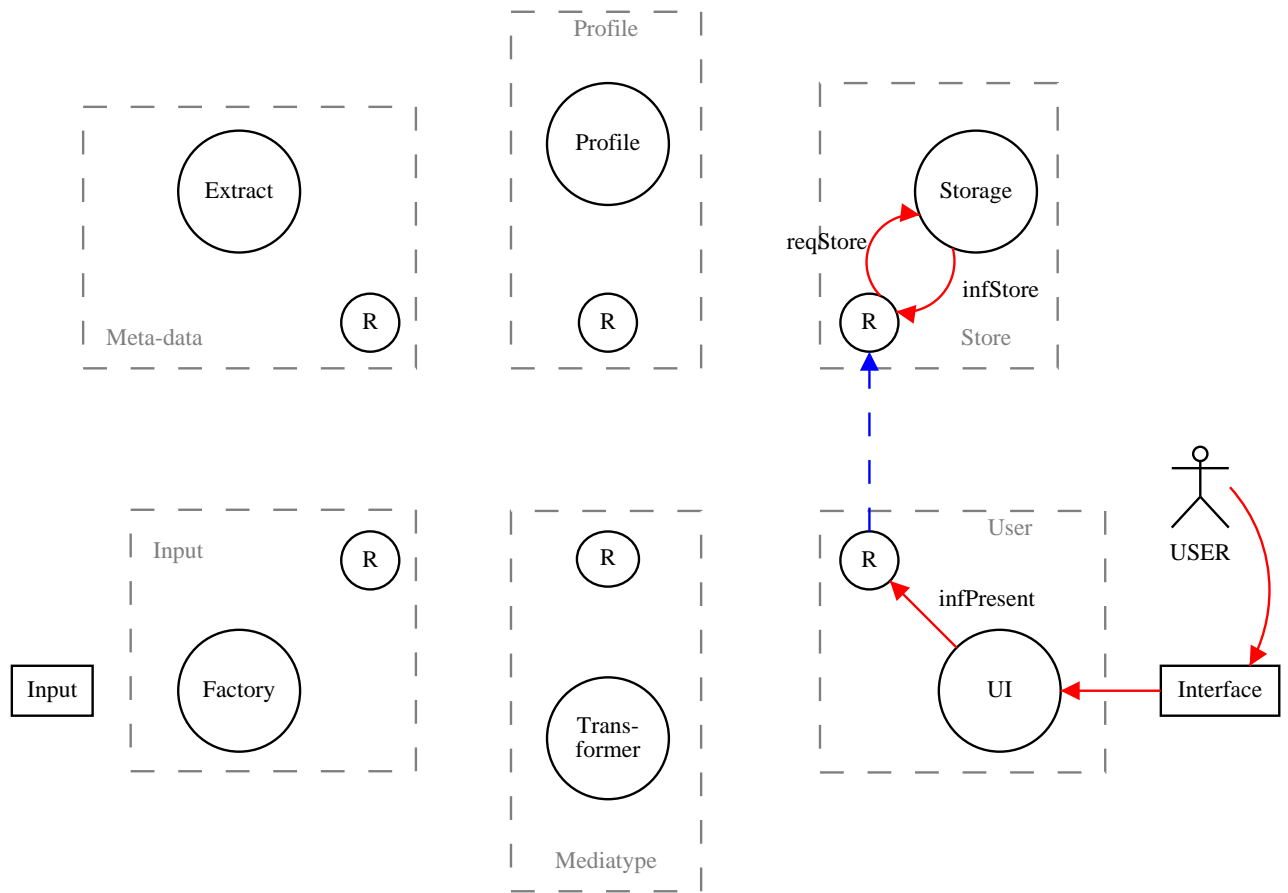


Figure 12.11: Dataflow store message

12.6.6 Request message

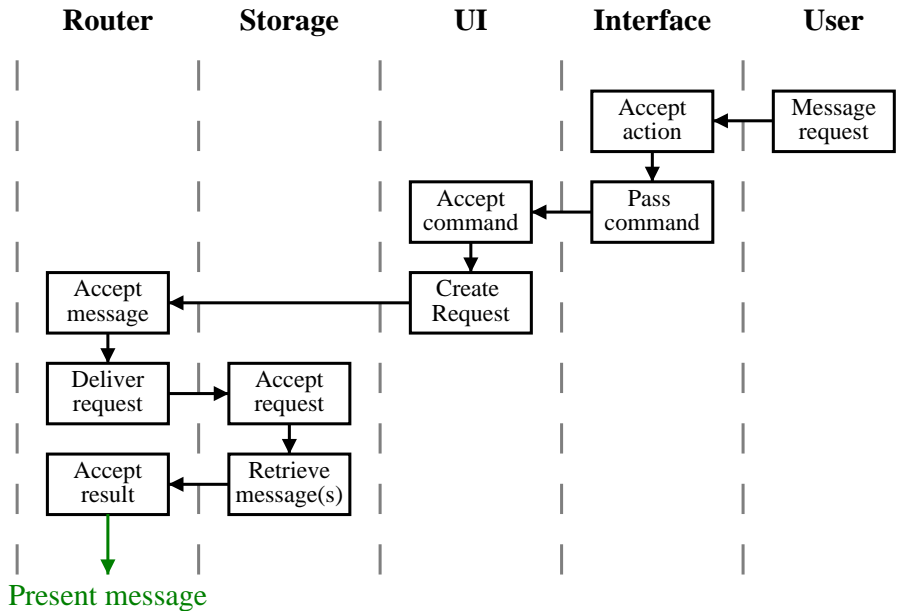


Figure 12.12: Workflow request message

The opposite of storing a message, is of course requesting a message from the storage. A request to restore a message is a special kind of message, that is initiated by the UI Agent. The Router Agent does not start directly with the message, but has to retrieve the message at the storage first. Each requested message has to be shown, hence only a format is requested from the profile. This workflow also ends with the presentation.

1. The User requests a message.
2. The user's Interface will trigger the User Interface Agent,
3. to send a corresponding request.
4. This is routed by a Router Agent to the Storage Agent.
5. This will post the requested message to the Router, to be handled for presentation.

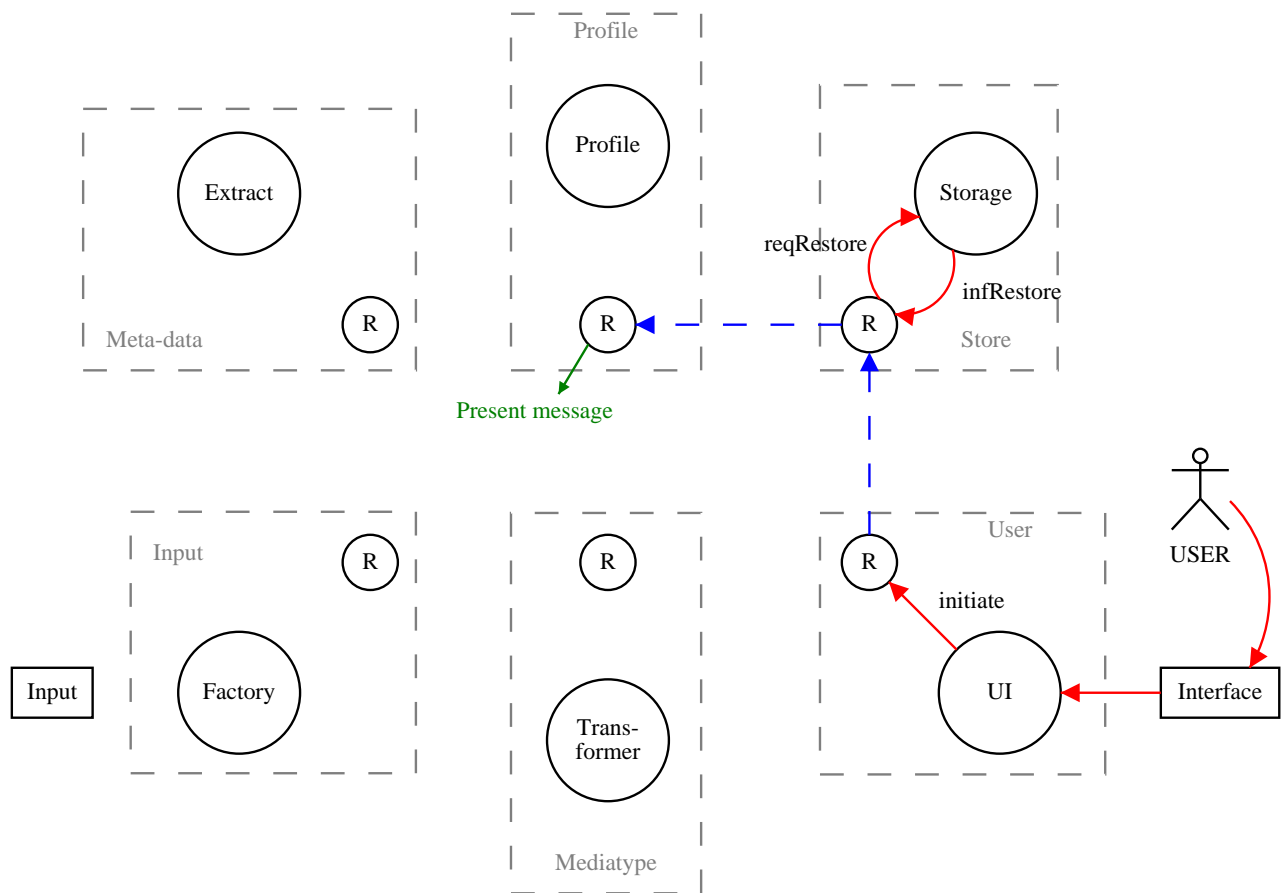


Figure 12.13: Dataflow request message

12.6.7 Present message

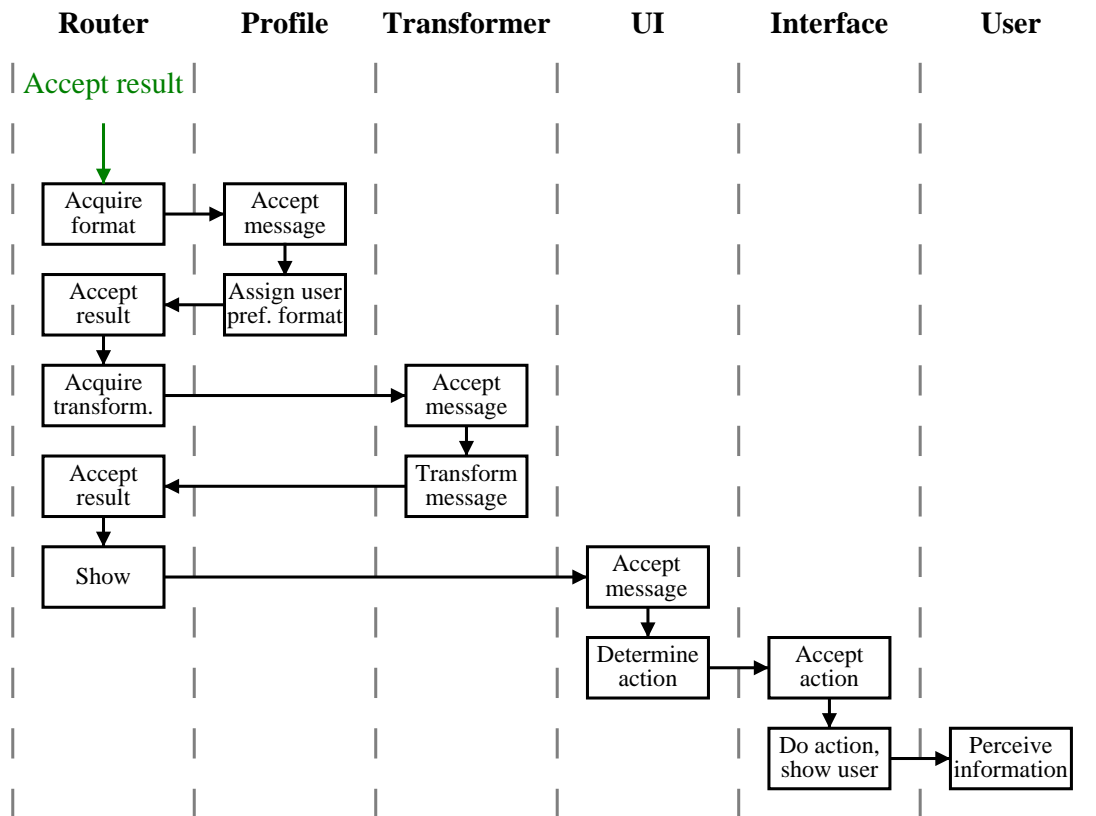


Figure 12.14: Workflow present message

The flows for the presentation of a message are equal for several of the previous mentioned situations. They are given in figures 12.14 and 12.15.

1. A Router Agent has a message to be presented, and sends it to the Profile Agent.
2. This agent decides the format to be used, and returns the message.
3. Now the router takes the message to the Data Transformer Agent,
4. where it is transformed and returned.
5. Next it is taken to the UI Agent, where it is given to the Interface to be shown to the user.

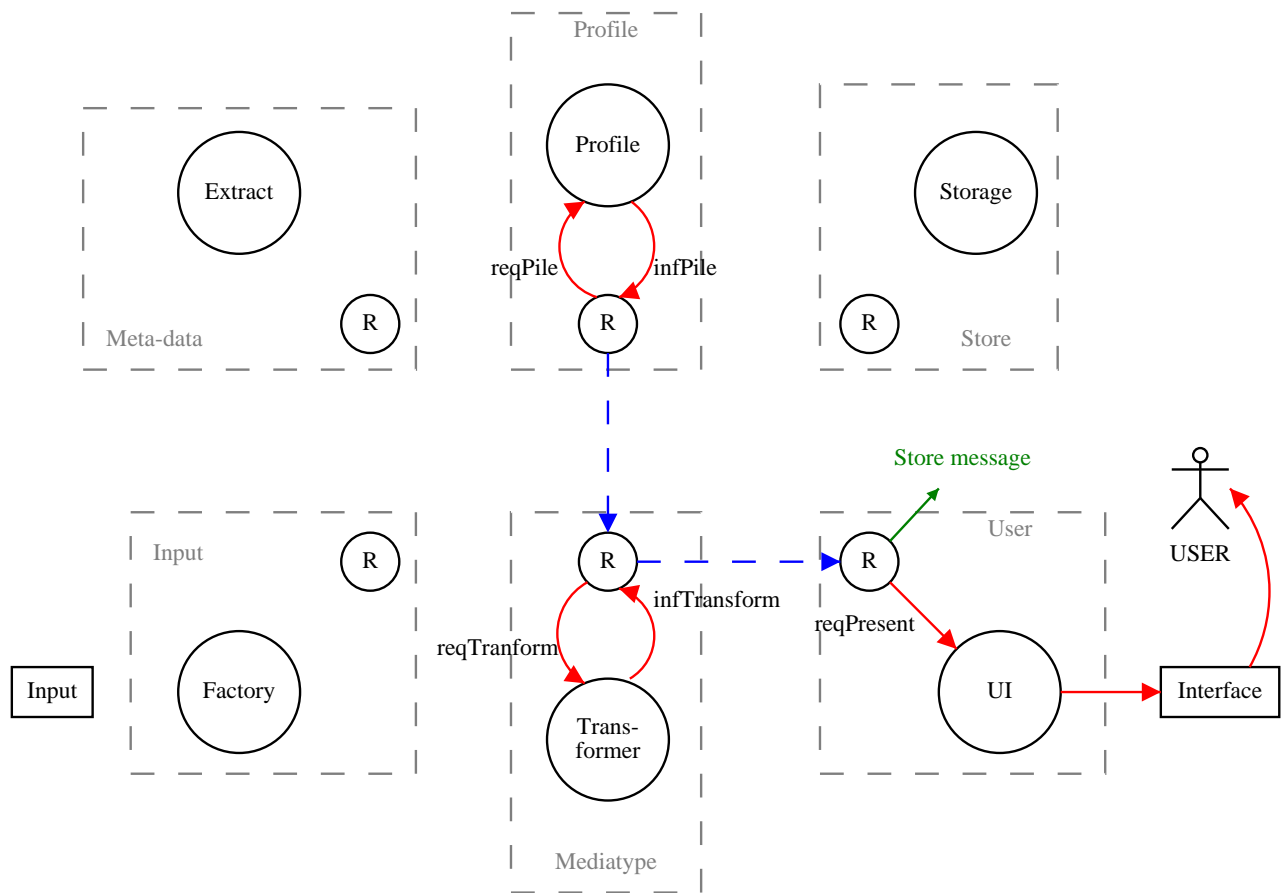


Figure 12.15: Dataflow present message

People might notice a weakness exists in this model. Once a format is chosen to present a message, and a transformation has happened, the user's situation might have changed. This could even involve the situation that the user is not reachable at all. In this case, a reevaluation can occur, which is described in more detail in the task-model (section 14.1.2, in particular in Figure 14.4 on page 124 and its description).

There is an important reason for the move from the Transformer Agent to the UI Agent. Although this move is the move of an agent through a network, this implies the move to the user's (physical) location as well. The UI Agent is located at the user's device, or directly connected to it. The move of the Router Agent is towards this UI Agent, thus towards the user. If the user is at home for instance, this moves indicates a movement the user's home-network. Would the user be travelling, the Router Agent delivers the

message through the user's telecom operator. The Router Agent's movements therefore brings the message to the user.

Chapter 13

Anticipations

Some further considerations, that do not directly affect the design, but are worth mentioning anyway. They start with learning capabilities for the profiler, followed by some ideas for addition of other functionality. These are anticipations for the feasibility and future developments of the proposed architecture.

13.1 Profile learning

One of the main aspects not covered by the architecture is the Profile Agent itself. In the architecture it is considered to be an user profiling agent capable of learning to improve future decisions. Seen from the architecture point-of-view, this is an implementation. An important issue therefore is to know if such an agent is possible. This learning problem is described below.

Note that no choice in favour of any algorithm is made here. Due to the use of a separate, *personal*, profile agent, different algorithms can be applied for different users. Replacing this agent allows to test the new profile algorithm, allowing to select the best fitting one. The list of algorithms presented in section 7.3 is neither complete, nor restrictive. It rather shows there are possibilities to use, not leaving the design with an unsolvable problem. The optimal algorithm will likely be a complex combination of algorithms, which will differ per user. Static rules as part of this profile method are neither excluded.

13.1.1 Learning problem

Before we can describe a solution for a learning-problem, a description of the problem itself is needed. A brief description with some examples is

given here. In principle, the Profiler has to assign situations¹ to a message, based on its meta-data. Feedback can be provided by the user regarding the decision made, to be used as positive and negative examples. In short, we learn decisions for situations based on meta-data.

For a learning algorithm in general, two domains are essential: the input, and the output. Other parameters like number of examples, possible error of judgement, etc. . . are not discussed, since performance is not considered to be in the main scope of the project. Started is with a short description of the output-domain, and the more distinct input-domain afterwards.

On the output-side of the profile, zero or more situations² have to be given. A situation is user-defined, thus can be regarded to be a label. No further assumption for correlations are used here, as those can differ per user, and complicate possible expression of situations dramatically. A hierarchical definition could be useful, but is more complicated as well. The use of labels as output allows a great degree of freedom in possible algorithms, many algorithms are supposed to be able to deliver such a set.

The input into the Profiler exists of meta-data. Meta-data has a broad meaning, thus should be evaluated further here. In general one can say meta-data exists of an amount of fields³, each with a semantical value, and their contents. Typical meta-data has four important degrees of freedom:

Number of fields The number of fields may vary per message, depending on the extractors available. Addition of a specific extractor can add one or more fields. Message Factories can generate meta-data as well, which can vary per device.

Fields present Fields present with one message, can be omitted with other messages. The presence of fields depends heavily on two things: source (input device) and extractors. The former defines the meta-data that arrives with the message itself (e.g. time, size, sender), that varies per device. For the latter one can think about extractors that are media-specific (e.g. voice-stress-analysis to determine the urgency), or those that depend on available data (such as an known contact when using address-book lookup to resolve the real name from a sender's device- or protocol-address).

Heterogeneous content Not only is there no (given) relation between the separate fields, their type of contents can vary too. An urgency determination can be seen as sortable values, while a sender's address is a

¹Formats are handled in a similar or even simpler way.

²For simplicity, assigning an extra level of priority is neglected.

³A field is often called attribute in machine learning.

full-text label. Another possibility is a list of values, for example when using a topic or keyword determination extractor.

Unknown content The content value of each field is not defined beforehand. When using dynamic extractors, new values can appear with each message. Factories that provide meta-data allow for varying content as well, e.g. since messages can arrive from nearly any sender.

Of course, two other problems cannot be neglected. These are quite common in machine learning [47] and user profiling however. The first is that in the described design, the learning of a profile should happen unsupervised, thus fully automatic. Another is the fact that users can behave inconsistent or make mistakes during feedback.

Conclusion hereof is that, within certain limitations, the meta-data is structured, but has no fixed structure⁴. Furthermore, the possible structure is not necessarily known in advance, due to replacing or adding agents in an existing system. Using non-fixed input-data *does* limit the possible choice of algorithms, since many require fixed-length or fixed content types. Those algorithms are thus not suitable for easy — on the fly — adapting to the user's situation, profiting from added capabilities in other parts of the system.

In 7.3 a few approaches for learning by machines are described. The algorithms stated in that section are considered reasonable candidates for profiling learning as stated here. Most of these can easily be adopted to cope with heterogeneous fields, as well as various available fields. More details of various machine learning algorithms can be found in for instance [47].

13.1.2 Examples

As a demonstration of the involved complexity, a few (hypothetical) examples of possible messages are given. The meta-data is given and the related classification is described as well.

Example 1 Given the simple fact that this is marked to be from a customer, this message is assigned the situation *work*. If the user's current situation is *work*, the message is forwarded immediately, otherwise it is stored till the user is at work.

⁴ One could argue that the meta-data can be grouped to a certain extent, i.e. meta-data related to sender, receiver, content, etc. . . Although this can be used in an implementation, it does not have a large impact on the point made. Within the limited structure of these groups, the same issues still apply.

Table 13.1: Meta-data example 1

Sender	012-345-6789
Source-device	fax
Size	3 pages
From	PaysMeWell Inc.
Relation	Customer

Example 2 In this case, there is reason to believe this message is indeed urgent. All possible situations could be assigned, allowing to reach the user as soon as possible.

Table 13.2: Meta-data example 2

Sender	987-654-3210
Source-device	voice-mail
Length	16 seconds
From	Brother
Urgency	Highest
Time	03:42

Example 3 Now the profiler could make the decision to assign this message no situation at all, since this is considered to be *spam*. This message is therefore not stored on any *pile*, but just in the storage, where it actually could result in discarding the message if spam is not stored at all.

Table 13.3: Meta-data example 3

Sender	joe@foo.bar
Source-device	e-mail
Spam-rating	0.76
Time-of-arrival	15:03
To	every-user@entire.planet.com

13.2 Extension

Although the required scenarios have been covered by the presented architecture, it would be useful to be able to cover other scenarios as well. Some alternate scenarios to extend the currently designed functionality are presented below. They are accompanied by a rough solution, but this shall be far from complete or mandatory. Scenarios to integrate with other systems and concepts are discussed as well, under the same conditions.

13.2.1 Automated translations

An useful feature that could be added is found in the fundamental concept of communication. The whole basis for unified messaging, as explained before, is to transfer a thought from one person to another. Though unified messaging will solve this by delivering this without the restriction of the used medium, people can still use different languages. Including automatic translation allows people with different native languages to communicate more easily. Quite decent automated translation tools are available nowadays (for example [1]). The functionality described above can be added to the design.

There are two possible solutions to be used here. The first is the addition of a phase in the entire process. As explained at the choice to have Router Agents, addition of a phase requires two adjustments. The Routers will have to incorporate a new phase, before or after the transformation. Another agent has to be created, which handles the translation (one could call it a Babel Agent).

This poses one problem however. Current implemented technologies often require a regular text format to be used. Therefore, extra transformations might be required, introducing their own risks. Integrating translation with the Transformer Agent might thus be another reasonable option. In this situation, the definition of the format, as used for indicating preferred formats of messages, has to be extended. A language parameter should be incorporated in a message's format. The plan formulated within the Transformer then can incorporate the translation.

13.2.2 Sending messages

All of the described functions so far, have been related to receiving messages. An important condition for receiving messages is that someone (or something) has send these. The user itself should be able to send messages as well, if only it were to respond to received messages.

This could actually be integrated with the existing system. Some of the components should be extended or adapted, but that is to be expected when adding functionality. The concept used to receive messages can mostly be adopted though. Some phases can be left out, and the direction has to be reversed however.

The UI Agent has to be able to create messages, which can be in any format. After the message has been created, the *virtual address-book* described at the Extractor can optionally be used to find the (device-)address. Now, a format can be assigned, based on the chosen device, through a preference agent or just as plain facts. The Transformer now can be used to achieve this format for the message. The last step would be to post a message with a *reversed* factory, which posts a message to another system instead of receiving messages from other systems.

13.2.3 Synchronous communication

Although synchronous communication (direct two-way communication like telephone) was left out of scope, a few things can be said. The described design can integrate synchronous information nonetheless. Half of the system will be unused, since it cannot be applied. The Router Agents have to be adjusted to leave these parts out, and their functions should be mainly to *set up* a connection. The remaining can be:

1. A Factory receives an incoming call, and generates a Router Agent, which will set up the call.
2. The Extractor is mainly left out, only the address to name translation can be incorporated.
3. The Profile can be used, but will have less meta-data to make a decision.
4. The current UI Agent must have the capability to use synchronous communication of course.
5. When a connection will be set up, the UI Agent will handle the remainder of the call.
6. Otherwise, the Router Agent can return to the Factory, which can switch to *voice-mail-mode*.

Note that (near) real-time constraints are in place when setting up a synchronous connection.

13.2.4 Enhanced dynamic extensibility

Although most of the designed agents will not change all that often, even further improvements can be added to the concept. Two agents in particular are designed to be extensible. Both the Extractor Agent and the Transformer Agent can be exposed to additional capabilities. The Extractor Agent can be extended with new extractor capabilities to improve its informativeness. Transformer Agents can incorporate additional transformations, to improve quality of transformation or possible transformations.

In order to maintain full dynamic extensibility, an enhancement in the proposed concept is possible. Instead of creating a single agent that both plans and performs the transformation or extraction, multiple agents can be used. The Extractor and Transformer need to be replaced once, with an agent that supports the used of extensions of transformations in external agents. The main agent will do the planning, while the external agents do the actual work. To provide the Router Agent with a single point of access, the planner remains the “central” point of contact. This agent will transfer its work to the actual performing agents. These agent can register at the planner-agent using the FIPA-performative `subscribe`.

This will not dramatically alter the design though. Both agents behave practically equal. Only difference will be that the performing agents are not directly contacted. The delegation of the actual task to other agents will not be noted⁵. It does allow easy addition or replacement of individual extractions and transformations. The single task can be added by launching an agent with just this task, that subscribes at the planner-agent.

13.2.5 Other agents from the user

Perhaps one of the most discussed type of agents is the personal assistant. In this relation, the agent is the user’s digital agenda, travel-agent, archiver, bookkeeper, etc... You could say the agent is supposed to be a full *digital secretary*. One task of such a secretary would of course be communication management. The described system can be integrated to handle this after a little modification. Other functions could be designed as agents as well. This leads to interaction between these agents, one example of which is described below.

An example of an agent would be a calender agent. This agent will keep track of the user’s appointments, and will make those with other people (or their agents). Notifications to change an appointment might be sent by e-mail. These messages should thus not be sent to the user itself, but have to

⁵Except perhaps some additional delay.

be handled by another of its agents.

A general solution to catch communication designated for other agents is the following. The Extractor should be extended with a subscription mechanism, as described in the previous section. An agent that expects messages can subscribe as an extractor. Beside from extracting meta-data, it should mark the message with a field to route it to itself. The profiler should be adjusted to route such messages to the appropriate agent.

Note that in this case the agent really uses the system for its own benefit. These agents can be used to improve the system's performance as well. A personal calendar agent can be included to mark messages regarding short-term appointments. The profile can utilise this information in the decision whether the message is important.

Part IV

Prototype

This part about the build prototype starts with a chapter about the technical design (chapter 14). This is rather generic, and could be used for any implementation. Chapter 15 describes the created implementation based on this design. Together they form the prototype that was build for the designed system.

Chapter 14

Technical Design

After the findings of part [III](#), we have a functional and global design. Based on this design, a technical design was made as well. This is a further refinement of the previous presented design. Since the design prescribes the functions each agent has to fulfil, these function will first be specified. The results are presented in the first section. The second section, from page [133](#) and further, partially describes the object model. This model is the last step before the actual implementation, and a few samples of this object model will be given.

14.1 Task model

In the following sections, the tasks of each agent are described. This is done with both task hierarchies and task state transaction diagrams. Both type of diagrams are explained in [appendix B](#).

Task-hierarchies display the relation between tasks. It does not imply any transitions between tasks, nor any transfer of data or decisions. A task-hierarchy represents the mutual relation between tasks, identifying the main tasks and their subtasks.

The accompanying task-states do show these transitions and decisions. These task-states show which steps are taken during the process, and when branches can occur. Regarding the task-states one has to keep in mind that missing failures are normally fatal ones. Some of these are not defined, like failings during storage of a message, and are left for further development.

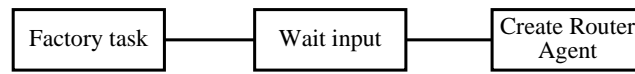


Figure 14.1: Task hierarchy of a Factory Agent

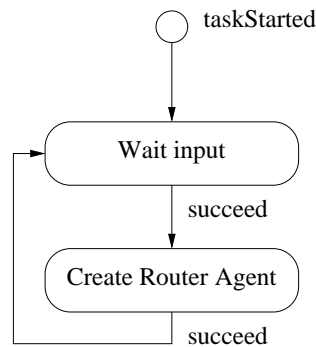


Figure 14.2: Task states of a Factory Agent

14.1.1 Factory Agent

A Factory’s main task is feeding messages into the system. The messages are either received by a waiting task, or polled at regular intervals by a similar task. Once received, it is packed as a message for the system, together with the known available meta-data (such as time of arrival and sender) and the current format. With this package, a Router Agent is created, which starts at the state of a new message.

14.1.2 Router Agent

A Router Agent “simply” helps the message through all required phases of processing. First it will attempt to acquire meta-data. Next, the Profile is given a chance to assign situations and a priority to the message. When a message has to be shown immediately, it will ask the Profile for a format. Next, it requests the Transformer for this format. Last, it is shown to the user via the UI Agent.

Some states are not always successful, due to (network-)failure, unknown media-formats, or other causes. This is where alternative scenario’s have to take over. In case no meta-data can be assigned, the message is to be judged without additional information. When no format can be assigned (or is not needed), a direct attempt to show the message in the current format is made. When a transformation cannot be accomplished, another format is requested.

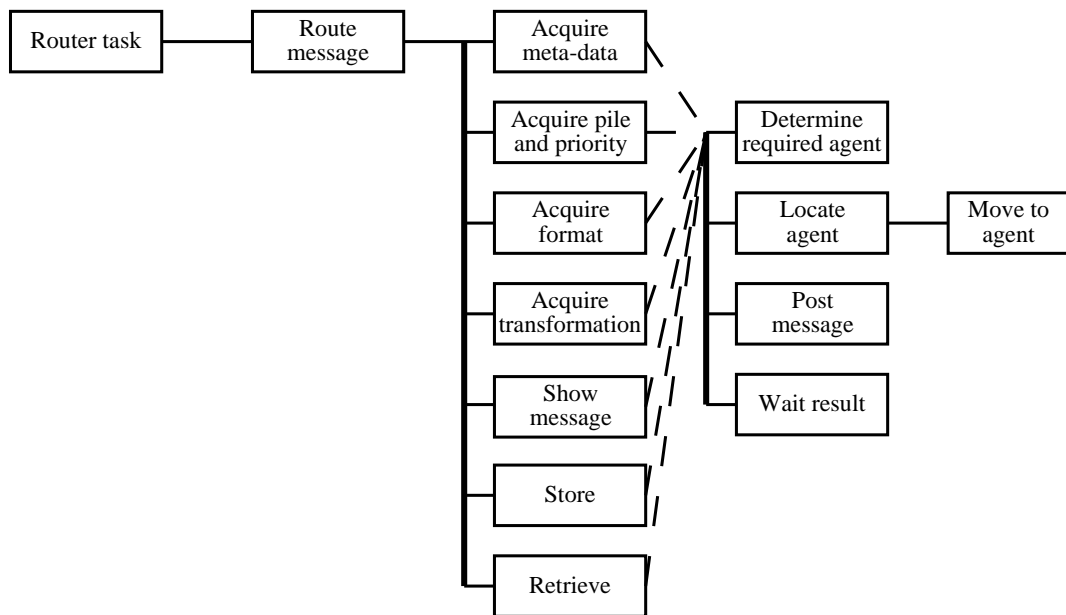


Figure 14.3: Task hierarchy of the Router Agent

If a message cannot be shown, another attempt to request its priority can be made, assigning it to either (return to) store, or make another attempt to be shown. Errors while storing or retrieving messages are fatal in this design.

When a Router Agent starts as a request to restore a message, it starts to retrieve the message. Next, it enters the same trajectory as a message that has to be shown now. Note that a message that enters the store returns in this state.

Phase details Note that each *phase*-task (in Figure 14.4) is actually a task consisting of multiple other tasks. All are basically the same: Find the agent, move there if needed, post the message and await the result. These are described below, and are shown in Figure 14.5. Only difference between these phases is the actual data that is communicated, as described in 12.4 and shown in the dataflow diagrams of 12.6.

A remark about finding an agent needs to be made: Finding another agent is not wandering around and see if its here. The infrastructure needs to support some type of name lookup. This theoretically requires all places to have a mechanism for such a lookup, which should be up-to-date for all locations. Although this implies a (massive) distributed system, this is not considered a problem. Such systems already have been employed on large

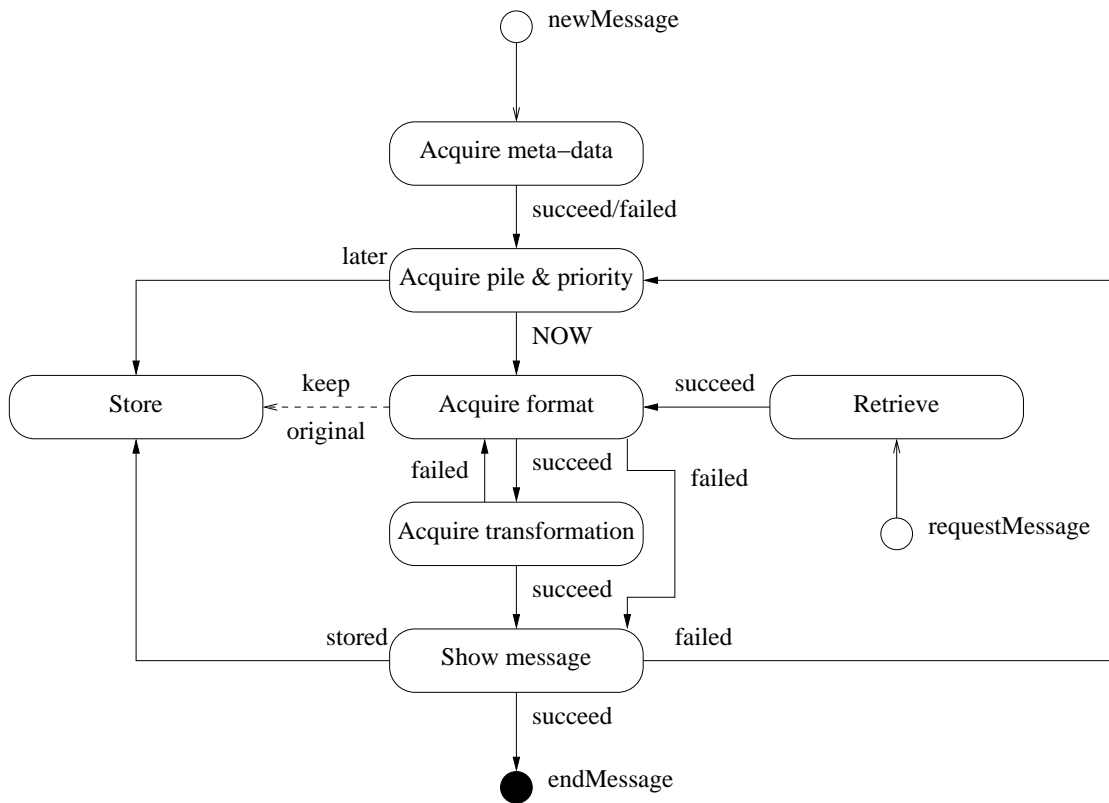


Figure 14.4: Task states of the Router Agent

scale based on open standards, implemented for nearly all networked systems in the form of DNS [49], which can quite easily be adopted.

After the required service is determined (which follows naturally from the *phase*-task), the agent that provides that service¹ is determined. Such an agent is located, and when it cannot be found locally, the Router moves to another location (where it is likely to find one, or a route towards one, making moves recursive). If this service can be found at the Router's location, the message is posted there. Now the Router waits for the result, which is used to update the message and / or its meta-data. Note that each move can be replaced by passing the message to another Router Agent, including the message's current state, which is essentially the same as a movement. Another remark has to be made regarding robustness. An agent that executes a move is supposed to move uncorrupted (errors introduced due to problems during the transport). If such corruption can occur, the task should include

¹The Extractor, Profile, Transformer, UI and Storage Agents.

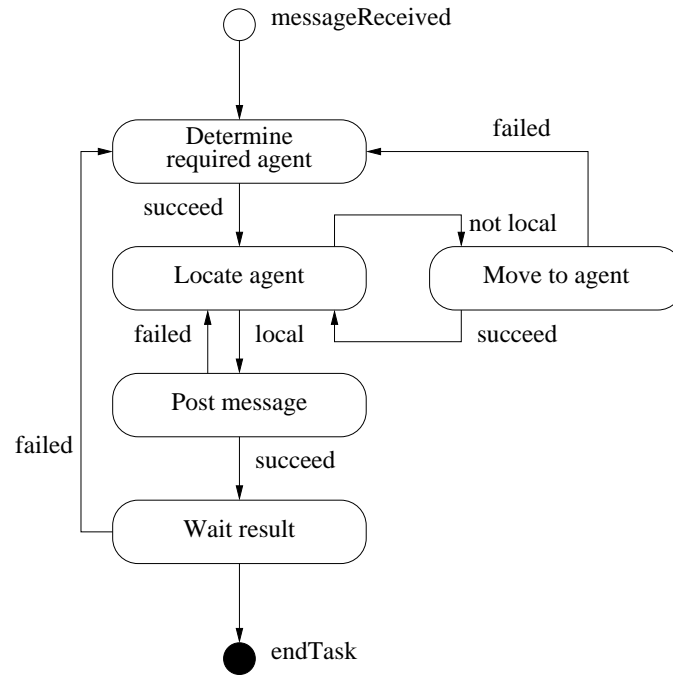


Figure 14.5: Task state of each Router Agent phase

an additional mechanism to recover from errors².

An important function is fulfilled by the *locate* and *move* in the phase task model. This move is essential for the distributed nature of the agent system. All agents in the system are bound together because the Router Agents can find them. In particular the move for the phase to *show* a message to the user is crucial. The move in the show-phase effectively brings the message to the user (and its location). This provides for the anywhere promise of unified messaging.

Specialisation Last thing a Router Agent can do is transport a notification from the UI Agent to the the Profile. These notifications contain the special messages with feedback or updates in the user’s situation. Essentially, this is the same as the (sub-)task shown in Figure 14.5. It needs to locate the Profile Agent, and deliver the notification. As a result, this forms the connection from the user back to the system through the network.

²Such as checkpoints, where the current situation is *frozen* and can be restored.

14.1.3 Extractor Agent

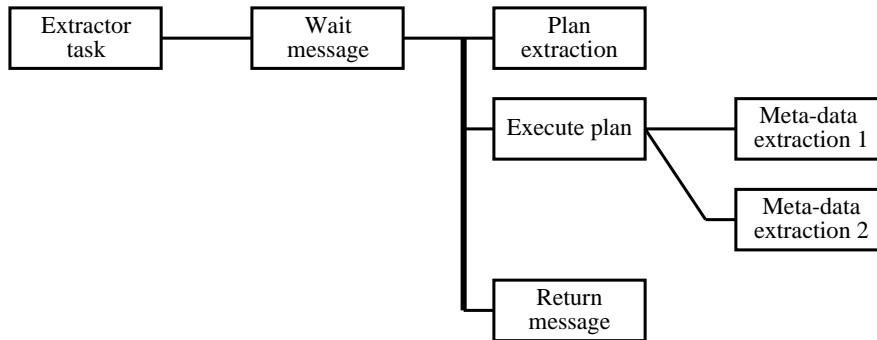


Figure 14.6: Task hierarchy of an Extractor Agent

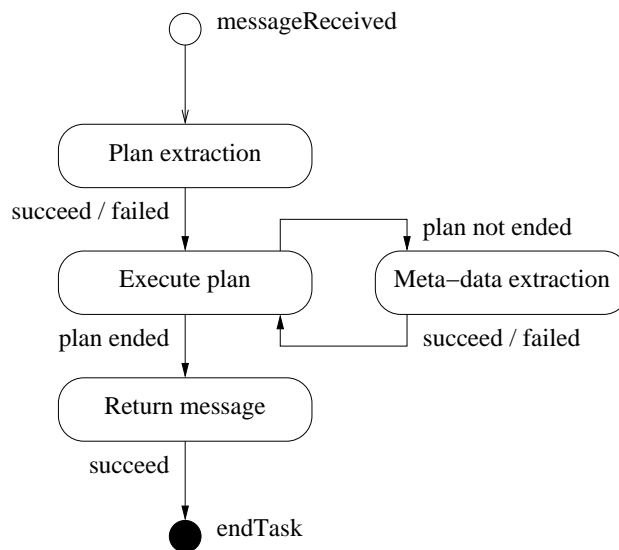


Figure 14.7: Task states of an Extractor Agent

The Meta-data Extraction Agent is a reactive agent. When a request to process a message arrives, it examines which meta-data extractions can be applied. This is formulated as a plan. Next, this plan is executed by performing each meta-data extraction task. Each such task annotates the message with additional meta-data. When an extraction fails, the plan therefore can continue, but without the additional meta-data. After the entire plan is executed, the result should be returned to the originating Router.

Extended extractions One may notice this design does not allow to dynamically add new extraction capabilities to this agent. In this design one has to replace the Extractor to add functionality, rather than add a new function to the existing Extractor. This is not difficult to realise however, as described in 13.2.4. One should replace the Extractor once, with an extraction task that allows other agent to *subscribe* as another extractor. Once a message arrives at this new extractor, it can be handed to other subscribed extractors with new capabilities as well. Only one additional task is needed, which allows a remote extraction task (handing the message to another extraction agent), and adjust the planner accordingly.

14.1.4 Profile Agent

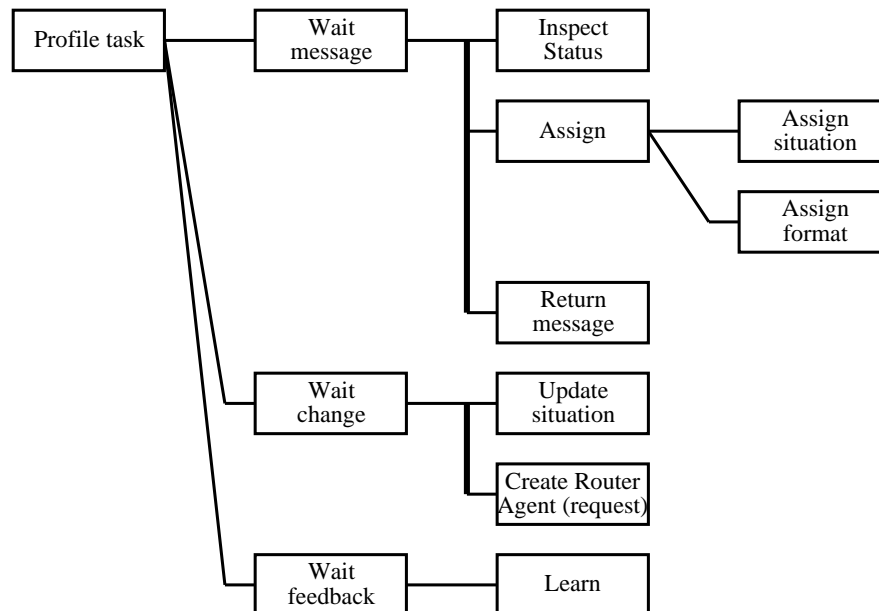


Figure 14.8: Task hierarchy of a Profile Agent

The Profile Agent is reactive to three events. First of all, it reacts to messages. When a message arrives, it inspects the message which service has to be provided. Next, either applicable situations or the preferred format are assigned. Afterwards, the result is reposted to the originating Router.

The second reactive task is related to the update of a user's situation. After receiving such a notification, the known current situation is updated. Next, requests are created to present all messages that have become impor-

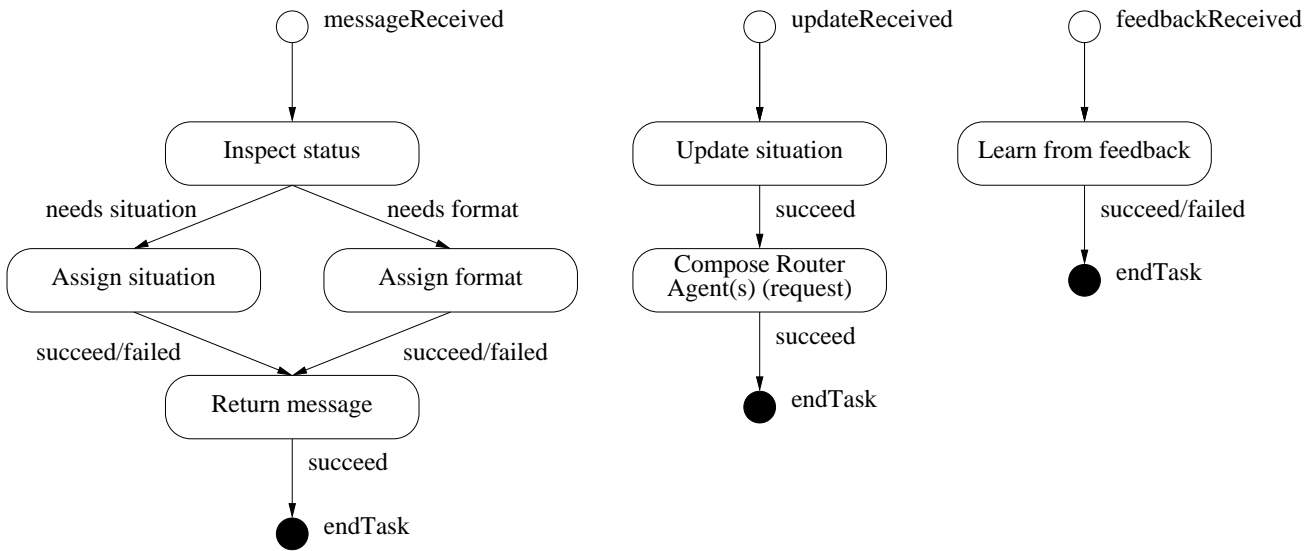


Figure 14.9: Task state of a Profile Agent

tant in the new situation. These requests are created as a Router Agent, each of which starts in the request state.

The third and last task of the Profile Agent is to receive feedback notifications. Learning from this feedback is the only subtask performed here.

14.1.5 Transformer Agent

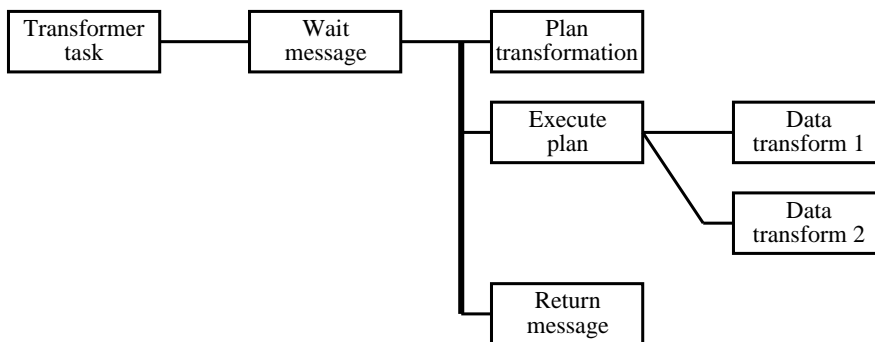


Figure 14.10: Task hierarchy of a Transformer Agent

The Transformer Agent handles conversion of one media-format to another. An arriving message includes the original message in a given format,

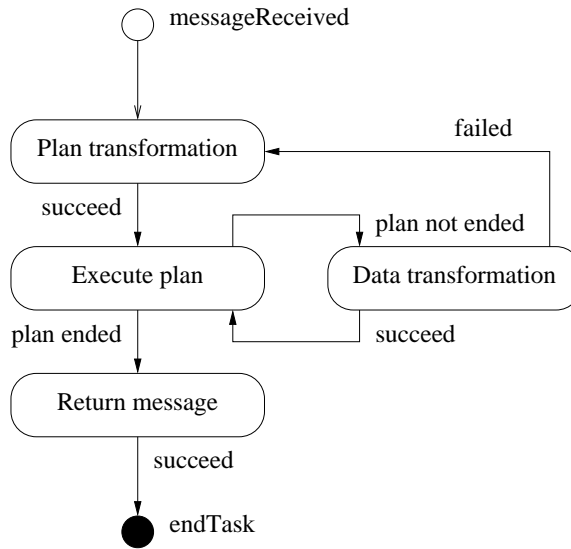


Figure 14.11: Task state of a Transformer Agent

and a destination format. The destination format has to be reached by the Transformer. First, a plan to reach this destination must be made (compare with the *Automatic Path Creation* found in [56] or the *Conversion Planner* of [58]). Next this plan is executed, and the result reposted. Note that this whole transition is similar to the Extractor. The only difference is the handling of a failure of a transformation step. Since the created plan is sequential instead of additional, a failure disables the entire sequence. Therefore, a new plan has to be created that avoids the failing path. When a succeeding plan cannot be established, the transformation cannot be made.

Extended transformations As with the Extractor Agent, it is possible to dynamically add new conversions with only a slight modification. Similar as described on page 127, one has to add a registration task. This registration should allow the agent to incorporate a remote task in the conversion plan.

14.1.6 UI Agent

Since the User Interface Agent handles all possible actions of the user, it has four tasks. The first is the only one triggered by the system, the event of an arriving message. The last three are caused by the user itself, namely the feedback for a message, changing the current situation, and requesting a message. Notice the other operations with regard to the storage are not

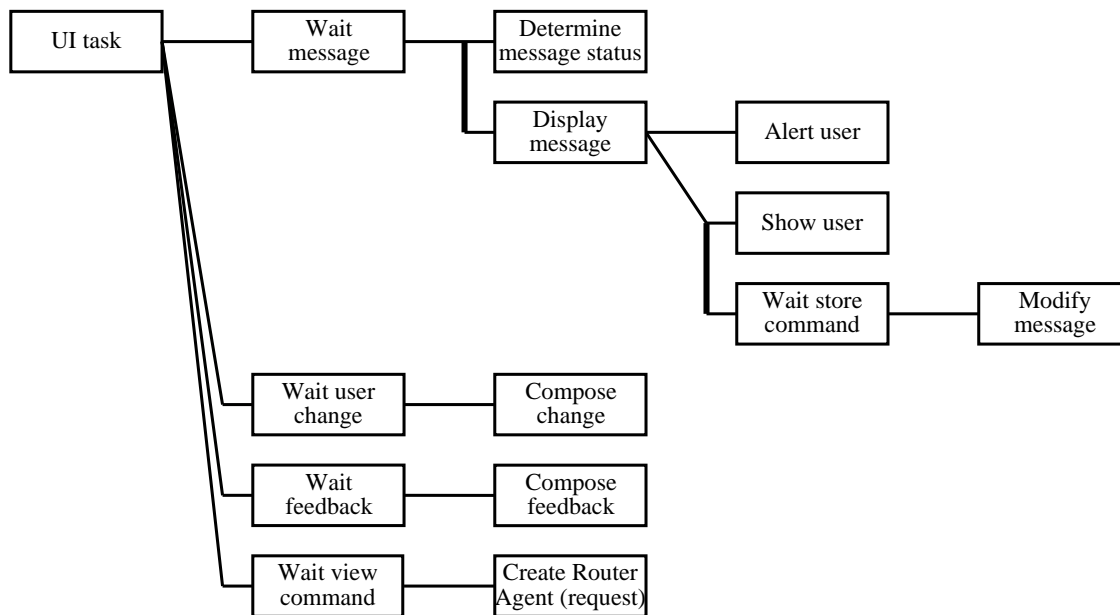


Figure 14.12: Task hierarchy of an UI Agent

described.

An arriving message is examined upon arrival. If it is a new message, the user is alerted in an appropriate fashion. When the user requested the message, it is shown. After an alert, the user may request to view a message. When a message is shown, the user can optionally store the message. In the latter case, the message is modified with additional storage instructions, and returned to the Router.

When an user provides feedback, a Router as notification is created. This is essentially the same as a normal Router Agent, only limited to take the special message (notification) to the Profile. When the user changes its current situation, a similar task is performed.

A command to view a message is handled in an equal way. The only difference is that instead of a notification a request for a message is created. This includes a request to receive a list of available messages, which is a special request for the headers of multiple messages.

14.1.7 Storage Agent

A Storage Agent handles 2 events. The first is a request, either directly from the user (through an UI Agent) or caused by the Profile Agent (to request important messages after a change in the situation). The second is

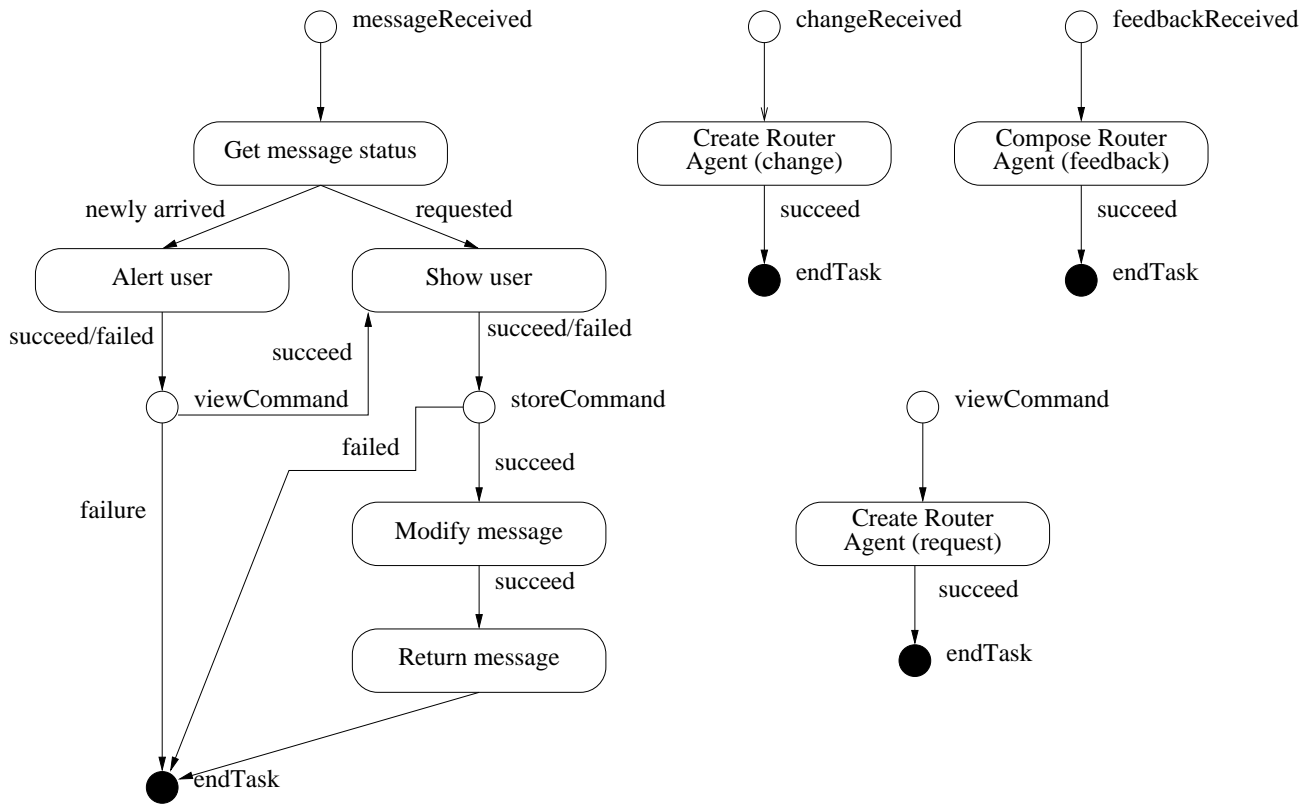


Figure 14.13: Task state of an UI Agent

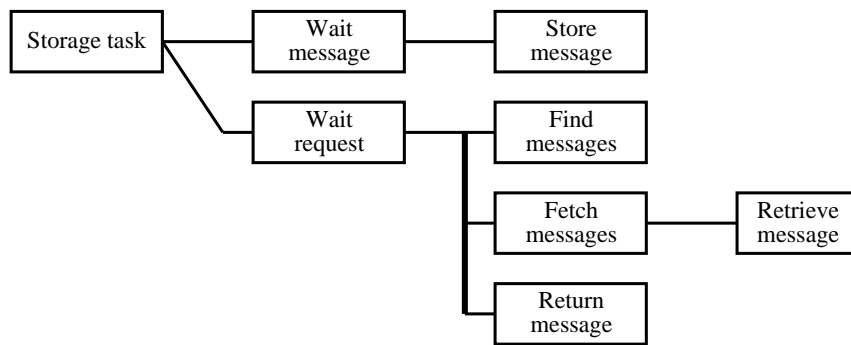


Figure 14.14: Task hierarchy of a Storage Agent

an arriving message, that has to be stored, from the same sources.

The first case starts with finding the message or all requested messages. Next, the message(s) have to be retrieved from the storage, by fetching them

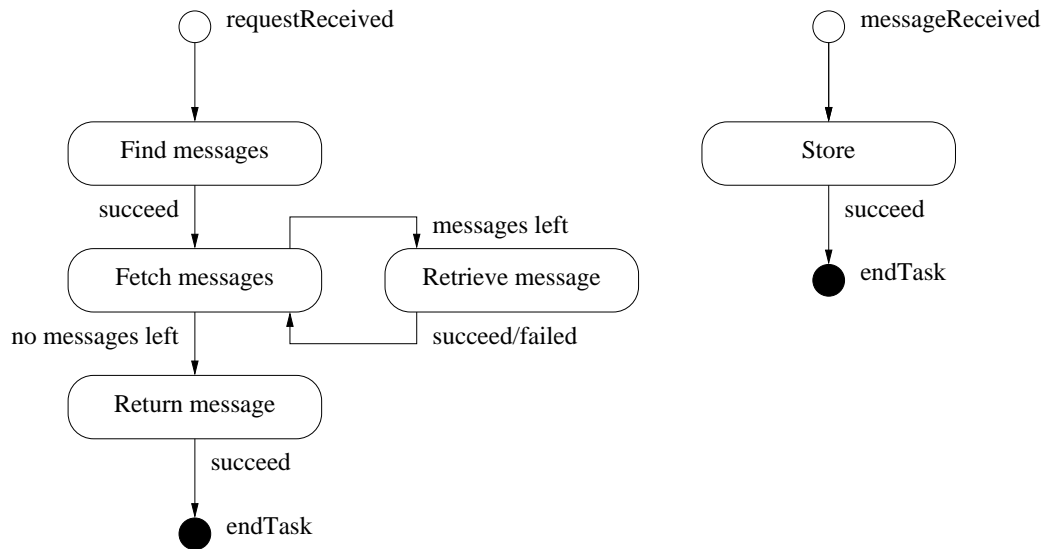


Figure 14.15: Task state of a Storage Agent

all. After fetching, they are returned to the Router Agent. Note that retrieval of multiple messages is accounted for in order to retrieve lists of available messages or aggregated messages.

Storing a message is pretty straightforward. A message with storage-instructions (i.e. with which indexes, in which folder, etc...) arrives. The message is stored according these instructions. Other operations are not described in detail here, but can be handled in a similar fashion.

14.2 Object model

Figure 14.16 shows a global view of the object model. This is presented in standard UML, although no attributes nor any methods are given. For the Router Agent in the design, a more detailed diagram will be given next. The other agents will not be described any further. Due to the size of the diagrams only the selected agent will be presented in this document. Note that this one is not entirely complete, the diagram is reduced for practical reasons. The implementation is commented with javadoc, which can provide a more detailed description. Note that the `Agent` and `TaskScheduler` are classes from the Tryllian ADK [75].

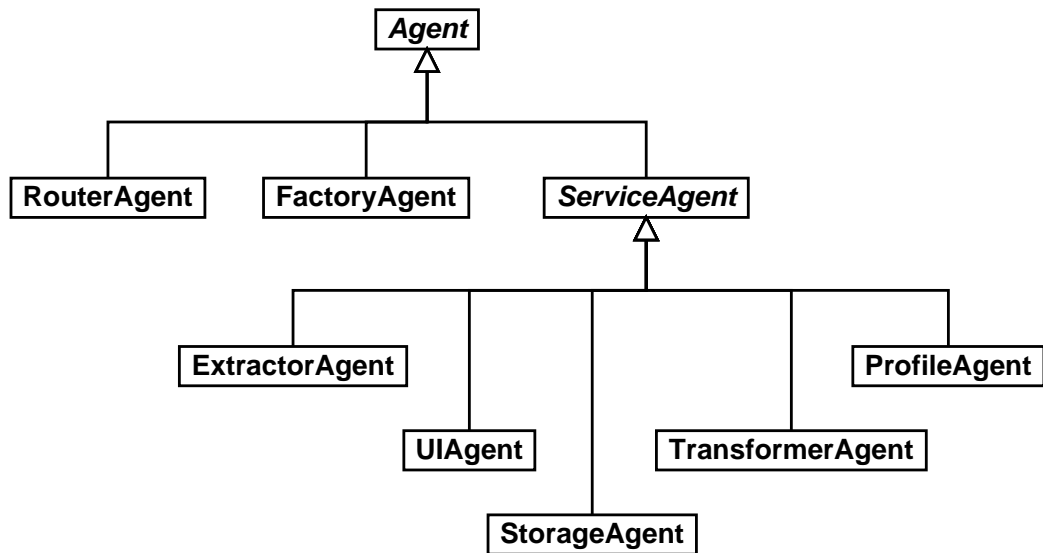


Figure 14.16: UML object model for agents

14.2.1 Router Agent

The object model of the implemented Router Agent looks rather complex at first sight. It involves a lot of inheritance however, since most tasks are only small variations of a more generic task. Figure 14.17 shows a part of the object model of the Router Agent. If one compares it with the Router Agent's task hierarchy and task-state transition (Figures 14.4 – 14.5), one can recognise each task. The easy way the process to handle a message can be composed on this basis is demonstrated in 15.5.1.

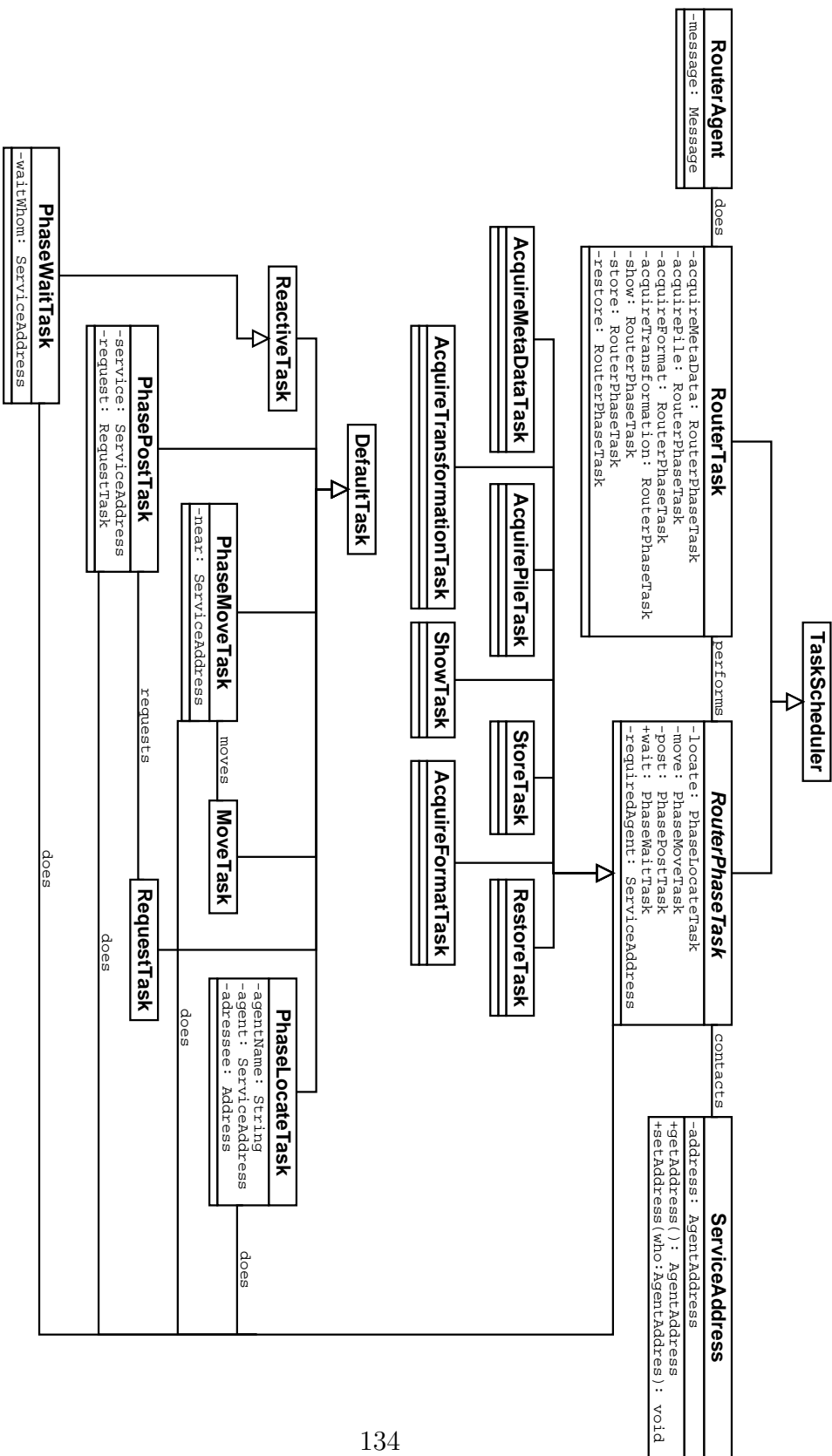


Figure 14.17: Partial UML for the Router Agent

Chapter 15

Implementation

This chapter describes the translation from design to the implementation of the prototype. It mainly describes the implementation of the prototype. It is not meant to describe design choices for the prototype. Note that the reader is expected to have prior knowledge of the design of the system.

15.1 Environment

The entire implementation has been based on Java 2 Standard Edition [69] (version 1.3) and the Tryllian ADK [75]. Java is a platform independent OO-language, running in a Java Virtual Machine. It is used as the basis for the Tryllian ADK, which is described in appendix A. The latter allows easy creation of mobile agents, creating agents with their tasks. As development system a standard Linux desktop computer was used. The User Interface Agents (see 15.9) were run under MS Windows as well.

15.2 Messages

A message consists of two parts, corresponding with the design. The first part is a descriptive part, containing all headers. These include headers added by agents throughout the process. The second part consists of the message itself.

15.2.1 Envelope

The first part of a message consists of headers. These headers are expressed as a XML-document [8]. Using XML has some advances, among which two important ones. The first is easy modification between versions, thus allowing

addition of headers without directly disturbing existing data or an agent using an older versions. The second is the availability of existing parsers, which allows for easy implementation.

```
<message_envelope
  id='1-machine.net-RemcoSchaar-1016453553467-6877721874412959'>
  <routing type='new'>
    <routeinfo
      type='tryllian.rschaar.common.message.Addressee'
      name='addressee'>
      <addressee id='RemcoSchaar'>RemcoSchaar</addressee>
    </routeinfo>
    <routeinfo
      type='tryllian.rschaar.common.message.FormatHeader'
      name='format'>image/gif</routeinfo>
    <routeinfo
      type='tryllian.rschaar.common.message.SituationHeader'
      name='situation'>home,work</routeinfo>
  </routing>
  <headers>
    <header
      type='tryllian.rschaar.common.message.StringHeader'
      name='device'>SMS: 0800-fakeSMS</header>
  </headers>
  <type>
    <format>text/plain</format>
    <parameters>
      <parameter name='lang'>en</parameter>
    </parameters>
  </type>
</message_envelope>
```

Figure 15.1: Sample envelope

A note has to be placed here. The implementation actually provides for two sets of headers. The first (**headers**) allows for headers that are actually headers for the message itself. The second part (**routing**) provides a part for the agents to place notes regarding the delivery. This second part is included to allow minor exchange of information between agents (like assigning a folder where the message should be stored), without the requirement of specialised communication between all different types of agents.

15.2.2 Content

The contents of a message is relative simple. It is a representation of the contents of the message as a sequence of bytes. The interpretation of these bytes is left to the using agents, based on the accompanying MIME-type [19] of the message.

15.3 Agent interaction

Agents communicate among each other. In the implementation this communication is kept as simple as possible. All communication is performed with one type of interaction. A `RouterAgent` always¹ initiates all interaction. It places a request to *process* a certain message, with the actual state and further information in the envelope. As data for such a request, the entire message (both envelope and content) is exchanged. The advantage of this approach is simplicity in implementation, although it implies that agents may receive or alter more information than actually needed. Section 19.6 has some further discussion on this topic.

15.4 Factory Agent

Description of the build Factory Agents. Three Factory Agents have currently been implemented. With these three agents, quite different sources are covered:

- E-mail.
- RSS (site summary, news sites article references).
- SMS.

15.4.1 E-mail Factory

Most Internet environments provide electronic mail as a means of communication. This factory is based on polling an IMAP-server at regular intervals, with the standard `javax.mail`-API [68]. New mail is delivered through the system.

¹Note that a `RouterAgent` itself is commonly initiated by a `Factory`.

```
<?xml version="1.0"?>
<backslash xmlns:backslash="http://slashdot.org/backslash.dtd">

  <story>
    <title>KaZaA Collapses</title>
    <url>http://slashdot.org/article.pl?sid=02/05/23/020245</url>
    <time>2002-05-23 09:30:26</time>
    <author>Hemos</author>
    <department>end-of-the-world-as-we-know-it</department>
    <topic>95</topic>
    <comments>39</comments>
    <section>articles</section>
    <image>topicinternet.gif</image>
  </story>

</backslash>
```

Figure 15.2: Sample article reference (RSS) from Slashdot.org

The E-mail Factory can handle attachments as well. Whenever a new message arrives without a textual body, but with an attachment, this attachment is send as the message. This way, nearly every format can be send.

15.4.2 RSS Factory

News sites like Slashdot and Linuxtoday provide a standardised way to check for new articles. This is provided through RSS (Rich Site Summary). By checking a small file at a fixed location, one can check for new articles. New articles on a site can thus easily be spotted. A sample, reduced to a reference of only one article, is shown in Figure 15.2. This sample is taken directly from Slashdot.org (“News for nerds, stuff that matters”), a technical news site. Please note that RSS has various (incompatible) versions, but are all based on the XML-standard [8].

15.4.3 SMS Factory

This is a fake factory. It allows to address a message to a certain user, with just a short text. It is thereby in function equal to SMS as used with most GSM-based mobile phones.

15.5 Router Agent

A description of the implementation of the Router Agent. The Router Agent is the only mobile agent, and achieves this by implementing the `MoveTask` [74].

15.5.1 Task model

According the design, the Router Agent has a task model for all jobs to do. Each Router Agent starts with a clean task model. As described in the design, a message has to go through several *phases*. Each of these phases corresponds with a set of tasks in the Router Agent, which are very similar to each other through OO-inheritance. These are scheduled (using a `TaskScheduler`) to form the complete set of actions that have to be performed with a message.

The task-model as shown in Figure 14.4 can be implemented as shown in Figure 15.3. This code example is somewhat reduced, mainly for aesthetically reasons. Some of the declarations, imports and comments are therefore not shown.

A task is added to the scheduler with the `addTask` method. The shown method's arguments represent the subtask that is added, the task that is scheduled next if the subtask succeeds, and the last argument is the task scheduled next if the subtask fails. Simple rearrangement of these task thus allows to change the routing process.

15.5.2 Locating Service Agents

One of the most important tasks in each phase is locating the agent that will actually do the work. These agents are dynamically located, using a lookup mechanism. The Router agent gains the address from a combination of DNS [49] and JNDI-registry [65]. Each user has a so called *Context*, in which the Service Agents can register themselves. When a selected Service Agent is not local, the Router Agent moves to the location of the particular Service Agent. DNS is used to resolve the remote systems for an user. A remote JNDI-query is performed to check for other agents.

15.6 Extractor Agent

A description of the implemented Extractor Agent. The Extractor Agent consists of multiple tasks, bound together by a planning task. Each extraction task has a corresponding filter. This filter is used to test whether the task

```
public class RouterTask
    extends TaskScheduler
{
    // Constructor
    public RouterTask (UM message) {

        // create and initialise tasks
        acquireMetaData = new AcquireMetaDataTask(message);
        acquirePile = new AcquirePileTask(message);
        acquireFormat = new AcquireFormatTask(message);
        store = new StoreTask(message);
        restore = new RestoreTask(message);
        acquireTransform = new AcquireTransformTask(message);
        show = new ShowTask(message);
        theEnd = new DieTask();
        failure = new LogTask("unrecoverable_failure");

        // task transition logic
        addTask(theEnd);
        addTask(failure, theEnd);
        addTask(restore, acquireFormat, failure );
        addTask(store, theEnd, failure );
        addTask(show, store, acquirePile );
        addTask(acquireTransform, show, acquireFormat);
        addTask(acquireFormat, acquireTransform, show);
        addTask(acquirePile, acquireFormat, store);
        addTask(acquireMetaData, acquirePile);

        // now start with acquireMetaData
        setFirstTask(acquireMetaData);

    }
} // RouterTask
```

Figure 15.3: Sample code of the created task-model

is (possibly) able to extract additional information from a message. These filters can be rather straightforward, and should decide on simple heuristics. For instance, a filter for an audio-specific extraction task will only have to test the type of the content of a message. Based on this decision, a task is added to a plan. This plan is executed afterwards, resulting in the addition of meta-information to the headers of the message.

Currently, three extraction tasks are implemented. The first is a lookup for device-dependant addresses, translating them to “real” names. The second tries to find the relation of the receiver to the sender for known senders. Last is the ability to (manually) add keywords to a message.

15.7 Profile Agent

The state of the Profile Agents. Currently available in the prototype:

15.7.1 Dummy Profile

All new messages are always important for the current situation. These messages will be shown in any applicable format. The first format that is available (for the user’s device and after transformation) is applied.

15.7.2 Random Profile

The current situation is assigned to each new message with a certain chance. For the applied format, the same decision is made as above with the dummy-profile. It is recognised that the random type of profiling will have no practical value. The Profile Agent was mainly build to test² the prototype.

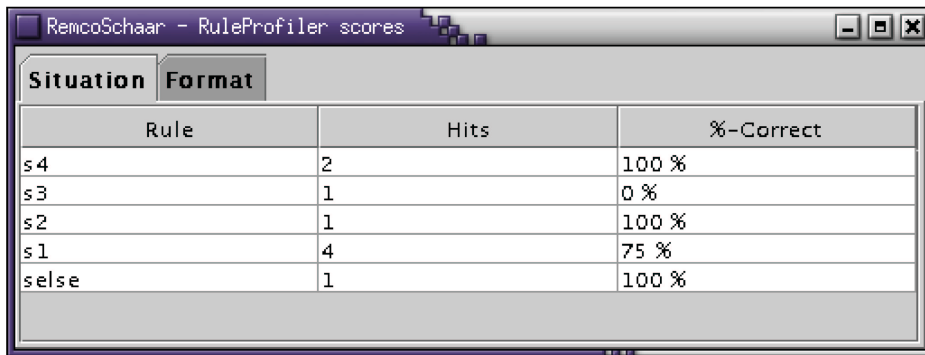
15.7.3 Rule-based Profile

An user can define rules to assign both a situation and a format to a message. These rules are tested one after another, until a matching rule is found. All tests in a rule must be satisfied, and when a rule is fully satisfied, the result is assigned and processing is stopped. The format is assigned through similar rules. Rules are written in XML for easy parsing, and are read for every message again so they can be adjusted in runtime³.

The feedback by the user to the Rule Profile Agent is used to compose some statistics. An user can itself use these statistics to improve the created

²Technically, not functionally as in chapter 16.

³Although at an efficiency cost.



Rule	Hits	%-Correct
s4	2	100 %
s3	1	0 %
s2	1	100 %
s1	4	75 %
else	1	100 %

Figure 15.4: Screen-dump of the Rule Profile Agent statistics

rule-set. A small screen-shot of this form of feedback is given in Figure 15.4. A rule with a low *score* can be improved by the user in its own advance.

```
<?xml version="1.0"?>
< rules_profile userID="RemcoSchaar" >

  <situation>
    <rule id="s1">
      <match
        header="Subject"
        matcher="equals">test</match>
      <result>home</result>
    </rule>
  </situation>

  <format>
    <rule id="f1">
      <situation>work</situation>
      <type>application/postscript</type>
      <result>text/html</result>
    </rule>
  </format>

</ rules_profile >
```

Figure 15.5: Sample rule set for the Rule Profile Agent

Rule example A small set of rules in Figure 15.5 shows an example of two rules. The first rule (id “s1”), tries to match a header named “Subject”, based on the operator “equals”. When it matches, it assigns the resulting situation “home”. Since it is contained within the section of situations, it is a rule applied to a decision for a situation.

The second rule applies to messages when assigning a format to a message, since it is contained in a section “format”. The rule here (id “f1”), is stated to apply to situation “work”, if the current type is `application/postscript`. Resulting format to be used to show the message, is assigned by the result `text/html`. Any postscript document that should be shown at work, should thus be transformed to HTML.

15.7.4 *K*-Nearest-Neighbours Profile

A learning profile has been implemented as well. This has been created as a variant of the well-known *k*-nearest-neighbour algorithm.

The used distance function is actually an inverted distance. The more similar a message, the higher the score. This similarity is determined with a very simple comparison, counting the fields that are present and equal for both messages. The mathematical equation for the used “distance”:

$$D(m, s) = \sum_{h \in (H_m \cap H_s)} d_h(h_m, h_s)$$

In this formula, *m* is the message to be classified and *s* is a known sample it is compared against. The function *D* determines the distance between two messages, based on the sum of distances of their *mutual* headers (*h_m* and *h_s*, *H* represents the collection of headers of a message). The distance between headers *d_h* is a function that varies per header *h*. A label will simply be compared whether it is equal, but a numerical header can have a mathematical distance function for instance. The implemented distance function for most headers only operates positively. Headers that are not equal are ignored, since some headers do not necessarily state a message is less similar if they are unequal. For instance the name of the sender cannot always be found when translated from device-specific addresses. This does not imply that `person@e-mail.server` is automatically another person than someone calling from 012-3456789.

The result of the *k*-nearest-neighbour function is determined as follows:

$$K(m, S) = \bigcup_{1 \leq i \leq k} \max_{s_i \in S} D(m, s_i)$$

Here, S is the collection of messages with known — by the user classified through feedback — applicable situations. Now the function K determines the resulting set of applicable situations, by taking the k samples with the maximum score. The implementation automatically uses the most recent by the user classified samples in favour of “older” samples in case two sample have the same score.

The applied algorithm can be summarised as follows:

1. Calculate the distance for the message to evaluate with all messages known from feedback.
2. Select the k most similar (highest score) messages.
3. Get the preferred situations for these selected samples.
4. Create the union of these situations, this is the result.

15.8 Transformer Agent

One Transformer agent has been created. This Transformer agent is fully Linux based, thus restricting it to a specific platform. Since this is a generic service for multiple users, in a networked environment, this is not considered a problem.

The reason to choose for a Linux specific implementation is rapid development. Since a properly installed Linux system has many command-line programs available, transformation tasks can (commonly) easily be created from a command-line. Two abstract tasks have been created, allowing easy composition of real transformation tasks. The first is a piped process, based on UNIX-pipes tied to the standard input and output. The other uses temporary files, for programs that cannot use pipes.

Confirming to the design, an overall planner decides which tasks should be linked together. This provides to create chains of required transformations, similar to those in the *universal inbox* in the *ICEBERG* project [56]. The chains are build using a simple breadth-first search algorithm.

Currently available are the following. Note that new tasks can rather easily be added when the proper programs are available⁴.

- MS Word to plain text.
- HTML to plain text.

⁴Certainly many graphical, as well as audio, converters are available for Unices.

- Text to speech, using `festival` (University of Edinburgh, [3]).
- Postscript to text.
- Adobe's Portable Document Format to plain text.
- Postscript to a Portable Network Graphic (PNG) file. Note that this is (currently) limited to the first page of a document.

15.9 UI Agent

Two User Interface Agents have been implemented. These are the Swing UI Agent and a Phone UI Agent.

15.9.1 Swing UI

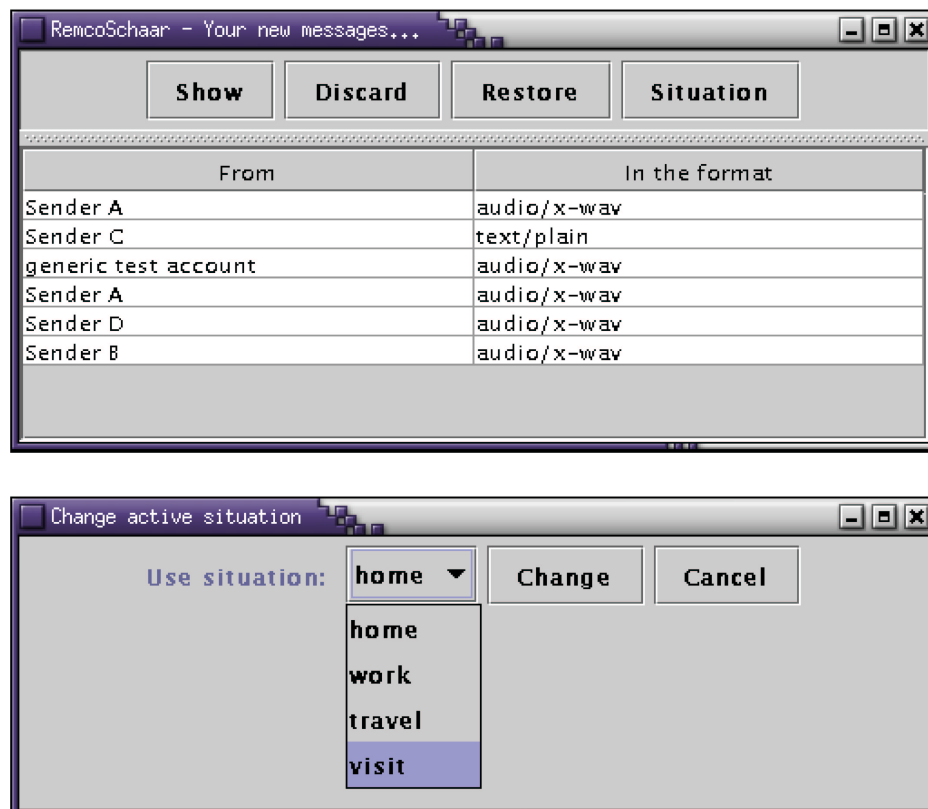


Figure 15.6: Screen-dumps of the Swing UI Agent

A desktop application is build, using the `javax.swing-API` [67]. This user interface is kept simple, emphasising the functionality. Due to the use of the standard `Swing` library, the application should be portable. Another property follows from the usage of `Swing`, the application has an adaptable look and feel. The `SwingUIAgent` should thus have a *native* interface on multiple platforms. A few screen-shots can be found in Figure 15.6.

15.9.2 Phone UI

The simulated phone actually consists of two parts. A simulation for a phone as well as a part for Short-Message-Service. Not all functionality of the design is supported, due to limited capabilities of both (corresponding real) devices. Creating an usable interface would require a study for itself.

15.10 Storage Agent

A storage has been created using JDBC [66], connecting to a MySQL-database [50]. This allows quite easily to store, retrieve and list messages.

15.11 Runtime examples

The prototype is an operational version of the developed design. Two examples of how the system operates will be given below. The first example contains a message that is, according to the user's profile, important at the current moment. The second example exists of a case where the message is judged to be applicable in other situations. Both examples given below cover the first two use cases given in 10.2.

15.11.1 Example 1: new important message

In this first example, the user will receive a new e-mail message. This e-mail is send through the system, and is judged to be important to the user's current interests. Due to this importance, the message has to be shown directly to the user. The user's current device should be considered. As a more practical note, the message itself is in plain text, and the user prefers audio as format to receive it.

As a brief explanation, the agents and rooms present here will be described. The set up of the environment consists of four rooms. Rooms are

conceptual, but are not only mentioned for aesthetic reasons, they could represent physically different machines as well. These rooms with the contained agents are:

1. **Input**, where messages are received. The agents that receive messages are contained here.
2. **User1**, which contains personal agents. The agents present here are the Extractor, Profile and Storage.
3. **Public**, containing an agent for all users. The Transformer is a generic service.
4. **Desktop**, where the user's interface is "located". The agent that handles the interaction with the user is located here.

First thing here is the E-mail Factory Agent receiving the new e-mail message.

```
[EmailFactory in /Input] Send a message: 0
```

The factory starts a new Router Agent with the message, that will carry it through the system. This Router Agent will first locate and consult an extractor agent for the user. Since the located Extractor is not in the same room, the Router Agent moves there and tries again. Afterwards the Router posts the message to the Extractor.

```
[Router-0 in /Input] Started           // A message to deliver!
[Router-0 in /Input] -> Extractor      // First consult the Extractor
[Router-0 in /Input] locate           // Locate Extractor
[Router-0 in /Input] move             // Move to the Extractor's location
[Router-0 in /User1] locate          // Check Extractor is here
[Router-0 in /User1] post            // Consult the Extractor
```

The Extractor performs a couple of applicable extractions, and returns the message to the Router Agent.

```
[Extractor in /User1] extracting      // Annotate: sender from white pages
[Extractor in /User1] extracting      // Mark with relation sender to user
[Router-0 in /User1] wait            // Wait for Extractor's reply
```

The Router Agent will receive this message and will perform its next task. This next task is to consult the Profile Agent for the importance of the message.

```
[Router-0 in /User1] <- Extractor // Done with extractor
[Router-0 in /User1] -> Profile // Next: Profile
[Router-0 in /User1] locate // Find the Profile, is local
[Router-0 in /User1] post // Ask the Profile
```

The Router now posts the message to the Profile Agent, to have it assigned the appropriate situation. Which the Rule Profile Agent does; based on the information in the header, it assigns a situation.

```
[RuleProfiler in /User1] situation // Note this one is important
[Router-0 in /User1] wait // Wait for Profile's reply
```

Since the situation is registered as the current one, the Profile Agent returns positive. After receiving this positive feedback, the Router's next task is acquiring the appropriate format to show to the user. The Router Agent responds now, by a repeated visit to the Profile for having a format assigned to the message.

```
[Router-0 in /User1] <- Profile // Done with situation
[Router-0 in /User1] -> Profile // Go get a format
[Router-0 in /User1] locate // Profile is still here
[Router-0 in /User1] post // Ask Profile for format
```

Which the Profile does, while the Router waits for the answer.

```
[RuleProfiler in /User1] format // Assign audio
[Router-0 in /User1] wait // Wait for Profile's reply
```

After the preferred format is annotated on the message, the Router will go to ask the Transformer to convert the message. The Router now locates a Transformer Agent. Since it is located elsewhere, another move is made.

```
[Router-0 in /User1] <- Profile // Done with format
[Router-0 in /User1] -> Transformer // Consult a Transformer
[Router-0 in /User1] locate // Locate a Transformer
[Router-0 in /User1] move // Move to Transformer's location
[Router-0 in /Public] locate // Check it is here
[Router-0 in /Public] post // Ask Transformer to transform
```

Transformer inspects the message, and notes it must transform text into audio. The Transformer thus transforms the message into the requested format.

```
[Transformer in /Public] transforming // Text -> audio
[Router-0 in /Public] wait // Wait for transformed message
```

Now the Router's next task is to actually show the message to the user. The Router continues with this modified message to the User Interaction Agent, which is located elsewhere as well. This thus implies another move, bringing the message to the user at its current device, and thus location. This move is to the user's device, which can be located anywhere with the user, e.g. at home, work, during travel or in a shop.

```
[Router-0 in /Public] <- Transformer // Acquired transformed message
[Router-0 in /Public] -> UI // Now go show to the user
[Router-0 in /Public] locate // Locate user's current device
[Router-0 in /Public] move // Move to user's device
[Router-0 in /Desktop] locate // Check UI Agent is here
[Router-0 in /Desktop] post // Show to user
```

The UI Agent will receive the message and show it to its user. The user now consumes the message, and adds information how to store the message.

```
[SwingUI in /Desktop] receiving // Receive, show to user
[Router-0 in /Desktop] wait // Wait till message is shown
```

The UI Agent alerted the user, and when needed makes sure the storage directives that are needed are added. It now hands the message back to the Router Agent. Last but not least, the Router hands the message to the user's Storage Agent. This agent enables the storage of the message, thus allowing persistency.

```
[Router-0 in /Desktop] <- UI // Done showing
[Router-0 in /Desktop] -> Storage // Acquire Storage
[Router-0 in /Desktop] locate // Locate user's Storage
[Router-0 in /Desktop] move // Move to location
[Router-0 in /User1] locate // Check Storage location
[Router-0 in /User1] post // Request Storage
```

The Storage receives the message, and responds to it. It inspects the message, and determines it has to be stored. Storage Agent thus stores the message in the database with the appropriate information. In this particular case, the message is stored in a specific folder.

```
[SQLStorage in /User1] storing // Storage stores
[Router-0 in /User1] wait // Wait confirmation
```

The Router Agent receives a positive response from the Storage Agent, so ends its job.

```
[Router-0 in /User1] <- Storage // Done storing
[Router-0 in /User1] job done // Message fully processed; done
```

15.11.2 Example 2: new unimportant message

In this second example, a few modifications to the previous example have been made. First of all, the message now arriving is considered not to be important at the moment. The situation assigned does not correspond to the current situation, so the message should be stored for a later time.

The second major difference is the lack of an Extractor agent, in this scenario it does not exist. The layout of rooms and agents therefore differs as well. The room `User1` does not contain an Extractor Agent.

The first few steps here correspond to the other example. The E-mail Factory Agent receives and packs a message, and launches a Router Agent with this message.

```
[EmailFactory in /Input] Send a message: 1
[Router-1 in /Input] Started           // A message to deliver!
[Router-1 in /Input] -> Extractor      // First consult the Extractor
[Router-1 in /Input] locate           // Locate Extractor
```

The Router has to conclude there is no Extractor Agent to locate though.

```
[Router-1 in /Input] No agents!       // None available!
```

Since there is no Extractor Agent currently available, the Router cannot locate one. When no Extractor Agent is located, the Router can still deliver the message. This can be found as an alternative scenario as described in i.e. [14.1.2](#).

```
[Router-1 in /Input] -> Profile       // Acquire situation
[Router-1 in /Input] locate           // Locate Profile
[Router-1 in /Input] move             // Move to Profile's location
[Router-1 in /User1] locate           // Check it is here
[Router-1 in /User1] post             // Ask Profile for situation
```

The Router Agent now waits for the Profile to make its decision. Profile Agent will have to make a decision for this message. Since the available (meta-)information for the message indicates there is no reason to decide this is an important message, it marks it as currently not applicable.

```
[RuleProfiler in /User1] situation    // Note currently not important
[Router-1 in /User1] wait             // Wait for decision
```

Since the Profile Agent has assigned a situation that does currently not apply, the Router Agent receives a failure. After a failure from the Profile Agent, the Router's task model instructs it to store the message for now. The Router Agent will thus try to find a Storage Agent. After this it posts the message to this Storage Agent.

```
[Router-1 in /User1] -> Storage      // Acquire Storage
[Router-1 in /User1] locate         // Locate here
[Router-1 in /User1] post          // Post message to Storage
```

The Storage Agent receives the message, and discovers it has to be stored. It thus modifies the database, and confirms to the Router Agent it has handled the message.

```
[SQLStorage in /User1] storing      // Message is stored
[Router-1 in /User1] wait           // Await confirmation
```

Since the Router Agent receives a confirming answer, it concludes its job is done.

```
[Router-1 in /User1] <- Storage     // Done storing
[Router-1 in /User1] job done       // Message fully processed; done
```

15.12 Deviations

Although the prototype is implemented according to the design, a few differences cannot be avoided. For practical reasons not all elements are completely implemented. A summary of the deviations from the original design is given in this section.

Agent communication To reduce the complexity of the implementation, the communication between the agents has been simplified. Instead of a specific type of request per service (as described in 12.4), a generic request is used for all services (see 15.3). The actual work to be done is given in the message itself. Gain is the reduced time for implementing and testing the agent communication. Further implications hereof are discussed in chapter 19.

Simulation Although the design incorporates certain devices, no specific devices are used. Instead of using the actual devices, the modalities are simulated. This reduces the dependency on a particular device or operator. Standard services (e.g. e-mail) are available through public protocols, and are implemented. Since the modalities are used, the impact for at least the tests are considered negligible.

Situations Another deviation in the prototype is considered a minor issue left for further implementation. Although the design defines the situations to be defined by the user, the prototype used fixed situations. These are hard-coded in two of the agents, but are treated as string-labels. Left is the implementation of an (part of the) user interface to manage these situations. The introduction of new situations should be tested for the profile as well of course.

Part V

Evaluation

After the implementation of the prototype, an evaluation was made. Before this was made, a series of functional tests were conducted. These are described in chapter 16. Chapter 17 deals with the interpretation of the result of these tests. It continues towards an evaluation of the prototype and the design as well.

Chapter 16

Functional Tests

This describes some functional tests of the prototype and design concepts. Since the prototype is a (partially) representative implementation of the design, it is used to test the design concepts. A qualitative evaluation of the most important aspects of the design is the aim of a series of tests. It is emphasised these are tests for functionality, not the technical tests. The prototype has been build as proof-of-concept, which should be established by these tests.

First of all, an overview of what these test try to establish is given. After that, the used approach and a description of the actual experiments are given. For an interpretation and discussion continue to page [177](#), to the chapter that contains the evaluation.

16.1 Objectives

This section will state which points are attempted to be proven. These issues are listed and explained below¹. They are derived from the goals and the requirements as they were delimited in chapters [4](#) and [9](#) respectively. Later sections will give the approach and the experiments themselves, a discussion follows in the next chapter.

- Extensibility
- Adaptability
- Personalisation and common services

¹One shall note that items like scalability and security are not present, as they were considered out of scope for this project.

- Robustness
- Cross-media profiling
- Communication overload

Extensibility The first element to be tested for is extensibility. Dynamic extensibility to increase functionality by adding devices to the system is indicated here. In other words, one could add new features by adding an agent with these new features to the system to be improved.

The additional gained features can be on all aspects of the system for various purposes. Note that with the extension, the original functionality has to be kept intact.

Adaptability Replacing or adjusting existing functionality is another possible gain. Agents can easily replace each other, by launching a new one and retracting the other. This allows for easy modification of required functionality, while other parts of the system should not be influenced. Although this is close to extending the system, there can be a major difference as well. When the replacement agent has completely different behaviour, this could have different results. The end result of the entire system can thus easily be adapted to changing needs.

Personalisation Since each person has its own way to deal with his or her communications, personalisation is an important requirement. Therefore, distinguishing between the addressee (receiver of the message) and handling accordingly, is required as well. Although personalising services is a key feature, this does not imply all services should be personal. Specially for the Transformer Agent, it has significant benefits to provide this as a common, shared service for all users. In this case the service can be improved by one maintainer, providing more capabilities for all users. The result is a requirement for partial personalised functionality. Thus the capabilities of the design for this functionality should be evaluated.

Robustness Another subject that can certainly be an issue is robustness. Due to the many degrees of freedom encountered in communication, one cannot rely on all messages to conform to strict standards. This requires a certain robustness when dealing with messages that cause unforeseen effects. Another important reason to require robustness is the lack of reliability in most networked systems. Since all systems are assumed to be connected to the network, unavailability of parts of the network must be accounted for.

Both of the above reasons require certain capabilities to recover in case one or more steps of the process fail. What has to be shown is the ability to recover in case of failures. This implies as well that failing for one message should not result in failures for other messages afterwards. Recovery does not require to fully gain the same targets, but allows for the usage of alternative scenarios.

Cross-media profiling Enabling profiling across multiple media is another high-level requirement. Since unified messaging is not bound to a single medium, it should be possible to establish a decision independent of the used media. This allows a message of one kind to be judged based on the similarity with a message of another type. Another benefit would closely relate to the extensibility. After addition of a new input device (of a new medium-type), messages from this device can be judged immediately. This should result in reaching reasonable assignment of importance to a message directly from the introduction of the new medium. A similar extension would be the addition of (newly available) extractors. This might provide additional meta-information, that was previously only available for another format, now enables the system to cross-reference both media. Evaluation should show whether the design can support this property.

Communication overload Last, but certainly not least, the system was designed to help an user overcome a communication overload. Due to the fact that this may be subjective when measured, it is certainly more complicated to test. Since a field test with a sufficiently large number of users is beyond proof-of-concept, such a test will not be performed. Preventing communication overload is a major issue however, and is not completely left unevaluated. One or more tests have to be performed to prove the functionalities, that could help prevent an overload, are present. The elements that are considered to be capable of helping to prevent such an overload are message queueing and profile learning. These are considered to be adequate enough, since they can achieve what was stated previously in the description of the problem. There it was stated that only receiving those messages that are important to the user have to be shown at this moment, by tracking the user's interests. Queueing messages for situations that are currently not relevant can solve the former. The latter can be resolved by learning the user's profile.

16.2 Approach

A description of the approach for the tests is given in this part, along with a motivation. All tests should be aimed to test for a specific target property, an item from the previous stated list. The requirement for a test is thus to be distinct between the presence and absence of certain behaviour. A brief consideration of combinations of several properties and tests follows last.

Extensibility The point to prove is that the system can be dynamically extended. As a result, the test should be performed with an uninterruptedly running system. The second requirement for the test follows from extensibility.

A new device should be added to the system. This device should be usable by only creating a new corresponding agent. For the rest of the process, all existing agents should be used. The functionalities that were already present will thereby be kept.

This test will prove successful that the design is extensible, when the newly added functionality is added transparently. In other words, the functionality should be added without altering other parts of the system. The other condition for success is that the added functionality behaves as expected. When both conditions are met, the extensibility can be considered proven.

Adaptability Adjusting functionality seems very similar to an extension as discussed above. One thing that is very different though, is adjusting behaviour. When one adjusts behaviour, one adds an agent that is not similar to existing agents. A test should thus show how using an agent with different behaviour influences the system.

The approach that results is the following. Two systems are run next to each other². The only difference between both systems should be a single agent that behaves differently. Resulting effects should be observed for evaluation, compared against the intended change in behaviour. This could be repeated for several different behaviours and agents.

Evaluation can conclude adaptability when a partial functionality is positively replaced. This is achieved when the new functionality does not affect other, unadapted, functionalities. The adapted functionality has to behave correctly as intended of course.

²Although this can of course be performed sequentially.

Personalisation Testing for personalised and generic services should be quite simple. Once a multi-user environment is set up, just sent similar messages to different users. Make sure you can keep good track of which agents are used. Furthermore, ensure the users behave different (e.g. with different interests) to the same message. Now one should be able to observe whether similar messages are dealt with in different ways for different addressees.

On evaluation one should observe the created track records. Each user should receive his or her personal messages, in a personal manner. Personal agents only have to be used for messages addressed to the owner of the agent. Generic agents may be addressed for use with anyone's message. The track has to show whether the proper agents were addressed in each separate step. Note that unexpected behaviour may have many other causes as well.

Cross-media profiling This aspect probably requires some more work. One of the things that is considered present here, is profiling. This should first be tested itself as well, of course. As a minor simplification of this test, cross-media profiling could be tested by using an explicit way of profiling. This can be achieved by using profiling without self-learning capabilities. Discarding learning capabilities assumes that a (static) profile is used, that is of a quality that can be achieved by learning.

Profiling across multiple media requires the ability to receive multiple types of media with the system. Furthermore, one should make sure that a profile is build (i.e. learned or created) against different media than those for which it is tested. This to make sure that the profile is clearly distinctive on characteristics of the message, instead of the used medium.

The approach used is using several media in the system, with one medium without extractors. The next step here is using new extractors for that one medium, resulting in meta-information that was previously only available for the other media. After addition of the new Extractor, that one medium has new meta-information that was available for the other media before. Using messages from this medium now should result in other (more accurate) assignment of importance, based on the profile gathered from the other media.

One can use white-box testing. This method may require a minor modification to the implementation, to have explicit access to certain information. With each decision, one should know the available meta-information, the decision made and the reason why this decision was made. By comparing these, one can check whether the decision was really made based on knowledge from another medium.

Robustness Recovering from failures implies the presence of errors within the system. A test for robustness should thus be run with (simulated) failing parts of the system. Only failing parts are considered, since an entirely failing system is hard to work with at all.

The method for testing recovering capabilities is quite clear. First run a system in a normal functioning way with some messages. Then run the system in the same way, while making parts of the system deliberately fail. Repeat the above for different parts and other ways of failing.

Judging the result is a little harder than the test itself though. Some of the recovery actions are deliberately set in advance. These can simply be compared to the intended actions. More complicated failures are less likely to occur, but might result in less obvious behaviour. Objective evaluation of more complicated errors throughout the system should decide how much failures the system can take before falling back to undesirable behaviour.

Communication overload As stated in the previous section, a full test for the system whether it prevents a communication overload is beyond the scope of the project. However, some features that can enable this can be shown to be present, identified as queueing and learning. Together they are likely up to the task of helping the user (as discussed before). Both will be tested separately as well, since there is no direct dependency between them.

Queueing The first is relatively easy to test when using a profiling system that is controllable. Sending a series of distinctive messages, one should simply change the applicable situation at the user's end once in a while and observe the results. Messages are explicitly required to be distinctive, since only messages that all apply to the same situation do not make sense. This can be repeated for different patterns of interest, thus by using different profiles.

Learning The latter is harder to test. A realistic performance (of decision) test requires a large set of real messages. Since this is not available, such a test is hard to perform. This makes a quantitative analysis rather impossible. A qualitative analysis is thus restricted to determining the presence or absence of learning capabilities. This can be achieved by sending a large set of reasonable messages through the system, while acting as a (mostly) consequent user. By tracking the decisions made by the system, and comparing them by the expected ones, an evaluation can be made.

Mixed tests All of the above tests aim to prove one particular property. As with all more complex systems the interaction of all parts can have its own effects. Therefore, tests should be made that combine several of the above. Most of the above should be able to combine. Several of these combinations will be tested.

Since many combinations are possible, no further combinations shall be chosen here. Note however, that some of the tests above are already closely related. For instance the testing for cross-media capabilities (implicitly) uses an extension to test for improved profiling.

16.3 Experiments

The tests described in the previous section have been conducted. Here a description of each of these tests is given. The tests themselves were conducted on standard personal computers connected by a network, with the developed prototype. Actual realisation of the tests tried to follow the approach defined in the previous section as strict as possible. Below is a summarised description of each experiment.

A remark has to be made regarding some terminology used in these descriptions. Since the prototype is implemented with the Tryllian ADK, some specific terms are used. A **habitat** is a platform that enables running agents. These **habitats** can be connected together. Within this **habitat**, one or more **rooms** can exist. A **room** is a logical place where agents can be located.

16.3.1 Extensibility

The extensibility test aims to show new devices can be added to the system dynamically. This was tested by addition of three devices to an existing running system. For this purpose, two additional computers were used to host the added agents.

Extensions Three ways of extending the system were tested. They were added to the system, by starting a new habitat on another machine, with the extension running in this new habitat.

1. SMS Factory Agent.
2. Phone UI Agent.
3. RSS Factory Agent.

Scenario Below is the scenario that was followed during the test. After each step, a couple of messages were sent from all sources to test they were functioning.

1. Start the system.
2. Add the SMS Factory Agent.
3. Add Phone UI Agent.
4. Activate new Phone UI Agent.
5. The SMS Factory Agent is stopped.
6. and the RSS Factory Agent is started.

Observations A brief summarisation of the observations, made from a user's point of view during the test run. All messages arrived at the most recent registered device. As an overall observation, one can say that new input and user interaction devices worked properly after addition. Messages from a new input device always appeared in the desired format of audio (speech).

The difference between the path of the original and the message originating from an extension was the first step it takes after its was started. Since this first step uses the Extractor Agent, which was located on the main machine, the messages from the extension made a move between both machines after the locate was completed. After this move, it travelled the same way as a message from an input device directly connected to the main machine.

The notification from the UI Agent to the Profile Agent moved between machines as well. The new messages that arrived after the additional UI Agent was registered, varied a little more. Now two movements were made, the first between transformation and display to the user, the second between display and storage.

The last result from the log-analysis was two-folded and involves the combination of multiple extensions. First of all, the path of messages with different origins or destinations (UI Agent) were only influenced by that factor. Thus, the addition of an agent does not alter existing scenarios. Secondly, one can combine the effects of an input and output (UI) device. The total effect of these additions was the same as the combination of the differences from the individual extensions.

16.3.2 Adaptability

The test for adaptability of the system was performed next. A complete system, with one of all the types of agents present was tested. Only for the UI Agent, both the Swing UI Agent and the Phone UI Agent were deployed.

Adjustments The following adjustments to the system were tested:

- Router: visit Storage with original message before any other operation.
- Router: skip the Extractor.
- Rule Profile Agent: Ignore feedback, do not use it any longer to provide statistics.
- Phone UI Agent: Automatically accept incoming calls (after registration), do not ring or ask to hear first.
- SQL Storage Agent: Do not store saved messages at all, only support the queueing system. Note that this might cause some overlap for the Router Agent with the robustness test.

Scenario The following activities have been deployed for this test. After each start of a (modified) system, some SMS and e-mail messages were fed to the system. Between some of these messages the user switched situations between both UI Agents. As a remark can be mentioned that the user *did* store all messages after receipt, and as feedback (when applicable) *did* confirm the interest.

1. The initial system is activated.
2. The first alternative is started, with the first modified agent, the Router Agent needs to visit the Storage Agent before any other operation.
3. The second edition is started, with the second altered Router Agent, the one that skips the Extractor phase.
4. Next, the third modified system with the other Rule Profile Agent is launched, ignoring all feedback.
5. A Phone UI Agent is the next variant, directly accepting all messages.
6. As last modified system, the SQL Storage Agent has been modified to drop all stored message, except those being queued.

Observations The run of the system with the original setup was as expected, as can be derived from the design and is similar to that in the description of the prototype. When the Router was adjusted to skip the phase of acquiring meta-data (at the Extractor Agent), there were no differences in the observations. After alteration of the Router to immediately store the messages, no differences were noticed either.

With an adjusted Rule Profile Agent the first differences in observations were noted. After giving feedback to the system with the Swing Interface, there was no reaction in the Rule Profile Agent.

Other differences were observed with the Phone UI Agent that accepts messages directly. After the phone was registered, the SMS send was read to the user. The question whether to listen to the message was not asked, the one asking to store the message was.

As a last test, the SQL Storage Agent was modified to ignore all requests except those directly related to queueing. The only remarkable thing happened on request of a list of messages, which contained only a message that was queued.

Although the user's observations did not always showed differences, the logs of the internals did, and vice versa. The first modified Router Agent's first action was to acquire an assignment for the applicable piles. The phase of acquiring meta-data on the message was completely skipped.

The second altered Router Agent's first move of the Router Agent was towards the Storage Agent. As effect of this, the Storage Agent stored the original before any other operation was performed on the message. This original was eventually lost when the user stored the transformed message.

Another modification was the handling of incoming feedback by the Rule Profile Agent. It received and acknowledged the message, but did not take any further action. The modified Phone UI Agent did not show any differences in the logs.

The last test showed the most (significant) internal differences. Since the database now stored only those messages that were queued, the first differences can be found there. All messages for saving, except the one being queued, were discarded by the adapted Storage Agent. The other remarkable effect was the interaction of the Storage Agent and the Router Agent. Since the Storage Agent responds negative to messages other than those to be queued, the Router Agent tried to find another Storage Agent³. Because it could not find one, it decided to continue its alternative task. Due to the current implementation, the request for a list of messages never returns to the UI Agent. The same applies after the Router Agent visits the Storage

³This is actual a part of the robustness behaviour of the Router Agent.

Agent for its last phase, but this has no further effect, the alternative ends as well.

16.3.3 Personalisation

For the personalisation test, three users were used. Each of these users had its own set of agents. All the users had an E-mail Factory Agent and a Swing UI Agent, both of these personal. The other agents are listed below.

Public Agents Some agents were run as a public available agent:

- Linux Transformer Agent.
- Two SMS Factory Agent, using different numbers.
- A SQL Storage Agent, shared between all users, but making a clear distinction between the messages of each user.

User 1 Agents The following agents were used for user 1, in a room named /User1:

- Extractor Agent, with applicable extractions for each message.
- Rule Profile Agent, with a broad rule-set.

User 2 Agents User 2 had the following agents in /User2.

- Extractor Agent, with one particular extraction.
- Rule Profile Agent, judging every message the same.

User 3 Agents /User3 contained the third user's agents.

- Extractor Agent, without applicable extractions.
- Random Profile Agent, with a chance of 0.5 messages are important right now (only for the current situation), in any available format.

Scenario The scenario can be described rather short. All three users were sent the same messages⁴. Between several of these messages, all users switched between situations sometimes. Each message was send to each user before a next message was send.

⁴Aside from the addressing.

Observations From a user’s point of view, messages appeared differently per user. Some of the messages did not arrive directly for all the users, but were being queued. After a change of the user’s situation, different messages showed up as being important. The other difference was found in the way the messages were displayed. Some of the messages arrived in their original format for an user, while an other user received those in another format. This varied not only per user, but for the first user this varied per message as well. The last user always received the message in its original format.

After checking the logs, the following items were noticed. Although most agents used were equal in code and all ran in the same habitat, they responded differently per user. Their configuration was varied per user. Not all users have to use the same variant of a type of agent (on a code-level). This follows from the Profile Agent in the test, where the third user had an entirely different Profile Agent. Although agents can be personal, shared agents can be used as well. Analysis of the logs shows that the Transformer Agent was shared by all users.

16.3.4 Cross-media profiling

Testing for cross-media profiling capabilities was performed next. For this purpose, three different media were used. With two of these media, a profile was developed based on available extractions. The third medium was used without extractions at first, with the same extractions as the other media becoming available during the test. The used media and the related agents were:

- Fax Factory Agent, emulated by sending only images to an e-mail account, using a slightly adapted E-mail Factory Agent.
- SMS Factory Agent.
- E-mail Factory Agent.

Extractions used The following extractions were tested. During the initialisation of the test, the extraction was avoided for the “extended medium”. Both extractions were added separately, as well as combined.

- From Lookup, translating machine- and protocol-addresses to real-life names.
- Keyword Extraction (faked by asking operator), adding relevant keywords to the message.

Scenario The following scenario was applied. The medium used to be extended with extraction was e-mail. As test set different SMS, e-mail and fax messages from different senders and with different content were used.

1. Start initial system.
2. Send messages from test set.
3. Start system with extra extraction functionality.
4. Send same set of messages again.
5. Repeat last two steps for both the other additional extraction and their combination.

Observations All observations were made by monitoring the logs and the database. From these was observed that all decisions by the profile were made based on two headers. These were the normalised “from” and “keywords” headers found with messages.

When no extraction for e-mail was available, no specific decision could be made for the e-mail messages. As result, all e-mails were received in the default situation *home*. On the other hand, the fax- and SMS-messages arrived in a more distinct manner.

After introduction of the from-lookup for e-mail, messages from e-mail were no longer all assigned to *home*. A separation of the messages in two groups was observed. This determination was, predictably, based on the sender of the message.

A similar observation was made when an extractor with keywords extraction was used. With this extractor, the separation was directly made based on the added keyword-information.

When using extractor tasks with both functions for e-mail as well, even more distinctive behaviour appeared. The result was seen in assignments to the e-mail messages that were equal to those for the fax- and SMS-messages. In other words, each combination of sender and keywords had its own assigned situations, whether it was a fax, SMS or e-mail.

16.3.5 Robustness

In order to test how robust a system is, things have to go wrong. Two systems, both with a full set of agents were used. These systems were dynamically connected over a network. The used tasks of the Router Agent were equipped with a sufficient long interval per phase before a timeout occurred. A set of possible failures that can affect the system have been tested:

- Disabling (remove) an instance of a type of Service Agent, with a redundant agent of the same type available.
- Disabling (remove) all Service Agents of an entire type, thus disabling that type completely.
- Killing `festival` while running Transformer Agent with text-to-speech.
- Receiving a corrupt message (PNG-image said to be `postscript`).

Scenario The reaction of the user was to store the message and confirm the decision.

1. Both systems were started, running all the mentioned agents.
2. The user initiates the Swing UI Agent for the situation *home*.
3. A plain text message is send through the system.
4. A second plain text message is send through the system.
5. The previous 4 steps are repeated for all the agents that have to be disabled, with one of those agents disabled at a time.
6. The same steps were repeated for each case where all agents of a certain type have to be disabled.
7. Again, but this time all agents are present and the used Transformer Agent is disturbed during the transformation of the first message.
8. The last cycle is slightly different: the entire system is started, but a corrupt message is send first. This is an image send to the system as being a `postscript`-document. The second message was a regular one.

Observations Some observations were made during the test run. A combination of user observations and logs was used.

- Disabling a single agent proved hard to test at all. To know the effect of a disabled agent, one should make sure it is used. Due to the distributed and redundant nature of the entire system, this could not be achieved without modification of the code⁵. Since the Service Agents are dynamically located during message processing, no fixed path was

⁵The prototype is not optimised to use the “nearest” agent. The local agent is thus not always chosen.

used. This mainly depends on network latency, and is thus hard to control on the user-application level.

Another important implication was found in the nature of a few agents. Some of these agents are build to run as the only agent of a type for each user. Therefore, duplicated agents in the same network caused even more unpredictable behaviour. This did not disturb the test however, since both the path and decision taken could be followed.

For all agents of which an instance was disabled, the dynamic locating mechanism helped the Router Agent to consult the other available one. From the user's point of view, only the user interface windows of the removed agent were missing. On the other hand, operations continued fine, although less possible instances of each type of agent were found.

- Disabling an entire type was testable of course. As predicted in the design of the system, alternative behaviour took over in case of a failing part. This behaviour varied per type of agent:

No Extractor Agent The message appeared without any observable difference. Alternative behaviour guided the Router Agent directly to the Profile Agent.

No Profile Agent This showed a clear difference; the messages were stored, after a couple of attempts to find the Profile Agent. This is the default behaviour of the prototype when the profile (for pile acquisition) cannot be found. Implied hereby, is that all messages become *unimportant per default*.

No Transformer Agent As result of a missing Transformer Agent, messages appeared in their original format. After a few attempts the message was delivered to the user, although it was in the original, instead of the preferred, format. Note that the Profile Agent was consulted again, so another format could have been chosen.

No Storage Agent When no Storage Agent was available, the Router Agent took its next step. In the prototype this is the end of the process, thus the message was discarded.

- Disrupting an agent during a process. The test showed that a failing task in an agent does not have to be fatal, for neither the message nor the agent. For the message itself, the Router Agent attempted to use another agent of the same type. The agent that encountered problems, recovered automatically after the failing task terminated.

- Last, but not least, malicious messages had only temporary and local effects as well. A message that caused a malfunction in the system, only affected itself. The agents could recover like above, but the corrupt message was received in its problematic format.

16.3.6 Queueing

As important aspect in the design to help the user overcome a communication overload, messages are put on virtual piles. Each messages is thus *queued* on zero or more of these piles. This queueing was tested with three situations for an user, although these are just user-defined labels:

1. home
2. work
3. travel

Messages The messages used for the test form a selected set, known to be assigned to different situations. All of these were send as e-mail, with different subjects to distinct between all combinations of the situations. The content of the messages was unique enough to distinguish between all different messages. A Rule Profile Agent was used to ensure the proper applicable situations were assigned.

Scenario The set of messages was send to the user with clear delays between the messages. Between various of these messages, the user altered its current situation. A list of messages in the storage was requested as well during the test. The user responded to messages simply by deleting them and confirming the assignment made.

Observations For the sake of readability, the observations will be summarised. Although one could summarise the behaviour as: “Some messages appeared immediately, while others only after a situation change”, a little more expressive summary will be given:

- Messages known to apply solely to the current situation, appeared directly to the user.
- If another single situation, that was not the currently active one, applied, messages appeared after that situation was registered.

- When multiple situations applied to a new message, these messages arrived:
 1. Directly, when the current situation was included in these situations.
 2. After a change of situation to any of the applicable situations.
 3. Otherwise when explicitly requested by the user.
- Messages that arrived after messages that were queued, could be received immediately. Messages in a queue did not hold up any other messages for the current situation or in other queues.
- Messages assigned to multiple queues appeared to the user only as a new message in one situation. Only the (chronologically) first applicable situation was used to show a queued message. The message was automatically removed from other queues of new messages in this case.

16.3.7 Learning

A k -nearest-neighbour Profile Agent has been used to test the capabilities of the system to learn an user profile. For this purpose, a new UI Agent was created, to automate the reaction of the user. A similar, but even slighter, change was made for the assignment of keywords. No particular other modifications were made.

Target values Four different situations were defined for this user. The following situations were applied:

H Home

W Work

T Travel

V Visit

The user replied to these messages as stated in Table 16.1. Since there were four times eight combinations, thirty-two possibilities exist. These have been divided as shown in the table, which can be handled with 12 rules in the Rule Profile Agent. The automated UI Agent has been constructed based on such rules.

Table 16.1: User's reaction to classes of messages

	Sender A			Sender B			Sender C			Sender D		
Trip	H	W	T	H	W	T	H	W	T	H	W	T
Research		W			W			W			W	
Code			V		W	V		W	V		W	V
Sport	H	W		H	W				T V	H		
Free	H	W		H	W				T V	H		T
Beer	H	W		H	W		H	W	V	H		V
Games	H	W		H	W				T V	H		
Weekend	H	W		H					T V	H	W	T

Additional tests The test above was based on the assumption users behave always consequent. Since this is not entirely realistic, two variations of the previous test have been run. In these tests, the automated UI Agent has been slightly modified. To emulate the inconsistent behaviour users can have, the feedback given by the UI Agent was changed based on chance. With a certain chance, the feedback did not comply with the actual preferred situation. In these cases another arbitrarily set of situations was send as feedback. This variant was run with both a 2.5% and a 5.0% chance of deviation. See the results and discussion for further elaboration on this test.

Scenario A message was send after a fixed interval of fifteen seconds. This message confirmed to one of the above categories, and the user responded as stated in Table 16.1. The response of the user was automated, through the specialised UI Agent. All messages were automatically send, based on a small script, generating a message from a random sender on a random subject. The used random function is considered to have an uniform distribution.

Observations Since the test was performed in an automated way, no user observations can be given. The results are therefore analysed for the assignments made. See Figure 16.1 for a visualisation of the cumulative wrongly assigned situations, drawn against the number of messages send. Since this is the total error over time, the fact this line levels out is a positive signal. After 200 messages nearly all of the assignments were made correct. In other words one could say the long term results in proper assignments of situations to messages. This test is not to be mistaken for a quantitative evaluation, although the displayed results include some numbers.

Figure 16.2 shows the same pattern. This is the number of wrongly assigned situations per message. Over time both the density and the size of

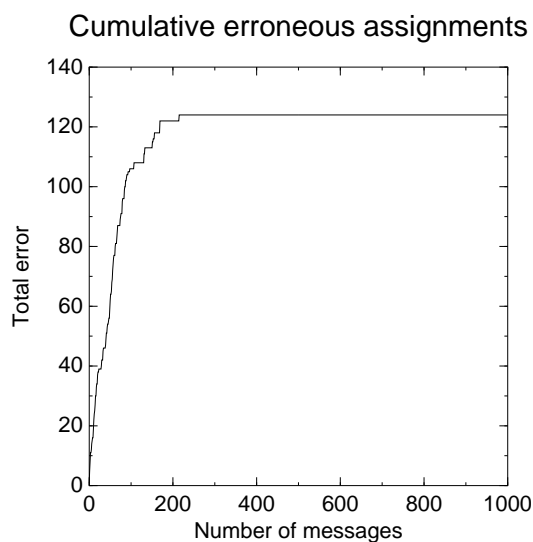


Figure 16.1: Profile learning: Cumulative error

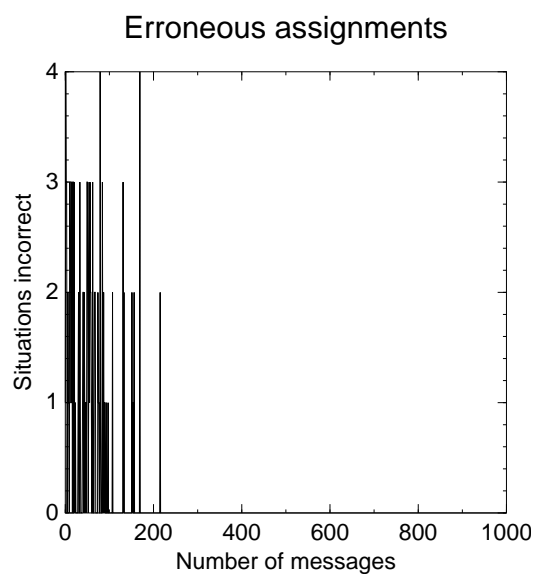


Figure 16.2: Profile learning: Errors per messages

these errors decrease. This says more messages are assigned correctly as well.

These errors are split apart in the remaining figures. The first one (Figure 16.3) represents the situations that were assigned when not needed. Figure 16.4 shows the situations that were *not* assigned (missing) for a message.

This shows that the used agent tends to send messages sooner than holding them back when less data is available. One can observe this from these

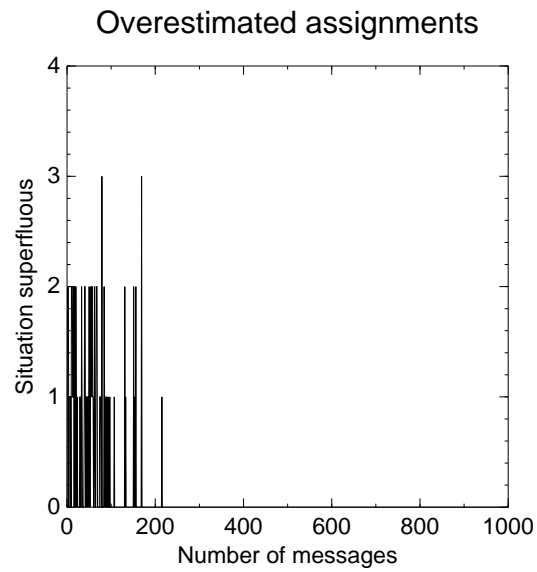


Figure 16.3: Profile learning: Extra assigned situations

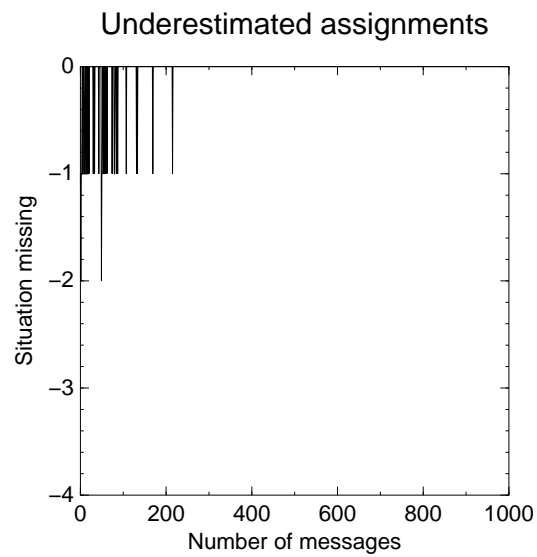


Figure 16.4: Profile learning: Missing situations

last two figures, where more weight is present in the first graph. Inferred hereof is that a larger part of the errors occur by assigning situations that do *not* apply, rather than lacking the assignment of a situation that *is* applicable.

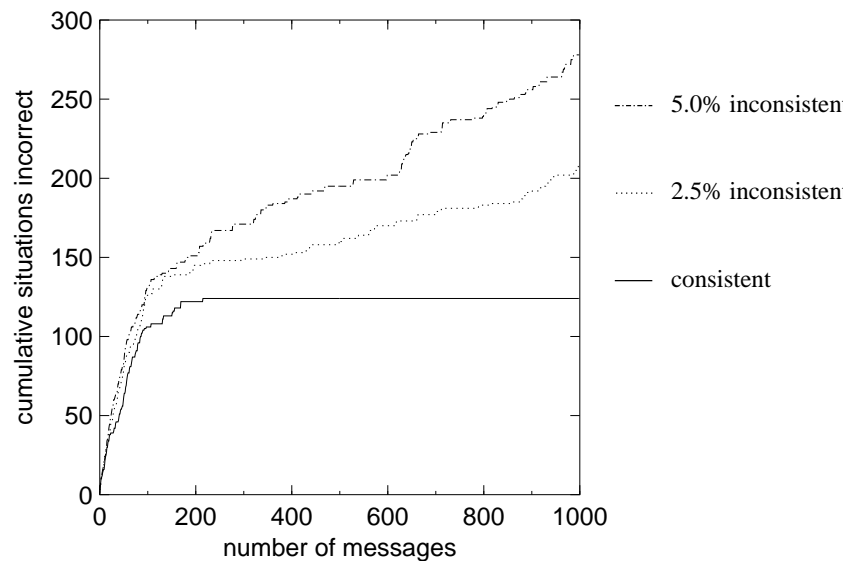


Figure 16.5: Profile learning: Various user inconsistencies compared

Additional test Two derived, randomly disturbed, tests were run as well. With these tests an approximation of inconsistent behaviour on the user’s behalf was made. The results are shown for a 2.5% and 5.0% chance of inconsistency in Figure 16.5. When compared, the last chance shows the steepest line. This means that with more inconsistent behaviour, more errors are shown, thus less accuracy is achieved.

16.3.8 Mixed

All of the previous tests aim for a single property. Many combinations of the above tests can be made, although some would have little added value. Some of the tests as described already have some mixed elements, although they emphasise a specific property. Some additional tests were performed, and will be described below, although less formal and elaborative as those above.

The experiments were mainly a combination of adaptability and personalisation. First thing tried was to use a variant of the Router Agent per originating location. As described in the adaptability test, some modified versions of the Router Agent were build, with a different routing schedule as result. Two different machines were used for hosting a Factory Agent. Each of these machines had another variant of the Router Agent stored as original code-file (called the DNA of the agent). A new message is routed based on a Router Agent created with this file. The result is a different routing process

per originating location of the message.

Another variant required a minor additional modification. Now the Router Agent was not varied per location, but per user. In principle, a Factory can send one message to several users. A practical example is the RSS Factory Agent (news-headlines), which can send the same new message to a list of users. A slight modification was made to this agent, to support a new feature. Instead of sending the message with a standard Router, a Router is selected based on the addressee. This can be done, since the used file to create the Router Agent is determined during runtime. For each new message, a different Router Agent can thus be created. The result hereof is that the addressee determines the effective Router Agent. Of course other criteria can be used to select the applicable Router as well.

Chapter 17

Interpretation

This is a discussion about the test- and evaluation itself, but ends with a discussion of the prototype and the entire design. One should keep in mind that these are not te final conclusions yet. Please refer to part **VI** for further discussion of these items.

17.1 Tests

First, a brief discussion on the strengths and weaknesses of the tests as they were performed. Seven tests were run, all emphasising on a particular intended property. Each of these tests was executed under clear delimited circumstances. On the user's side idealistic behaviour was incorporated. It is acknowledged this is not realistic for a (commercial) deployable system.

Nonetheless, the prototype and tests were meant to be proof-of-concept. For this purpose they are considered to be sufficient. Although the tests are specialised towards a single point of interest, all these points combined have to accomplish the desired functionality. All tests did highlight a particular property, but involved (nearly) all parts of the system. These tests thus show the properties can be achieved within a complete system, not only a specific part.

A brief discussion per test follows below. The tests for the mixed properties are not discussed, since they were conducted in a less formal and elaborate manner. First of all a list of what has been achieved, corresponding to their related tests:

- Extensibility.
- Adaptability.

- Personalisation.
- Cross-media.
- Robustness.
- Queueing.
- Learning.

17.1.1 Extensibility

As the test shows, the system can be dynamically extended. New devices for both input and interaction with the user (which cover output as well) have successfully been added. These could integrate with the existing parts of the system, without any problems. Messages created on the new input devices were automatically routed through existing systems. The new device with an appropriate User Interaction Agent was able to provide the user with all of its available communication transparently. Messages received in other *habitats* of the system were automatically delivered at the new device. Logically, this only happened after it was registered, since the most recent activated UI Agent is used.

The only requirement for such an addition is the locating mechanism used. Although this is accomplished with simulated DNS, the implementation of standard DNS is merely a coding-exercise rather than a challenge of skills in design. An internal equivalent of this DNS capability would be more appropriate with respect to a large-scale deployment¹. The newly added devices can be utilised without further requirements. If the Factory and UI Agent are running, the Router Agent resolves all other connectivity.

The dynamic extensibility is partially a gain of the usage of agents. Due to the clear separation of functionality, a new device can be added to the system transparently. No particular knowledge of the device is needed in the system, the agent shields it off. Furthermore, the new device only needs an agent that packages the messages. All knowledge of the network and processing the message is present in the Router Agents, relieving the Factory Agent hereof.

¹ Specifically when security and deployment become involved. Running a secure, fully-featured DNS-server is not a trivial task for all users.

17.1.2 Adaptability

Although only a few modifications were tested, these do show quite a few things. By the use of agents, easy partial modification of the system becomes possible. Replacing a single agent can modify a part of the system, without affecting the remaining functionality. Such a replacement can be two-folded, for both the result noticed by the user as well as the internals of the system. It is thus possible to adjust the interface for the user, without further alterations. On the other hand it is possible to adjust the processing of a message without changing the interaction with the user.

The second conclusion is more related to the handling of messages. The Router Agent can be altered to follow a different pattern. This can result in extra or less operations on the message, thus changing the global process for a message. Of course there are limitations to the magnitude of the modification. Some modifications (as shown by the modified storage test) have such impact that they alter the functionality of the entire system, and thus require further attention.

This is probably one of the strongest agent properties present in the system. Partially as a result of the separation of functionality, otherwise of its mobility. The former ensures each functionality is properly compartmentalised. This allows the modification of a function as long as it behaves the same towards the rest of the system. The Router Agent's mobility allows each message to have its own code. Adding code to a message provides the adaptability of processing for each individual instance.

17.1.3 Personalisation

Since people have different interests and behaviour, a one-size-fits-all solution is not realistic. Personalisation is shown to be achievable within an agent-based system. Agents can be configured differently to behave in correspondence with the user they represent. Not only the look-and-feel can be altered, complete behaviour can be changed, thus enabling customisation of the entire system. Personalisation goes beyond configuration, by allowing specific agents to be replaced, thus easily replacing parts of code. Depending on the recipient of the message, other agents can be used for performing certain tasks. This allows for different behaviour as response to a message. As an added benefit, generic services can still be kept, thus enabling certain improvements to the system for all users at once.

On the downside of this personalisation is an issue that is out of scope for this study. When each user has its own set of agents, system-requirements and scalability may become a problem. Using some generic agents might

help to reduce these requirements.

Agents play an important role in the personalisation of the system. Although other solutions, like client-server, can use different configurations for different users, agents easily enable a more powerful form of personalisation. By using another type of agent for a certain user, entirely new behaviour can be utilised. The treatment of messages can thus completely be modified per user. Since agents communicate among each other, they do not need to know any of the details of the other agent internals or working.

17.1.4 Cross-media

A small test proves it is possible to decide whether a message is important independent from the used medium. As long as similar meta-data is available for messages from different media, a classification of messages of one medium can be used for other media. First of all one has to take into consideration two important assumptions. One is the availability of similar extraction tasks for different media. Secondly, the profile needs to be able to handle this increasing amount of information. A profile is thus expected to become more distinctive where appropriate.

Acquiring such information can be handled per medium, allowing specialisation for this job. The overall benefit is the ability to judge messages based on their content rather than the used medium. Due to the separation of the extraction and the assignment phase, the latter can be achieved independent of the used medium. How the information is gained from a medium is left to the Extractor. Shown is that information previously only available for other media can be used when it becomes available for another medium. To summarise one could say it is possible to achieve a higher granularity in filtering with more meta-information, without requiring more data (messages) for the particular medium, but can be based on messages from “any” medium.

17.1.5 Robustness

To protect the system from the networked and open nature of communication systems, robustness measures have been integrated in the system. These measures are effective when an agent of the system is not available, or its operations get corrupted. Although the functionality of the system is not fully accomplished, the system does not collapse either. In contradiction, alternative scenarios can be provided beforehand. In this way the developer or user of the system can define how to handle failures. Due to the separation of process logic and actual operations, together with the independent nature of agents and tasks, errors can be dealt with rather easily.

The provided scenarios in the prototype are not all that useful, since some end up in only logging the failure. In a full-sized system better scenarios can be implemented, providing better preventive measures against data loss or corruption. Creating good scenarios for erroneous cases is required of course, but can be dealt with on a per task level. Missing or corrupted agents thus have little impact on the system as a whole.

The test itself can be improved as well. Particular mentioned here is the case when the Extractor Agent could not be used. If messages are used that will be annotated with meta-data that results in distinct assignments by the profile, the assignment of importance changes as well. This will show that a failing extraction reduces accuracy and granularity, but does not render the user unreachable.

Basis for this robustness is twofold. A function to be performed (task) may fail, but the next message is handled as a new task, thus is processed without any influence of a previous task. Another reason agents contribute to the robustness of the system is their independence. Since each agent runs as a separate program², a failure of one agent (program) thus only affects that agent. Because each message is handled by its own agent, messages are thus isolated from each other. Due to the task-based design of agents, recovery scenarios can be established for each task. Whenever a consulted agent does not continue after failure, the message is not lost but the router can time out and try again elsewhere.

17.1.6 Queueing

An important achievement in handling the communication overload is queueing messages. Depending on the assigned applicable situations of different messages, only those messages that are currently interesting are shown directly to the user. Messages arrive only when the user says (s)he is in a situation where those messages are useful. The user is thus not bothered or distracted with messages that are not interesting now. The assignment of situations to a message is not part of this test, although the success of the queueing depends on the quality of this assignment.

Messages only arrive in the user's experience when the user activates a situation where those messages apply. Message that belong in other situations are hold back until that applies. When multiple situations are assigned to a message, a message appears in the first activated situation it applies to. Due to the fact queued messages are saved in the Storage, messages from other situations can still be accessed.

²At least in concept, technical optimisation implies a few differences.

17.1.7 Learning

With what is probably the most idealised test, it is shown the designed agent-based architecture can be used to learn an user's profile. It is shown that after enough messages with (consequent) feedback are given, the profile can increasingly correctly assign applicable situations to messages. This judgement is based on feedback provided by the user in a consistent manner. When the user responds less consistent, additional tests show the accuracy to decline. The worse the quality of the feedback, the worse the assignments made as well. Providing representative and consistent feedback on the user's interests for different messages thus seems essential.

Since the implemented algorithm is a loosely founded, simplistic, unoptimised one, further study can result in higher learning accuracy. The best algorithm is currently neither given nor described, but the test with the k -nearest-neighbour algorithm shows automated profiling is achievable. Which algorithm is the best approach for this learning is not established. Neither is it shown how well this is able to handle changing interests, nor human-error and inconsequential responses in feedback. A quantitative analysis of different algorithms and situations remains necessary before those conclusions can be made.

17.2 Prototype

Although the tests were based on the prototype, they did not fully cover the prototype. From the implementation, usage and tests of the prototype several things can be learned as well. These will be described below.

Agent languages Using generalised, instead of specific, communication between the Router Agent and the Service Agents has benefits for implementation. This requires more strictness on the developer's side though, since one can easily operate beyond the scope of the required service. This has more impact on extensibility and adaptability however, as will be discussed in [19.6](#).

Locating mechanism A distributed, searchable and scalable locating system is essential for dynamic personalisation and adaptability. In order to add devices dynamically to the system, a flexible runtime locating mechanism is essential as well, in order to prevent modification and changes to many parts of the system. As the prototype shows this can be accomplished with DNS and JNDI, but a single solution would be preferable.

Automated transformations Creating automated transformations is relative simple when appropriate software is available. Ensuring quality, speed and content are optimised, requires further insight in conversion and more sophisticated planning algorithms.

User Interfaces Although no particular effort has been spend on user interfaces, a noteworthy fact showed up. Since different user interfaces are supported aside each other, a light weighted user interface can be very usable. The Phone UI Agent does not implement all the functionalities to operate on the storage. Still, a phone is usable as a device, since the storage can be accessed through another UI Agent as well. Note that these functionalities are designed to be optional in [10.3](#).

Multi-part messages From the practises of e-mail the concept of multi-part messages is known. In a multi-part message, the actual message consists of several pieces which, when combined, form the actual message itself³. These pieces can be of multiple media types, and handling these combined parts needs special attention. Either each part is to be send as a message, or these packages of parts should be handled by each agent. The former has huge implications on the idea that one communicates a certain thought within this one message. The latter would result in a more complicated implementation for all agents, since several pieces have to be handled at a single time. For instance, a transformation can fail on each of these parts, and the question rises whether the whole transformation thus failed.

Meta-data and profile Another issue encountered during the implementation is the relation between the acquired meta-data and the profile. Many machine learning algorithms need further knowledge of relations between different instances of data. Either a comparing (i.e. distance) function is needed, or normalisation of the input is needed.

Object oriented programming allows inheritance of classes. Several elementary classes can be implemented, providing methods to compare different values of a meta-data field. This relieves the learning algorithm of the specific details of the meta-data. Examples of the generic classes can be numerical, sortable, sets, labels, booleans, date and time. Specific meta-data can be marked as instances of these classes, e.g. urgency is a numerical in the range 1 to 5 and keywords are a set of labels.

³A single message on a single subject if used properly.

Distributed agent types As briefly discussed at the robustness test, certain types of agents are not designed to run multiple instances per user. For several reasons (i.e. robustness and performance) it can be desirable to run several instances though. When in future implementation this has to be supported, these agents should take their own precautions for synchronisation. This way the behaviour of the combined agents of this type will be coherent to the remainder of the system and the user.

Device data Most of the input devices, and the corresponding agents, are able to provide the system with some meta-data from the start on. One should be careful not to blindly include all of this information with the message. Two side-effects have been noticed from the prototype:

- The relevance and quality of the provided meta-data can be doubtful. Due to the open nature of most communication systems, unreliable information can slip into the system. Other information is unrelated to the actual content of the message (e.g. the `List-Unsubscribe` in case of an e-mail mailing-list).
- Excessive meta-data for a single medium can disturb the cross-media learning capabilities of the system. When a certain medium has a relative overhead of headers, comparing messages from different devices and media becomes more problematic. Due to the small share of generic headers, media-specific headers can mislead the distinction between messages from different media. Particular the message with too much meta-data can hardly be compared with messages from other media, at least with the learning profile in the prototype. Note that the use of various and heterogeneous meta-data is encouraged and necessary, one should just be careful not to include useless redundancy.

17.3 Design

Since the prototype is directly derived from the design and the tests were run against this prototype, the evaluation applies to the design as well. Apart of the results of the tests, the following can be said from the experiences gained throughout the project:

Unified messaging The proposed design is able to handle the concept of unified messaging. It is able to automatically transform between different media. New devices can be added dynamically and transparently, when the

generated messages consist of formats known by the Transformer Agent. How these devices are handled is only bounded to the corresponding agent. The limitations for the particular device are thus shifted more from the provider towards implementation and connectivity (e.g. one could have a computer call ones voice-mail service and record it, if your voice-mail operator will not cooperate).

Unified messaging was declared in chapter 6 to provide messaging using any device, anywhere and anytime. The latter is rather simple using electronic systems, that can operate around the clock. Using device anywhere is solved with the mobile Router Agent. This agent travels to the user's device to let him or her perceive a message. With a dynamic locating mechanism, this agent is capable to connect to each networked device. Although this satisfies for delivering messages anywhere, using any device implies the message needs to be adapted to this device. With the automated data transformation of the Transformer Agent, messages can be delivered in a format the user prefers and the device can handle.

Adaptable processing Furthermore, it allows for easy adaptation of the processing of such messages per user. Not only can the user receive messages in its own preferred format, the timing of each message can even be adjusted. It thereby should be able to protect the user from a communication overload.

The entire processing has been reduced to the application of certain services to a message. Creating scenarios for ones messages would (theoretically) become as simple as drawing a task-diagram of applicable services. The Tryllian Visual Agent Designer [74] (a description can be found in appendix A) could be taken as the basis for an implementation of this idea. The main requirement would then be the creation and explanation of useful tasks, to let the user define its own system.

Architecture only The design only describes an architecture to handle personalised messages. How to treat each specific medium for extraction of meta-data, or how to learn an user profile, is subject of further research. The proposed architecture does enable incorporation of new insights. In this way the system, and thus the users, can benefit from progress in a specific area without the requirement to completely redesign and rewrite software for these purposes.

Agent are facilitators Closely related to the previous issue, is the fact that agents are very good facilitators for personalised unified messaging. They do not provide any form of magic or automated intelligence, just be-

cause they are agents. Elements like machine learning must still be implemented separately.

Dynamic additions In order to be truly capable of dynamically adding (new) extractions and transformations to the system, a slight modification might be useful. In the current implementation, an entire agent has to be replaced. A better alternative would be the scenario suggested earlier (page 117). In this case one agent (per service) would be connected directly to the system. This “central” agent would only act as the planning agent. The actual extractions or transformations are performed by execution agents, that should register themselves with the planning agent. Addition (and retraction) of a capability could thus be resolved by (un)registering an execution agent.

Separation of services In the design a separation of services was made. Through the separation of each phase, several advantages have been achieved. Functionalities can be added or modified, while failing steps do not compromise the entire system. The prototype showed the separate phases have enough unity to provide the required functionality. Where modifications and personalisation were considered, no steps show reasons for further logical combinations either.

Mobile agent: semantic router Since each message is handled by its own agent, this facilitates sophisticated handling of each separate message. Using mobile agents for this concept has been recognised early. Intelligent mail handling and semantic routing are the terms used by Chess et al. [9].

This concept has been considered before, and assessments were made. Two of the conclusions of Chess et al:

Although mobile agents offer no exclusive advantage [...] One begins to see here the flexibility gained from this approach.

Mobile agents are a convenient transport mechanism for mail, but have no essential role in the attribute processing.

These findings can only be acknowledged, based on the findings of this project. The Router Agent does not perform any processing itself. It does on the other hand not only transport the message, the router can vary the

processing sequence at each phase of the process. This provides extraordinary flexibility, since each message can (theoretically⁴) be handled in its own way.

⁴Nobody will write code for each individual message in practise, only groups of messages are realistic achievable.

Part VI

Conclusions & Recommendations

A last, but not least, part of this document. First of all, a brief overview of what has been accomplished will be given in chapter 18. Next, chapter 19 will discuss these items. The final conclusions follow on page 205. After this some recommendations for future work follow in 21.

Chapter 18

Overview

In the previous parts of this document, most of the stated goals have been accomplished. This chapter will give a brief overview of what has been accomplished. It will mainly give a summary of the previously achieved. This is not intended as a conclusion and neither as a discussion.

18.1 Backgrounds

A survey of related research and developments has been conducted. The areas of interest that are covered include unified messaging, user profiling and agent technology. A brief summary of these backgrounds can be found in this section.

Unified messaging Research directly targeted at unified messaging is quite rare. Although some projects exist, they form a rather narrow basis. There are several related areas of interests that do have a lot of coverage however. Among these are speech recognition and synthesis, OCR and human expression recognition. Standards for unified messaging are restricted to SMTP and MIME, the basis of e-mail.

These standards are often encountered in practise as well. Most commercial offerings are limited to a specific set of media, and have further restrictions on their usage. Accessing all messages from any device is often not the case, receiving them in ones e-mail is common practise. Having a fax read to you by these systems is almost never an option.

Recent developments can offer possibilities for further improvements. Network technologies provide faster data communication and continuous reachability. New mobile phones and PDA's have more capabilities, thus providing a basis for more advanced systems.

User profiling Information filtering is often applied to reduce streams of information towards an user. One of the most used examples is found in e-mail. Information filtering tries to establish a profile of the user to compare new messages against. This profile is used to classify new objects, according to the user's interests. Most of the systems for e-mail are based on textual content of e-mails however.

An important technique for information filtering is creating an user profile with machine learning. Several algorithms for machine learning are briefly described. Among these are nearest-neighbour, decision trees, rule-based systems and Bayesian learning. Some other very common methods are not described since they are considered less applicable.

Agent technology Although agent technology is applied in many areas, there is now common definition of an agent. All systems based on agents do have some of a collection of common properties though. Characteristics of agents are autonomy, intelligence, cooperation, mobility, reactivity and various others. An agent can have a certain degree of agency, based on these properties.

In multi-agent environments, agents can communicate among each other. KQML and FIPA ACL are the most used languages for this purpose. As underlying architecture, a blackboard or an asynchronous messaging system is often chosen.

Agent technology has been proposed in telecommunication, aside from network management, for several goals. For the UMTS Virtual Home Environment, mobile agents are used to utilise the same software on any device. Intelligent Networks can be build with multi-agents, and IMPAX aims to support unified messaging with multi-agents. Alternatives for agents can be found in the client-server concept.

18.2 Design

After the problem was defined, a way to deal with this has been developed. This has been detailed in the design, which was presented in two parts. First the concept has been presented in part III. Chapter 14 detailed this concept on a lower level and in a more technical way. A brief summary of the concept will be given here.

Multi-Agent The proposed design consists of seven generic types of agents. Each serves a well-defined functionality. The generic idea behind the design

is a collection of agents that provide services for the processing of a message. A router per message allows individual handling through these services.

Mobile Router A mobile Router Agent helps the message past all services, controlling the actual processing. This router will do all that has to be done per message, but to prevent an enormous amount of overhead per message, it delegates all “real” work to the other agents. These Service Agents act as consultants, but the high-level decisions are taken by the Router Agent. The Router Agent effectively controls the workflow of a message.

Agents Aside from a mobile agent as router, several agents have been identified to be needed. They form the services which the Router Agent can consult. A brief overview of these:

Factory Agent A Factory Agent handles the input to the system. It shields any specific handling of input devices from the rest of system, by starting a Router Agent directly with the actual message.

Extractor Agent This class of agents makes information, implicitly present in a message and its context, explicitly available (extracts it). The resulting meta-data describes the actual message and its content at a higher level.

Profile Agent The purpose of profiling is to keep track of the user’s interests. Effectively, decisions whether a message is important or not are made by this agent, and this should provide the protection for the user against a flood of messages. Another decision made is the applied format to present the message to the user.

Transformer Agent A generic Transformer Agent takes care of the realisation of the “unified” part of the messaging. This relieves the input and output (user interface) of supporting each other’s specific formats.

User Interface Agent Handling the output device, the UI Agent allows users to perceive messages and interact with the system.

Storage Agent This last type of agent takes care of archiving messages for the user.

Remainder Within the document, further details are given. This includes descriptions of the agents, up to a model of the tasks and objects to implement. These will not be repeated here, but can be found in the related chapters.

18.3 Prototype

Based on the design a prototype has been implemented. Although this prototype is not exactly confirming to the design, the difference are little. Several agents of different types have been implemented. Some of these are fake implementations, using the same modalities but avoiding specific (hardware) implementation dependencies. Therefore the prototype is not ready for production-level usage right now, but is usable as proof-of-concept.

Implemented Agents The implementation constructed included the following agents.

- E-mail Factory Agent.
- RSS Factory Agent (Rich Site Summary: new articles published on site).
- SMS Factory Agent (faked short-text modality).
- Router Agent.
- Extractor Agent, supporting keyword-annotation, address resolving and relation determination.
- Rule-based Profile Agent.
- *K*-Nearest-Neighbour Profile Agent.
- Transformer Agent, supporting various formats.
- Swing UI Agent (desktop application based on Java's default graphical interface).
- Phone UI Agent (faking telephone and SMS modalities).
- Database Storage Agent.

Several other components are implemented, such as shared libraries. A few debugging- and testing agents have been created as well. They shall not be discussed in more detail here however.

18.4 Evaluation

With the created prototype an evaluation of the concept was made. As result of some tests, the key characteristics of the system are investigated. Based on the findings of this evaluation, one can say the system:

- Is extensible with new devices and user interfaces.
- Is adaptable per functionality to a high degree.
- Can be personalised per functionality.
- Is capable of queueing unimportant messages.
- Can recover from network-failings and malicious messages.
- Can operate across different media.
- Can facilitate generalisation (learning) of the user's preferences.
- Allows users to receive their messages anywhere, anytime, anyhow.

Aside from these findings, several other facts were found during the development. These mainly arose from the implementation and usage of the prototype as representation of the design.

- A locating mechanism is essential for full dynamic, personalised extensibility.
- Automated transformations are not hard to implement, although maintaining a minimum level of quality and semantics requires more efforts.
- Multi-part messages (e-mail with attachment) need to be accounted for early in the implementation.
- User interfaces can implement different levels of functionalities, and can be independent of underlying implementation of the functionality.
- Redundancy can be used, but certain types of agents require special measures to be taken.
- One should be careful using the additional information that comes with a message without proper selection.

A few remarks about the design as a whole can be made as well. These are only preliminary conclusions of provided capabilities. Further discussion follows in the next chapters.

- The concept of unified messaging can easily be achieved with a few types of agents.
- A mobile agent as semantic router allows for flexible, adaptive processing of each individual instance.
- Agents can provide a decent architecture for personalisation, separate for each function.
- Agents are very good facilitators, but the intelligence must still be created by the developer.
- Separating different processing-steps as services allows for easy composition of new processing.
- Agents can easily provide dynamic additional or alternative functionality.

Chapter 19

Discussion

The presented design is rather complete and its characteristics have been determined. This chapter will discuss some issues that have not yet been discussed. Note that chapters 13 and 17 have some preliminary discussion as well. The discussed issues include:

- Non-mobile router.
- Overhead.
- Scalability.
- Communication overload protection.
- Commercial realisation.
- Agent communication and negotiation.
- Other issues out of scope.

19.1 Non-mobile Router

A primary aspect of the design is the usage of a mobile Router Agent. Due to the use of a mobile agent, flexible and adaptable handling of messages becomes available. It does require an underlying platform that supports agent mobility. This might be a drawback in certain implementations for, among others, reasons of availability, performance and security. Although the last two subjects are discussed in this chapter, an alternative variant of the design is possible at a certain cost.

Instead of moving a Router Agent with the message, the message is passed between multiple agents. Each location now has a Router Agent, that exchanges a message with a Router Agent in another location where the design would prescribe a move. Instead of the agent keeping the state of the message, this state must now be past along with the message itself. In general, this would be very like regular e-mail is handled through SMTP [55]. This will be extended with some additional semantic routing, with the advantage of adaptability per location.

The main disadvantage is the loss of flexibility of handling each message in another way. The process a message is subjected to is fixed to a single scenario. When this process must be modified, all Router Agents need an update and are still restricted to a fixed kind of processing for a message. This could be solved by allowing a scenario to be present as part of the message's state. When the scripting support for this purpose becomes more sophisticated, one is effectively back again at the concept of mobile agents.

Along the same lines, one can discuss the mobility of the other agents. It may be clear the device specific agents are bound to their location. The Service Agents have no such limitations however, although the Storage may depend on a database or a physical system. They should be able to move around, as long as they update their location-record for the locating mechanism. This should allow them to find a better host, avoid system downtime or reduce network latency for the user. However, consider that specifically the Transformer Agent can be redundant, thus reducing this advantage of mobility. The agents can of course always utilise mobility to introduce new software and services at remote locations.

19.2 Overhead

The usage of mobile agents as message transporters has been suggested before. Chess et al. [9] discuss it in their assessment of mobile agent technology. Their conclusion towards using mobile agents as transport mechanism is reckoned here:

Mobile agents are a convenient transport mechanism for mail
[...]

With the additional conclusion that mobile agents have as disadvantage:

Transmission efficiency, for example a courier agent compared
to a simple SMTP mail object.

This last issue involves the overhead of using mobile code. As discussed previously in 19.1, one could use static agents with the loss of flexibility. For several reasons, the overhead can largely be reduced however:

- Different levels of code. By creating a library with common low-level standard tasks, agents can be reduced in size, still with the advantage of variation per agent (thus message). The agent can be composed of several smaller and simpler common tasks.
- Caching allows for optimisation. The platform used to implement the agents can cache code, to avoid retransmission. The Tryllian ADK [75] is optimised to cache both the code of libraries and agents.
- Distinction between code and data. Furthermore, the Tryllian ADK allows agents to implement a **state-serialisation**. This allows agents to be transported with a minimal exchange of data. Only the first time a particular implementation of an agent is moved the entire code needs to be transported. It is unlikely that each individual message will be handled by an individual implementation, therefore allowing optimisation over classes of messages.

19.3 Scalability

An issue that was placed out of scope is scalability. An important aspect that is often evaluated of architectures is its scalability however. Although agents have been said to provide scalability in general, a brief evaluation of expected scalability and bottlenecks of the proposed design is given in this section. Two potential bottlenecks are identified and discussed here:

- Locating mechanism.
- Network load.

Locating mechanism Partially based on the prototype, a main bottleneck could be the locating mechanism. Since each user has effectively its own set of agents, these can be distributed across the network at will. Each user thus has its own cluster of agents within the whole system and network. An average server or modern personal computer can support these agents for at least one user¹. With the increasing availability of continuous broadband connections and personal computer possession, the capacity is provided for,

¹Based on the prototype.

leaving the problem to locate these. As the prototype shows, standard DNS or a similar concept can solve this. DNS has proven to be quite scalable through its hierarchical and distributed nature, as it is still in use in the Internet after its enormous growth.

Network load An advantage of the use of separate, mobile Router Agents is the reduction of network traffic (page 72). This comparison was opposed a centralised agent though. When compared to, say, regular e-mail (SMTP), several additional phases of network traffic are added to facilitate the extra functionality. In 9.4, this was set aside by assuming the presence of a decent backbone. Further calculations and field test must show whether this assumption is valid enough if the approach is applied on large scale. Inefficient distribution of the agents, as discussed in the previous paragraph, will of course increase network load as well.

19.4 Communication overload

One of the problems identified at the start of the project is to prevent the user from a communication overload. Now, near the end, it is time to evaluate the proposed solution on this point. A field test with a significant number of users could do this, but the prototype is proof-of-concept. Since no field tests were conducted, a very solid assessment cannot be made.

What is shown however, is the possibility to queue messages based on the user profile. As it is also exhibited this profile can be learned, automatically reducing the amount of messages that arrive at the user. Combined with receiving messages in a preferred format, this can help to reduce the load on the user. The crucial part here is the capability to learn the user's profile, which is addressed in section 13.1 and is discussed as future work as well (see 21.1).

19.5 Commercial realisation

Within this project, only a prototype has been implemented. To be of any use to the end-user, an usable, qualitative and practical implementation is needed. This would be the case for commercial deployment of the idea. Although this is out of scope, a few considerations for the possible applicability of the system in commercial practise:

Corporate communication networks A possible area of application is as an internal system for (large) companies, or by means of an Application Service Provider (ASP). All employees of the company could be connected to the system, allowing personalisation of their communication. As advantage for the company employees are less interrupted, except for important business. The employee itself can customise its communication and is thus provided with more comfort.

Competitive service The telecommunication providers actually need the intensive usage of (mobile-) data traffic to make money, but it could be useful to accept a system as proposed. A telecom-provider that supports this kind of personalisation and bandwidth-reduction, can offer a very attractive, competitive service. The offering of a personalised, protective system might appeal to many customers. A proper and understandable user profiling method is needed however, otherwise users will be lost because they experience unexplainable, seemingly random, behaviour of their communication.

19.6 Agent communication and negotiation

An important difference between the design and the prototype is the communication between agents. Where the design included different forms of communication between the router and each service, the implementation used one common way to exchange messages. Both manners have their advantages and disadvantages, as explained here.

Specific exchanges When using specialised communication between the Router Agent and the Service Agents, each step in the process has its own *ontology*². This ontology allows to pass information specific for what is passed between the agents. This has advantages, that are commonly seen as disadvantage of a generic approach:

- Negotiation. Agents that communicate in a specialised manner about a job, can more easily negotiate about the job itself. Since a specific ontology is used, characteristics of the job can be expressed better. This allows a higher degree of adaptability and adjustment of details of the job to be done. For instance, the Router and Transformer Agents could negotiate quality of service. The resulting document size, quality, costs and response time can be negotiated.

²A systematic way to describe all concepts in some field of discourse.

- **Strictness.** Communicating only the required data increases strictness. The data and results can more easily be checked by the receiving agent. Manipulation of data that was not supposed to be used is thus impossible.
- **Conceptual correct.** Multi-agent systems should conceptually be used in a specialised manner. This includes less risk of abusing and confusing different agents, and improved possibilities of emerging new utilisation of the present functionality. It better represents the human society on which the multi-agent concept is based.

Generic communication Using a generic way to exchange messages is shown in the prototype. More information about “what to do” (the state of the message) is contained with the message itself. This has some advantages as well:

- **Reduces size.** Since only one way of communication is used, less size (of agents and their implementation) is needed to support other ontologies.
- **Easier to implement.** Related to the previous is the reduced implementation effort. Only one ontology needs to be implemented and tested, allowing the usage of more shared code.
- **Easier for extension.** Creating an entirely new step in the processing of a message is easier. Since no specialised ontology needs to be implemented for the new service, the router only needs to be modified to utilise the new phase.

Further implementation need to consider this important issue. Although agents can easily be adjusted, creating a sound and consequent system from the start on is always a wise thing to do.

19.7 Other issues out of scope

In 5.3 several issues were left out of the scope for the project. Although they were stated not to be addressed in the project, a minor discussion is reasonable. Notice that scalability and commercial aspects have been addressed in resp. 19.3 and 19.5.

User interfaces No particular effort has been made to design an user interface. The required functionalities have been identified in chapter 9, providing a basis for user interface designers. Great similarity can exist with regular e-mail applications, which have already had a lot of attention to their design. As shown in the prototype, the usage of different agents and devices allows for different and lighter interfaces, with reduced support for certain functionalities.

Security One of the main concerns about mobile agents has been the security of an open execution environment. The underlying platform can resolve this by providing encryption, access-control and similar tools. All these security measures must protect privacy, confidentiality, availability and prevent abuse. The Tryllian ADK [75] resolves many of the security related issues.

Chapter 20

Conclusions

This section describes the conclusions that can be made after the research, development and evaluation in the previous parts of this document. All three main problems and their corresponding goals are considered.

20.1 Unified messaging

With the design and prototype, it is shown that unified messaging can be quite easily accomplished. Through the separation of input, output and transformations, a way to transparently extend the system with new devices has been established. Furthermore, all messages are not simply delivered as ones e-mail, but can be received in any way. The latter can be achieved by using automated transformations, based on chains of smaller conversions. This can allow you to receive all your messages on nearly any device. Since new devices can be added dynamically, any new device can be added as soon as a corresponding agent is available.

An agent-based architecture for personalised unified messaging has been proposed. The architecture provides unified messaging, combined with personalised user profiling across different media. Key element of the architecture is a multi-agent based design, with a mobile agent as semantic router for messages. Proven is this concept can easily be extended with new devices, user interfaces and technologies. The latter allows new developments in message- and media-processing, user-profiling and machine-learning to be efficiently integrated into a practical application. Other properties follow from its agent-based nature, and are listed later on.

20.2 Communication overload

To help an user to overcome a communication overload, information filtering is used. This filtering is used to classify each new message to applicable piles. An user defines a pile for each situation.

It has been demonstrated it is possible to create a user profile across different media. The profiling is separated in two parts for this purpose. First, media specific information is extracted from a message, generating meta-data that can be used for all media. The actual profiling happens next, based on this meta-data.

If the user provides feedback, the user profiling can be implemented so it learns the user's interests. This user profile learning will have to cope with a complex learning problem. In the developed architecture it has been defined to have the following characteristics:

- Heterogeneous available input.
- Heterogeneous input types.
- Dynamic input and output.
- Unknown input and output values during design and implementation.
- The feedback from the user may be inconsistent or erroneous.

No quantitative analysis of algorithms has been performed. The best performing algorithm(s) can thus not be given. Nonetheless, some possible algorithms are estimated to be usable.

In the developed prototype a simple variant of the k -nearest-neighbour algorithm was used. This algorithm was shown to be capable of learning a profile of the user. Although it might not be the best solution, it proofs learning a profile is possible. As important advantage of the developed architecture, each user can easily use its own profiler. This allows the usage of different profiling methods, avoiding the necessity of an one-size-fits-all algorithm. Furthermore, the prototype can be used to easily integrate and test other algorithms.

20.3 Agent technology

Mobile agent technology has successfully been applied to personalised unified messaging. Basis for the developed architecture is a multi-agent system. A mobile agent is used to route a message through the process. As a result, an autonomous message is created.

Some tests have been conducted to evaluate the strengths of the system. Although the developed system can be accomplished with other technologies, a rather unique combination of capabilities are present. Exhibited are the following important features:

- Dynamically extensible. Agents provide two important capabilities to support dynamic extensibility. First, they compartmentalise functionality, encapsulating the characteristics of a device and shield those from the rest of the system. Secondly, agents can be naturally distributed, thus reducing the problem of connectivity.
- Highly adaptable. Another effect of the clear separation of functionality is the high adaptability. The agent only has to respond equally towards the system, allowing modification of its internal behaviour without side-effects. Mobility of the Router Agent increases this flexibility, and allows alternative processing of an individual message to be defined with the message itself, making it independent of the services provided by the rest of the system.
- Robust. Because each agent is kept apart from the others as a process, failures in one agent do not affect others. Since each task of an agent is evaluated to be successful, alternative scenarios in case of failure can be established in advance. Equipping a message with its own agent keeps it independent of the offered services. If a consulted service fails, it can take care itself it is further processed elsewhere.
- Personalisation up to the behavioural level. Since agents are only bound together by their communication, the usage of an agent for personal or public purposes is transparent. An agent can thus provide a customised program for its user for a specific functionality. Other agents can service multiple users, and can supply, for instance, improved transformations for all users at once.

Of course these benefits are not the only one. Other benefits often found in agent systems, such as security and scalability need more investigation, but are estimated to be present in the proposed design as well.

But there are drawbacks as well. A very important implication of the usage of agents is the requirement of a supporting platform on all components in the system. A common runtime platform thus needs to be installed and present as a widespread infrastructure. Because a message is equipped with its own code for increased flexibility, this implies a slight penalty due to this overhead. This will affect performance and efficiency, although the impact is

yet unknown. A last issue is found in security as well. Since new code can be introduced on a hosting system, one has to ensure this code can be trusted.

Agent technology is not the only possible solution when it comes to personalised unified messaging. Most of the benefits can be achieved with other technologies as well, but most only excel on one of the benefits. The use of agents offers the combination of these benefits, which makes it rather unique. If the system will be further developed, agent negotiation can be an important advantage for quality of service. The replacement of agents can be utilised to test and deploy new technologies and insights in media processing and user profiling. Agent technology thus forms a great facilitator for personalised unified messaging.

The Tryllian ADK is a powerful software development kit for mobile agent applications. Nearly all of the required functionality for all aspects of agent technology as used in the design are available. Moving an agent is very simple and can be achieved based on only a hostname. The task model of agents allows for an almost direct mapping of a schedule of jobs to perform to the behaviour of an agent. With a minor addition, the created agents can support personalisation as well. To summarise, the ADK is well suited to support personalised unified messaging. Nonetheless, some recommendations can be found in the next chapter.

Chapter 21

Future Work

Although this study nearly reaches its end, research and development can continue. Similar to almost every software project, progress is never completely finished. Therefore some issues that can be further investigated will be described here. A few other recommendations are given as well.

Each of these issues requires attention of course. If a selection must be made however, user profiling probably deserves the first and most interest. It has the greatest impact on a usable, deployable system.

21.1 User profiling

At least one important issue is left open during the project. This is one of the items that will need further investigation. As described before (for instance in 13.1), automating the process to learn the user's interests is a complicated task. It is shown that learning the user's preferences for receiving ones unified messages can be done.

The question that has not been answered, is how it can be done best. What has been achieved is the setting of the learning problem. Although this is considered a complex one, there is some relieve as well. First of all, there is no need for an one-size-fits-all solution. Since the agent-based architecture allows personalisation, different users can use different algorithms. Secondly, an extremely high accuracy is not needed. A tendency to overestimate a message's importance is preferable, since most users rather receive a few less important messages than missing an important one, certainly if they know they are reachable in the first place.

Nevertheless, one or more appropriate algorithms need to be selected or developed. These should then be tested on a large group of users over a longer period. With the designed architecture and prototype a first environment to

apply and test these algorithms is available. This can reduce the time needed before the developed profiling techniques can be used in practise.

21.2 Security

Since communication is personal, and sometimes even confidential, measures to ensure security are needed. One has to take into account the connection to open systems and use of open networks. Other important reasons for security are dictated by law, in particular laws regarding privacy [5].

The Tryllian ADK uses encrypted connections to protect against malicious interception of data, together with digital signatures to ensure code authenticity during runtime. This provides a secure environment to run ones agent-based applications. Although agent systems can be made secure, this does ensure complete security.

This would involve much more, as is common in computer system security. Procedures and people are involved as well. In all areas large amounts of information are available, including literature specialised in agent-based security.

21.3 Performance

Before a (commercial) realisation can be deployed as a large-scale, operational system, one needs to know how many users and messages it can support. Performance figures cannot be given at the moment. First of all, several parts have not been implemented in a deployable manner. Both (hardware) devices and network environment depend on the choices made by the implementor. These can greatly affect the effective performance in terms of costs and time.

An analysis of quantitative requirements need to be made. This includes an estimation of the average number of messages per user and their size, the number of users, and the desired response-time. Combine these with some tests for achievable throughput. With these and other figures, hardware requirements can be calculated, resulting in a partial number for initial costs. Many other types of quantitative analyses can be performed, and several are necessary. This is not the place to discuss these any further however.

21.4 Towards a virtual secretary

Handling someones communications is one of the tasks of a secretary. A flesh-and-blood secretary does a lot more though. Several of these tasks

can and have been created as digital services as well. Widely used calendar and planning applications already exist. Speech recognition is still under development, but can already allow one to dictate letters.

It is already shown (in 13.2.4 and 13.2.5) that various tasks like calendar and address-books can be integrated in this system. Information retrieval was one of the first task assigned to a personal agent. Perhaps the flexible facilitating capabilities of mobile agent technology can be used for other typical secretary tasks as well. Developing and integrating secretary tasks using agent technology may provide all of us who cannot afford a real secretary with a reasonable alternative.

21.5 Related work

The proposed design has been strictly limited to receiving message-based communication. Other areas are, to a more or lesser extent, related however.

Other communications In section 13.2 this subject has been discussed. Sending messages can be accomplished by reversing some parts of the designed process. Integrating system of several persons can even reduce the need for external communication media and devices. When a few response-time constraints are resolved, partial combination with synchronous communication (i.e. telephone) is possible, to set up a synchronous connection. Until this is accomplished, people might otherwise still need two devices for communicating.

Document Management Systems Another candidate for combination or integration can be Document Management Systems. Many shared interests exist, like classification of documents, usage of meta-data and workflow. Computer Supported Cooperative Work (CSCW) is a similar closely related area. Both systems are closely related to exchanging documents and information between users. Further research for differences and shared properties can further improve each concept and mutual integration.

Agent theories Currently available agent theories already provide guidance for further development of the system itself. The proposed system allows the creation of new ways of processing. Theories of multi-agent systems aim for agent negotiation to create these new processes. These include negotiation, discovery and reasoning to allow new emerging behaviour.

Other results of agent theories can be applied to less futuristic tasks. Agent negotiation for instance, can lead to further refinement of transfor-

mations. The transformation can be negotiated between a Router Agent and a Transformer Agent. The subject of negotiation can be the quality of service (QoS), regarding costs, time and size of the message. These have not been implemented, and thus multi-agent theories leaves room for various improvements.

21.6 Tryllian ADK

Although one of the conclusions, as found in the previous chapter, is that the Tryllian ADK is very well suited to support personalised unified messaging, further improvements can always be made. A few recommendations for some of these improvements are given in this section.

J2ME To get the best out of the design, the UI Agents should be run directly on the user's device. Since a number of the latest PDA's and mobile phones come with support for Java, this is a feasible option. There is a small limitation to this possibility however. The current ADK is build on top of the Java 2 Standard Edition (J2SE). Sun Microsystems has introduced the Java 2 Micro Edition (J2ME) for light and embedded devices. Most of the mobile phones and PDA's fall in this category, and thus provide a J2ME environment. Support for the ADK's Agent Runtime Environment (ARE) would be a great benefit for the proposed architecture.

Locating mechanism An important element in the prototype to support personalisation was the use of a locating mechanism. This mechanism was build from two parts; JNDI and DNS. The Java Naming and Directory Interface is natively supported within the ADK. Within a JNDI *Context*, agents can register themselves by a chosen name. This provides the means to find a certain agent of a certain user in the local habitat. Remote JNDI requests can be made if the remote system's name or IP-address is known.

Dynamically adding a new device to the system was supported with the standard Domain Name System¹. DNS allows to find one or more machine-addresses based on a name, and was used to translate the name of the addressee to the effective machines. This selection of machines was (remotely) searched for the applicable agent, searching all machines is not considered reasonable in a large multi-user system. As mentioned in section 17.1.1, an integrated version of DNS would be beneficial. This can effectively result in a native distributed version of the implemented JNDI.

¹Actually an effectively equivalent hereof for practical reasons

Specialised VAD A last recommendation only applies in a later stadium of possible deployment. One of the shown advantages is the possibility for personal adaptability. It would be unrealistic to have each user write its own agents in code. For this purpose, the Visual Agent Designer might be refitted. A special version with only relevant tasks could be created. All an user has to do is draw a diagram like Figure 14.4. This would provide customisation on the highest-level for a wide range of users.

Bibliography

- [1] AltaVista — translate with the Babel Fish. WWW site. <http://babel.altavista.com/>.
- [2] S. R. van den Berg and P. A. Guenther. Procmail — autonomous mail processor. WWW site and program, 2002. <http://www.procmail.org/>.
- [3] R. Black, A. W. Clark, R. Caley, and P. Taylor. *The Festival Speech Synthesis System*. The Centre for Speech Technology Research, University of Edinburgh, 2002. <http://www.cstr.ed.ac.uk/projects/festival/>.
- [4] G. Boone. Concept features in re:agent, an intelligent email agent. In *Second International Conference on Autonomous Agents (Agent '98)*, 1998.
- [5] J. J. Borking, B. M. A. van Eck, and P. Siepel. Intelligent software agents and privacy. Technical Report Achtergrondstudies en Verkenningen 13, Registratiekamer, January 1999.
- [6] J. M. Bradshaw. An introduction to software agents. In *Software Agents* [7].
- [7] J. M. Bradshaw, editor. *Software Agents*. AAAI, 1997.
- [8] T. Bray et al. Extensible markup language. Specification, World Wide Web Consortium, October 2000.
- [9] D. Chess, C. Harisson, and A. Kershenbaum. Mobile agents: Are they a good idea? Technical Report RC 19887, IBM, December 1994.
- [10] W. W. Cohen. Learning rules that classify e-mail. In H. P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *AAAI Spring Symposium on Machine Learning in Information Access*, pages 307–315. ACM Press, New York, US, 1996.

BIBLIOGRAPHY

- [11] D. H. Crocker. Standard for the format of ARPA internet text messages. RFC 822, Internet Engineering Task Force, August 1982. <http://www.ietf.org/rfc/rfc0822.txt>.
- [12] P. Farjami, C. Görg, and F. Bell. A mobile agent-based approach for the UMTS/VHE concept. In *Proc. Smartnet '99 — The Fifth IFIP Conference on Intelligence in Networks*, Bangkok, Thailand, November 1999.
- [13] D. Ferris. Drowning in email overload? Ferris Research forecasts it will only get worse. Press Release, July 2000.
- [14] T. Finin, Y. Labrou, and M. James. KQML as an agent communication language. In Bradshaw [7].
- [15] FIPA. FIPA agent communication language. Technical report, Foundation for Intelligent Physical Agents, 1999. <http://www.fipa.org/>.
- [16] FIPA. FIPA agent communication language. Technical report, Foundation for Intelligent Physical Agents, 2000. <http://www.fipa.org/>.
- [17] G. Flood. Managers complain of email overload. *Computing*, February 2001. <http://www.vnunet.com/News/1118317>.
- [18] N. Freed and N. Borenstein. Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies. RFC 2045, Internet Engineering Task Force, November 1996. <http://www.ietf.org/rfc/rfc2045.txt>.
- [19] N. Freed and N. Borenstein. Multipurpose internet mail extensions (MIME) part two: Media types. RFC 2046, Internet Engineering Task Force, November 1996. <http://www.ietf.org/rfc/rfc2046.txt>.
- [20] N. Fujino, Y. Matsuda, T Nishigaya, and I. Idia. Multi-agent solution for virtual home environment. In Hayzelden and Bourne [24], chapter 8, pages 102–110.
- [21] GSM world news — press releases. WWW site, 2001. <http://www.gsmworld.com/news/statistics/index.shtml>.
- [22] L. Hagen, J. Mauersberger, and C. Weckerle. Mobile agent based services subscription and customizing using the UMTS virtual home environment. *Computer Networks*, 31(19):2063–2078, August 1999.

BIBLIOGRAPHY

- [23] U. Hanani, B. Shapira, and P. Shoval. Information filtering: An overview of issues, research and systems. *User Modelling and User Adaptive Interaction*, 11:203–259, 2001.
- [24] A. L. G. Hayzelden and R. A. Bourne, editors. *Agent technology for communication infrastructures*. John Wiley & Sons Ltd., 2001.
- [25] A. L. G. Hayzelden et al. Future communication networks using software agents. In A. L. G. Hayzelden and J. Bigham, editors, *Software Agents for Future Communication Systems*, pages 1–57. Springer, 1999.
- [26] P. Helmersen et al. Impacts of information overload. Technical Report P947, Eurescom, January 2001. <http://www.eurescom.de/public/projects/P900-series/P947/>.
- [27] Hotvoice.com — world’s largest free web based unified messaging network. WWW site, 2002. <http://www.hotvoice.com/>.
- [28] X. Huang, A. Acero, and H.-W. Hon. *Spoken Language Processing: A guide to Theory, Algorithms and System Development*. Prentice Hall, 2001.
- [29] M. N. Huhns and L. M. Stephens. Multiagent systems and societies of agents. In G. Weiss, editor, *Multiagent system: a modern approach to distributed artificial intelligence*. MIT press, 1999.
- [30] IDC. Email mailboxes to increase to 1.2 billion worldwide by 2005. Press release, September 2001. <http://www.cnn.com/2001/TECH/internet/09/19/email.usage.idg/>.
- [31] Active IETF working groups. WWW site, 2002. <http://www.ietf.org/html.charters/wg-dir.html>.
- [32] W. Kepinski. De keuzes in unified messaging. *Infoworld (Dutch edition)*, 6(19), October 2001.
- [33] D. Kerr et al. An agent-based platform for next-generation IN services. In Hayzelden and Bourne [24], chapter 2, pages 19–31.
- [34] Y. Labrou. Standardizing agent communication. In M. Luck et al., editors, *Multi-Agent Systems & Applications, Advanced Course on Artificial Intelligence (ACAI-01) proceedings*, number 2086 in Lecture Notes in Artificial Intelligence. Springer, 2001.

BIBLIOGRAPHY

- [35] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, March 1999.
- [36] M. Lauff, A. Schmidt, and H. Gellersen. A universal messaging agent integrating device modalities for individualised mobile communication. In *Proc. of the 13th International Symposium on Computer and Information Sciences*, Belek-Antalya, Turkey, October 1998.
- [37] S. Lloyd, A. L. G. Hayzelden, and L. G. Cuthbert. Virtual home environments to be negotiated by a multi-agent system. In Hayzelden and Bourne [24], chapter 9, pages 111–121.
- [38] S. A. Macskassy, A. A. Dayanik, and H. Hirsh. EmailValet: Learning user preferences for wireless email. In *IJCAI-99 Workshops: Learning about Users and Machine Learning for Information Filtering*, Stockholm, Sweden, August 1999.
- [39] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, July 1994.
- [40] N. Marmasse and C. Schmandt. Location-aware information delivery with *comMotion*. In *HUC 2000 Proceedings*, pages 155–171, 2000.
- [41] S. J. W. Marti. Active messenger: e-mail filtering and mobile delivery. Master’s thesis, MIT Media Lab, August 1999.
- [42] M. Marx and C. Schmandt. CLUES: Dynamic personalized message filtering. In *Proc. of ACM Computer Supported Cooperative Work*, pages 113–121, November 1996.
- [43] L. McIntyre et al. File format for internet fax. RFC 2301, Internet Engineering Task Force, March 1998. <http://www.ietf.org/rfc/rfc2301.txt>.
- [44] P. McNamara. E-mail overload drives many users bananas. WWW page, June 1998. <http://www.cnn.com/TECH/computing/9806/18/email.overload.idg/>.
- [45] F. Meech, K. Baker, E. Law, and R. Liscano. A multi-agent system for personal messaging. In *Proc. of the 4th Int. Conf. on Autonomous Agents 2000*, pages 144–145, 2000.
- [46] Message4u. WWW site, 2002. <http://www.message4u.nl>, in Dutch.
- [47] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

BIBLIOGRAPHY

- [48] K. Mock. An experimental framework for e-mail categorization and management. In *24th Annual Int. ACM SIGIR Conf. on Information Retrieval (SIGIR '01)*, September 2001.
- [49] P. Mockapetris. Domain names — concepts and facilities. RFC 1034, Internet Engineering Task Force, November 1987. <http://www.ietf.org/rfc/rfc1034.txt>.
- [50] MySQL AB. *MySQL database*. <http://www.mysql.com/>.
- [51] Nokia 7650. WWW page — product description, 2002. <http://www.nokia.com/phones/7650/>.
- [52] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 1996.
- [53] M. Pantic. *Facial expression analysis by computational intelligence techniques*. PhD thesis, Delft University of Technology, 2001.
- [54] A. Pizano and W. V. Su. Multimedia messaging systems. In B. Furht, editor, *Handbook of Internet and Multimedia systems and applications*, pages 420–436. CRC Press, 1998.
- [55] J. B. Postel. Simple mail transfer protocol. RFC 821, Internet Engineering TaskForce, August 1982. <http://www.ietf.org/rfc/rfc0821.txt>.
- [56] B. Raman, R. H. Katz, and A. D. Joseph. Universal inbox: Providing extensible personal mobility and service mobility in an integrated communication network. In *Proc. of the Workshop on Mobile Computing Systems and Applications (WMCSA '00)*, December 2000.
- [57] J. D. M. Rennie. ifile: An application of machine learning to e-mail filtering. In *Knowledge Discovery and Data Mining (KDD-2000); Text mining Workshop*, Boston, MA USA, 2000. ACM.
- [58] M. Roussopoulos et al. Person-level routing in the mobile people architecture. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, October 1999.
- [59] S. Russell and P. Norvig. *Artificial Intelligence, a modern approach*, chapter 2: Intelligent agents. Prentice Hall, 1995.
- [60] N. Sawhney. Contextual awareness, messaging and communication in nomadic audio environments. Master's thesis, MIT Media Lab, June 1998.

BIBLIOGRAPHY

- [61] R. M. Schaar. Literature survey. Technical report, Delft University of Technology / Tryllian B.V., 2001.
- [62] R. B. Segal and J. O. Kephart. Incremental learning in SwiftFile. In *Proceedings of the Seventh International Conference on Machine Learning*, June 2000.
- [63] Siemens SX45. WWW page — product description, 2002. http://www.siemenscomms.co.uk/online/catalogue/product_display.php?ID=498.
- [64] A. Silberschatz and P. Galvin. *Operating Systems Concepts*, chapter 18. Addison-Wesley, 4th edition, 1994.
- [65] Sun Microsystems. *Java Naming and Directory Interface*. <http://java.sun.com/products/jndi/>.
- [66] Sun Microsystems. *JDBC*. <http://java.sun.com/products/jdbc/>.
- [67] Sun Microsystems. *Swing*. <http://java.sun.com/j2se/1.3/docs/guide/swing/>.
- [68] Sun Microsystems. *Javamail API*, 1.2 edition, December 2000. <http://java.sun.com/products/javamail/>.
- [69] Sun Microsystems. Java 2 Standard Edition. Software Development Kit, 2002. <http://java.sun.com/j2se/>.
- [70] J. Takinnen and S. Shahmehri. Are you busy, cool, or just curious? — CAFE: a model with three different states of mind for a user to manage information in electronic mail. *Human IT*, 2(1), March 1998.
- [71] Inc. Tornado Development. Tems unified messaging — e-mail, voice mail, faxing and paging. WWW site, 2002. <http://www.tems.com/>.
- [72] Tryllian. SmartAgent. Internal methodology, Tryllian B.V., May 2001.
- [73] Tryllian. Tryllian agent development kit. Technical white paper, Tryllian B.V., 2001. <http://www.tryllian.com/>.
- [74] Tryllian. *Tryllian ADK developer's guide*, 2002. <http://www.tryllian.com/>.
- [75] Tryllian. Tryllian Agent Development Kit. Software Development Kit, 2002. <http://www.tryllian.com/>.

BIBLIOGRAPHY

- [76] A. Tveit. A survey of agent-oriented software engineering. In *Proceedings of the First NTNU Computer Science Graduate Student Conference*. Norwegian University of Science and Technology, May 2001.
- [77] UMTS forum. What is UMTS? WWW page, 2002. http://www.umts-forum.org/what_is_umts.html.
- [78] Unified messaging, e-mail, fax, voicemail, sms, phone all in one in-box. WWW site, 2002. <http://www.unified-messaging.com/>.
- [79] UnifiedMessaging.com — unified messaging news and information portal. WWW site, 2002. <http://www.unifiedmessaging.com>, provided by Captaris, a company offering unified messaging solutions for businesses.
- [80] G. Vaudreuil and G. Parsons. Voice profile for internet mail — version 2. RFC 2421, Internet Engineering Task Force, September 1998. <http://www.ietf.org/rfc/rfc2421.txt>.
- [81] W. de Vries. Spam kan i-mode de kop kosten. WWW page, April 2002. http://www.infoworld.nl/nieuws/bericht.phtml?id=4397&is_nieuws=1, in Dutch.
- [82] S. Whittaker and C. Sidner. E-mail overload: Exploring personal information management of e-mail. In *Conference proceedings on Human factors in computing systems*, pages 276–283, 1996.
- [83] M. Wooldridge and P. Ciancarini. Agent-oriented software engineering: The state of the art. In P. Ciancarini and W. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in AI*. Springer-Verlag, Januari 2001.
- [84] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.
- [85] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.
- [86] XOIP unified messaging. WWW site, 2002. <http://www.xoip.nl/>.
- [87] C. K. Yeo, S. C. Hui, I. Y. Soon, and C. T. Lau. A unified messaging system on the internet. *Microprocessors and Microsystems*, 24:523–530, November 2001.

BIBLIOGRAPHY

- [88] J. Zawinski. Message threading. WWW page, 2002. <http://www.jwz.org/doc/threading.html>.
- [89] G. Zhou. Managing the e-mail explosion. WWW page, September 2000. <http://www.cnn.com/2000/TECH/computing/09/06/e-mail.management.idg/>.

List of Figures

8.1	Sample FIPA message (fictive)	48
11.1	Chains of transformations	70
11.2	Centralised versus distributed routing over network	72
12.1	Overview of the architecture	80
12.2	Workflow for an important new message	96
12.3	Dataflow for an important new message	97
12.4	Workflow for an unimportant new message	98
12.5	Dataflow for an unimportant new message	99
12.6	Workflow user situation change	100
12.7	Dataflow user situation change	101
12.8	Workflow user feedback	102
12.9	Dataflow user feedback	103
12.10	Workflow store message	104
12.11	Dataflow store message	105
12.12	Workflow request message	106
12.13	Dataflow request message	107
12.14	Workflow present message	108
12.15	Dataflow present message	109
14.1	Task hierarchy of a Factory Agent	122
14.2	Task states of a Factory Agent	122
14.3	Task hierarchy of the Router Agent	123
14.4	Task states of the Router Agent	124
14.5	Task state of each Router Agent phase	125
14.6	Task hierarchy of an Extractor Agent	126
14.7	Task states of an Extractor Agent	126
14.8	Task hierarchy of a Profile Agent	127
14.9	Task state of a Profile Agent	128
14.10	Task hierarchy of a Transformer Agent	128

LIST OF FIGURES

14.11	Task state of a Transformer Agent	129
14.12	Task hierarchy of an UI Agent	130
14.13	Task state of an UI Agent	131
14.14	Task hierarchy of a Storage Agent	131
14.15	Task state of a Storage Agent	132
14.16	UML object model for agents	133
14.17	Partial UML for the Router Agent	134
15.1	Sample envelope	136
15.2	Sample article reference (RSS) from Slashdot.org	138
15.3	Sample code of the created task-model	140
15.4	Screen-dump of the Rule Profile Agent statistics	142
15.5	Sample rule set for the Rule Profile Agent	142
15.6	Screen-dumps of the Swing UI Agent	145
16.1	Profile learning: Cumulative error	173
16.2	Profile learning: Errors per messages	173
16.3	Profile learning: Extra assigned situations	174
16.4	Profile learning: Missing situations	174
16.5	Profile learning: Various user inconsistencies compared	175
A.1	Screen-dump of the Visual Agent Designer	232
B.1	Example workflow with legend	233
B.2	Example dataflow with legend	234
B.3	Example task hierarchy with legend	235
B.4	Example task-state transition with legend	236

List of Tables

12.1	Properties of a Factory Agent	84
12.2	Properties of the Router Agent	85
12.3	Properties of the Extractor Agent	86
12.4	Properties of the Profile Agent	88
12.5	Properties of the Transformer Agent	90
12.6	Properties of the UI Agent	91
12.7	Properties of the Storage Agent	92
12.8	Agent communication	93
13.1	Meta-data example 1	114
13.2	Meta-data example 2	114
13.3	Meta-data example 3	114
16.1	User's reaction to classes of messages	172

Part VII

Appendices

This part contains some additional information. The first appendix forms a description of the Tryllian Agent Development Kit. The used diagrams are explained in appendix [B](#). Abbreviations used throughout the document can be found in appendix [C](#). Page [239](#) and further hold a paper written on the subject of this project.

Appendix A

Tryllian ADK

The Tryllian Agent Development Kit (ADK) is an industrial strength software development environment based on mobile agent technology. It implements a number of relevant standards, such as FIPA ACL [15], JNDI [65], JXTA, SOAP, SNMP and XML [8], and is purely Java based. Below the concept of the ADK will be accounted for. Afterwards, further details of important components will briefly be described. For further information, please visit the Tryllian website at <http://www.tryllian.com/>.

Concept Concept behind the ADK can be divided in several parts. The first is an *habitat*, the environment in which agents are deployed. The habitat supports all facilities needed for the other concepts. It runs in a Java Virtual Machine and manages all other concepts.

The second is a *room*, a logical part of a habitat, that can contain agents. A room can contain many agents, but an agent can and must be in only one room.

An *agent* itself is a mobile agent, consisting of a body and behaviour. The behaviour is defined by tasks and its knowledge. Agents can move between various habitats and rooms, after which it can continue its tasks as before it moved. An agent is also capable of communicating with other agents based on asynchronous messaging.

A.1 Architecture

The ADK consists of two important parts. The first is the Agent Runtime Environment (ARE) which provides a basis for running and using agents. The other are the Agent Foundation Classes (AFC), that provide the fundamentals to create agents.

Agent Runtime Environment The ARE is the supportive basis for agents to exist and run. It provides the infrastructure needed by agents to operate. These include an environment where agents are run, facilities to transport an agent from one system to another and the means for agents to communicate. Various system agents are available as well, to implement these capabilities and offer various services agents might need. Noteworthy example of the latter is the support for JNDI-queries, allowing agents to find other agents even from remote systems. Other elements the ARE takes care of include security and persistency of agents.

Agent Foundation Classes To create agents, one can use the AFC. These foundation classes form the API for the programmer to create agents. The AFC is formed by a rich collection of Java-libraries, that provide the necessary basic needs of an agent.

It provides various standard tasks, that can be used to construct an agent. Examples include tasks for moving, waiting and communicating. Other classes form the body of an agent itself or can be used to construct languages for communication.

A.2 API highlights

Two elements of the Tryllian ADK API are described below. They are mentioned throughout this document. The discussed elements from the API are the `TaskScheduler` and the `MoveTask`.

TaskScheduler The `TaskScheduler` is one of the two main classes that can be used to create tasks for an agent. Important difference with the other, `DefaultTask`, is the way it is executed. Normally, all subtasks of a task can be executed in parallel, thus effectively all at the same time¹. Within a `TaskScheduler` however, all subtasks are executed in sequence, in an order depending on the success of each subtask. Each task can either succeed or fail, and to both results, another task can be bound. In other words, one can simply create branches of a normal sequence for each job that might fail. A simple example is²: `addTask(A, B, C);`. After task A finishes, task B is normally executed if A succeeded. If task A failed however, task C will be run next. Loops and nesting of tasks are allowed as well, providing for very powerful constructions. Figure A.3 includes a visualisation of a simple task that can be accomplished this way.

¹Technically there are differences with this concept.

²Without any constructor or initialisation...

MoveTask Another important task is the `MoveTask`. This task is all that is needed to move an agent from one place to another. These places are habitats or rooms within these habitats. Only argument needed for a `MoveTask` is the destination *location*. This location can be constructed from a hostname or IP-address in the standard configuration. A hostname is a textual Internet address, like `www.tryllian.com`. Making an agent move around can thus be accomplished solely based on names.

A.3 More

Although the AFC and ARE are very important components of the ADK there is more. Several tools will be named, and some pointers for further readings follow as well.

Tools Various tools are provided with the ADK as well. Some of these tools are essential, but some are optional. An essential tool is of course the composing tool, which allows a developer to construct and digitally sign an agent. Optional tools include for instance a habitat-visualisation.

Another tool is the Visual Agent Designer (VAD), that allows easy composition of agents. A developer using the VAD can construct agents, even without any prior knowledge of Java. Creating agents becomes as simple as drawing as task-transition diagrams. This is a drag-and-drop utility with various useful tasks, and other *building blocks* are available for various functionalities. A screen-dump is shown in Figure A.3.

Further information More information can be obtained from the Tryllian website: <http://www.tryllian.com/>. A more elaborate introduction to the ADK can be found in a *Technical White Paper* found on the website [73]. A complete *Developers Guide* is available for developers of agents with the ADK [74].

Research licenses are available for universities and other research institutes. Last, the next release (ADK v2.0) is scheduled to be released for free³ in June 2002. This can be obtained through the Tryllian website as well when available.

³Commercial usage is excluded.

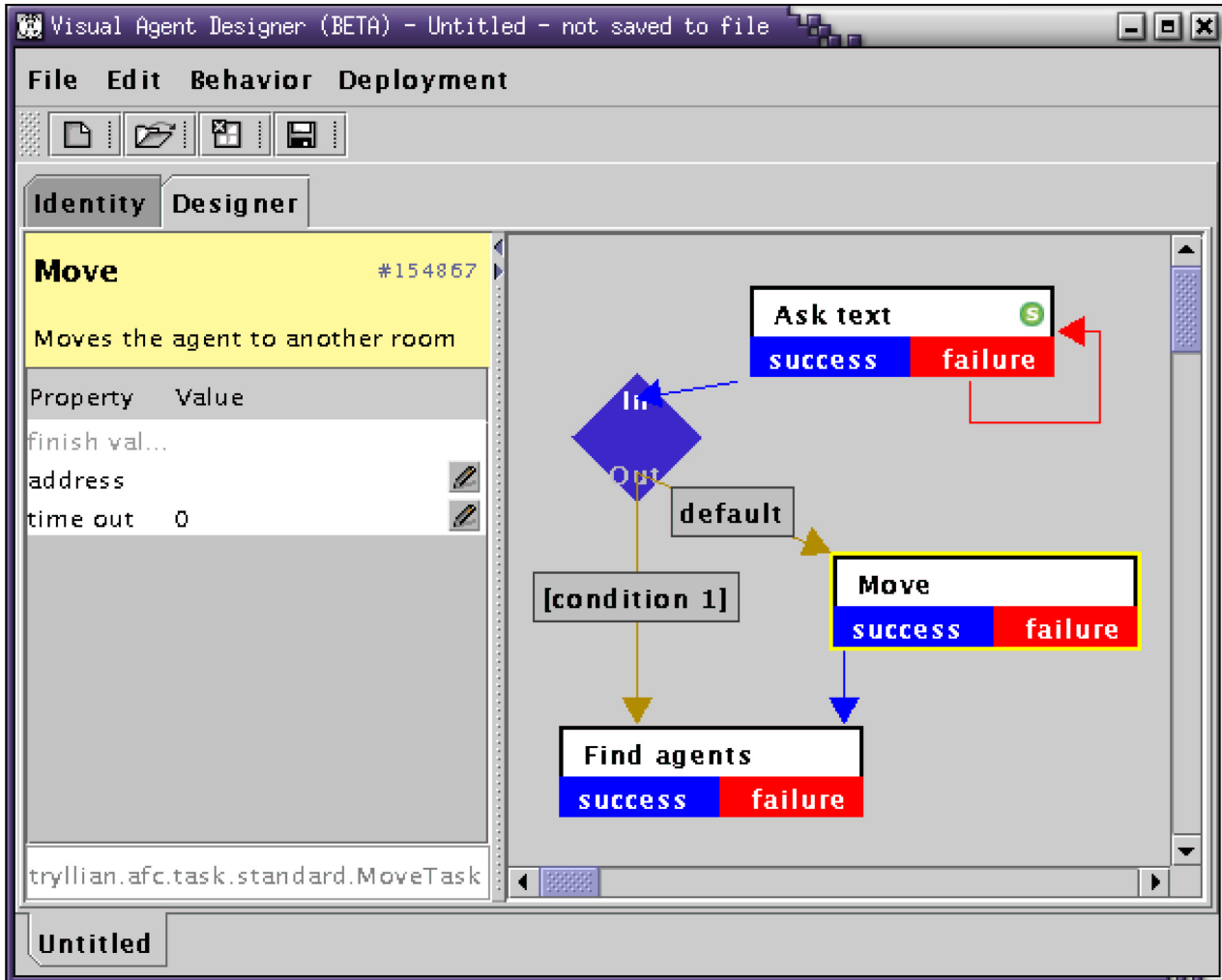


Figure A.1: Screen-dump of the Visual Agent Designer

Appendix B

Diagram Legend

Throughout this document, several types of diagrams have been used. In this appendix, the legend of each of these diagram-types is given. This should explain the diagrams, in case some are not obvious yet. In the first two diagram types a green arrow indicates the diagram is (logically) continued in another figure. Note that UML, as used for the object models in 14.2 can be found in various literature and will not be described, since no specific adjustments or notations are used.

B.1 Workflow

The workflow diagrams, as given in section 12.6. An example is the workflow in case an important message arrives in Figure 12.2. These diagrams are standard swimlane-diagrams.

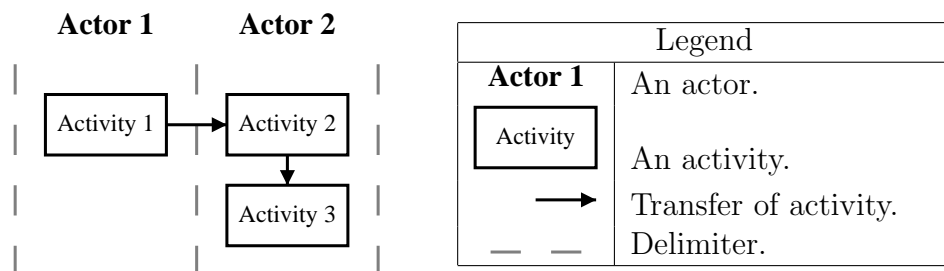


Figure B.1: Example workflow with legend

An *actor* is an entity that deploys activities in the system. An *activity* is an operation in the process, performed by the actor. The *transfer* of an activity in the process is the transfer of an activity of one actor to another activity of another actor. The vertical version represents a transfer from

one activity to another within the same actor. A *delimiter* that separates columns represents the boundaries of the actor's responsibilities, which is shown on top of the column. Arrows of transfers indicate the chronological sequence of activities in a process.

B.2 Dataflow

Dataflow diagrams display how the data is exchanged between various components of the system. One can for instance refer to Figure 12.15 for the dataflow used for the presentation of a message to the user. This section explains the meanings of all used symbols.

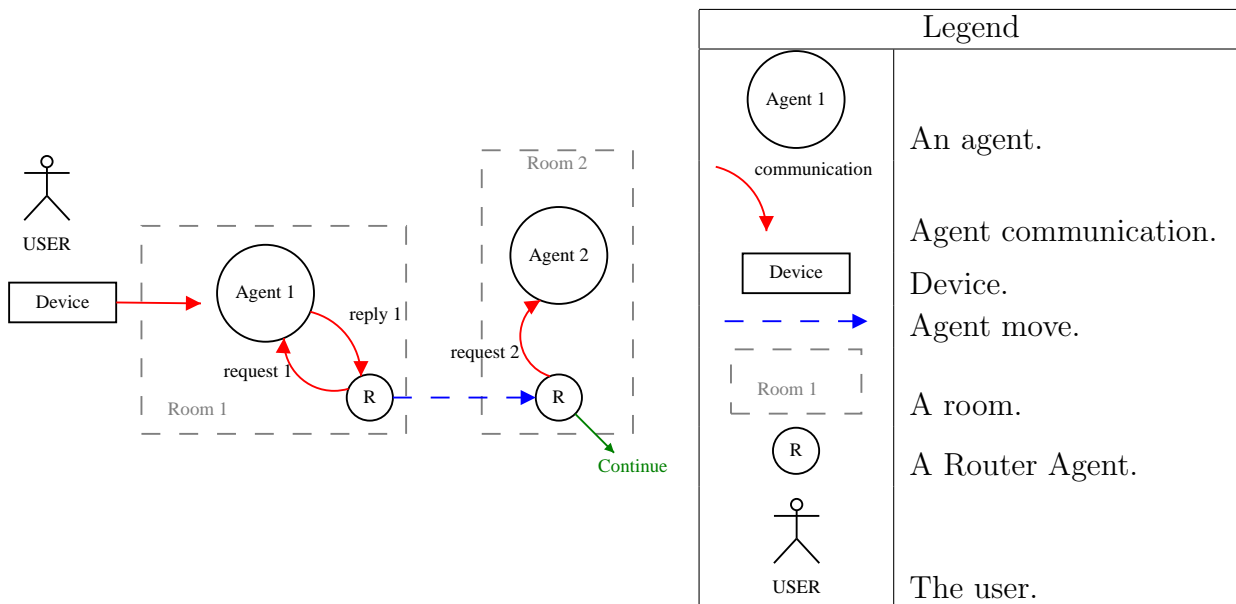


Figure B.2: Example dataflow with legend

An *agent* is a software agent that is part of the system. Each agent is located in a *room*, although the used rooms in the diagrams are more or less conceptual. A room can be empty or contain more agents, and rooms may be located throughout the network in various habitats, but an agent can be in only one room at a time. An agent can *move* between different rooms, thereby leaving its original location. In the used diagrams, the *Router Agent* is the only mobile agent, and each representation of a Router Agent thus represents the same agent before and after the connecting move. Agents can *communicate*, this is where the actual exchange of data happens. The last entities are not agents, they are the *user* and its *devices* and can communicate

as well. By following the arrows, one can determine the order of data- and agent transfers.

B.3 Task hierarchy

The task hierarchies are given as part of the prototype's design in chapter 14. A task hierarchy shows the relation between tasks. It shows how a task consists of other tasks, and vice versa. One has to read these from the left to the right.

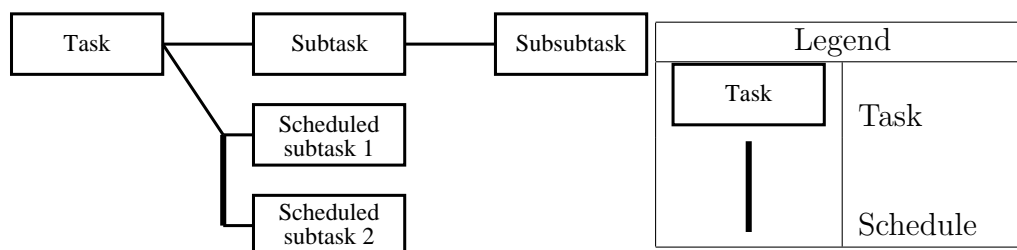


Figure B.3: Example task hierarchy with legend

A *task* is a sort of job to be performed by an agent. When a task is divided in several smaller jobs, these are subtasks, subsubtask etc. . . Tasks with multiple subtasks may have scheduled subtasks. Normal subtasks are all run at the same time (in parallel), but a *schedule* defines the transitions from one subtask to another. Scheduled subtasks are executed in sequence, and their order is determined by their success and the schedule. These are further described by task-state transitions diagrams, as described in the next section. More information about tasks and schedules can be found in [74].

B.4 Task-state transition

The mutual relation of several tasks is visualised by a task-state transition diagram. Figure 14.4 for instance, describes the transitions for the task that handles the routing in a Router Agent. Below an example with a legend and explanation.

A task is *initialised* by the agent itself. This can be due to the initialisation of the agent itself, a reaction to an external event or another reason. A *subtask* is performed, and when it finishes, a transition to another task can be made. Such a *transition* can have a *condition*, which determines the next task, based on the result of the previous subtask. A *command* is a special

APPENDIX B. DIAGRAM LEGEND B.4. TASK-STATE TRANSITION

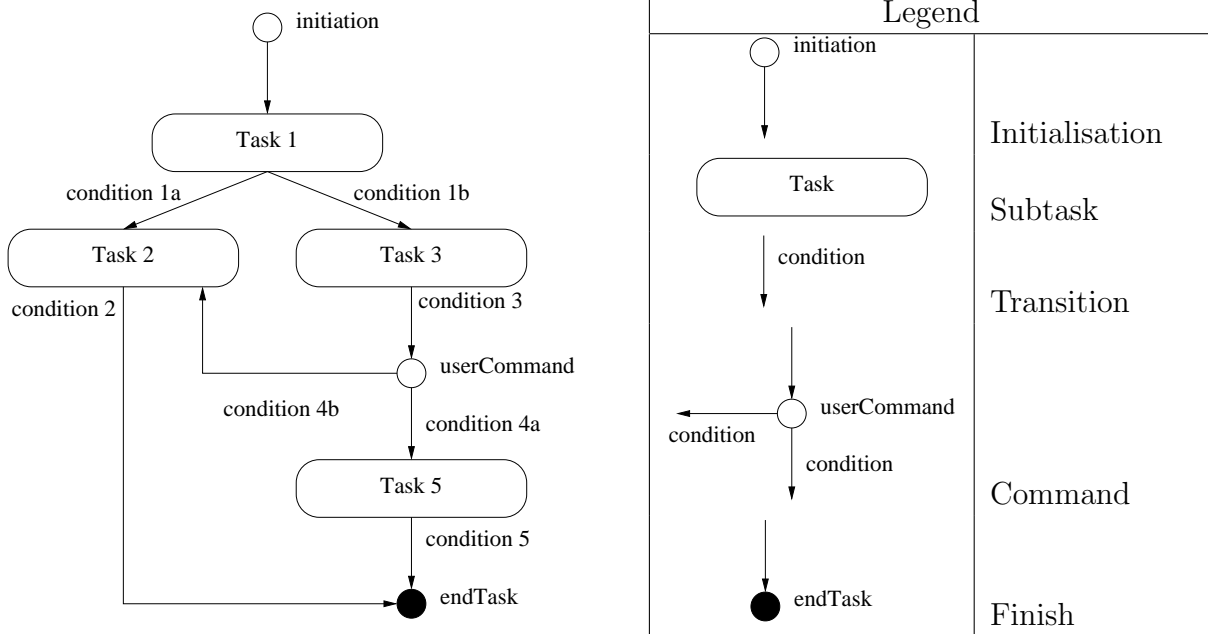


Figure B.4: Example task-state transition with legend

kind of task, it waits for a command of the user or another special external event. A task is completed when it *finishes*.

Appendix C

Abbreviations

- ACL** Agent Communication Language
- ADK** Tryllian Agent Development Kit
- ADSL** Asymmetric Digital Subscriber Line
- AFC** ADK Agent Foundation Classes
- AI** Artificial Intelligence
- AIM** AOL Instant Messenger
- API** Application Programmers Interface
- ARE** ADK Agent Runtime Environment
- ARPA** Advanced Research Projects Agency
- ASP** Application Service Provider
- CLID** Calling Line IDentification
- CSCW** Computer Supported Cooperative Work
- DNS** Domain Name System
- DSL** Digital Subscriber Line
- ICQ** I-seeq-you
- IF** Information Filtering
- IR** Information Retrieval

APPENDIX C. ABBREVIATIONS

IVR Interactive Voice Response

JNDI Java Naming and Directory Interface

JVM Java Virtual Machine

FIPA Foundation for Intelligent Physical Agents

GPRS General Packet Radio Service

GPS Global Positioning System

HTML HyperText Markup Language

KQML Knowledge Query and Manipulation Language

KSE ARPA Knowledge Sharing Effort

MAS Multi-Agent System

MAT Mobile Agent Technology

MSM MicroSoft Messenger

NLP Natural Language Processing

OCR Optical Character Recognition

PDA Personal Digital Assistant

PDF Portable Document Format

PNG Portable Network Graphics

RFC Request for Comments

SMTP Simple Mail Transfer Protocol

TF-IDF Term Frequency - Inverse Document Frequency

TTS Text to speech

UMTS Universal Mobile Telecommunications System

VAD ADK Visual Agent Designer

VHE Virtual Home Environment

XML eXtensible Markup Language

Appendix D

Paper

This paper has been submitted for the Belgian-Dutch Conference on Artificial Intelligence (BNAIC) 2002. Notification of acceptance is scheduled for July 5, 2002. Since this is after the date of publication of this document, publication is unknown at the time of writing.

Agent-Based Intelligent Personal Unified Messaging

R. M. Schaar ^a L. J. M. Rothkrantz ^a M. Lassche ^b
M. V. Jonkers ^b

^a Delft University of Technology, Mekelweg 4, 2628 CD Delft

^b Tryllian B.V., Joop Geesinkweg 701, 1096 AZ Amsterdam

Abstract

People who are already flooded with e-mail, will be overloaded by unified messaging now telecommunication and Internet are merging together. An architecture based on multi- and mobile-agents is proposed as solution. Personalised behaviour is included in a flexible and extensible system. New technology, insights and developments can be integrated in cross-media user profiling. This is facilitated by using meta-data to describe a message, extracted in a separate phase from the profiling itself. An user profile classifies messages that apply to the user's current situation, thus reducing the user's new messages to the important ones. A prototype of the architecture has been implemented and was used for an evaluation.

1 Introduction

“You have thirteen new e-mails, six new SMS-messages, three people tried to ICQ with you, there are four new news-items, seventeen new Usenet-discussions in your favourite groups and you have two faxes and one voice-mail.”

With the forthcoming merger of telephone-networks and Internet, it becomes possible to create an integrated communication-portal for users. This will allow users to receive all their messages using a single application. Included are all kinds of message-based communications, from e-mail to voice-mail and CNN-headlines to Usenet-discussions. But would it not be more usable if the above was stated as follows:

“You have four new messages that are currently important.”

The last statement is more informative for the user, a lot more comprehensible and shorter as additional gain. Previous work in this area mainly focused on e-mail, using text-based information filtering. Here it is tried to accomplish the same, but applied across all possible media found in unified messaging. This article will discuss problems involved, present an architecture and its evaluation based on a created prototype.

2 Problem

Telecommunication allows us to receive new messages anywhere and at any time. With modern mobile telecommunication equipment combined with the Internet, people can communicate everywhere. People will therefore be able to be continuously reachable for all kinds of messages. Three issues with regard to this phenomenon are considered in this study:

Unified messaging Telecommunication and Internet are converging to one global communication network. As an advantage, it is nowadays possible to receive your voice-mails and faxes as regular e-mail. Through mobile phones and computers, you can access your e-mail everywhere. Although one can receive the data of all received messages, one cannot always access the information in the message. Using a phone to access a fax for example, is not a trivial task. An user should be able to access messages independent of the used device [5]. The question is how the conversion of different media can be automated, so the information can be received rather than the data.

Communication overload Many people already complain about a flood of new e-mail every day. The usage of e-mail is still growing, being used by more people, who use it more frequently. Similar to the *information overload* encountered when searching the Internet, people experience an *e-mail overload* [10]. With more means of communication tied together, this can increase even further. This problem might therefore become a broader *communication overload* [4].

To overcome the e-mail overload, information filtering has been applied. Most of the proposed solutions to handle e-mail are text-based however (e.g. [7]). Since unified messaging includes all kinds of media, a similar solution cannot be used. What needs to be addressed is: A) how filtering can be accomplished across different media, and B) can such a mechanism improve its performance in time with machine learning.

Agent benefits Agents have been successfully used to help solve various problems. A last issue is the way agents can be deployed for this particular problem and what the benefits of using agent technology will be.

3 Design

An architecture was developed to cope with these problems. Figure 1 presents an overview of the designed architecture. Described below will be the architecture in general, the method of profiling and how messages are routed in the system.

3.1 Architecture

A multi-agent architecture has been developed for personalised unified messaging. Seven types of agents are identified to be necessary. Three of these handle standard

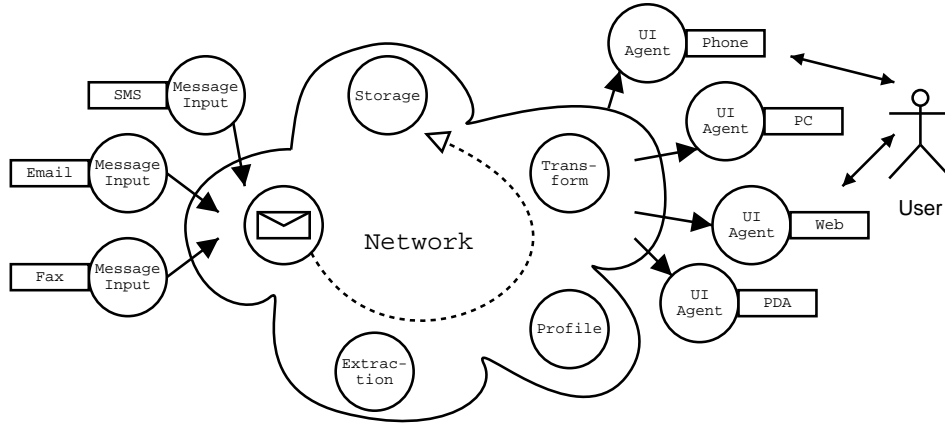


Figure 1: Overview of the designed architecture.

input, output and storage of messages. A fourth agent performs transformations of messages from one media-format to another. Each transformation is build of a sequence of smaller standard document-conversions [5]. Two more agents take care of the filtering of messages, they will be explained in 3.2. The seventh and last type of agent conducts the routing of messages, and is discussed in 3.3. Each of these agents offers a function, but shields the details from the rest of the system.

No unified format is used to represent the message, to avoid useless conversions or the (unrealistic) choice for one universal type of medium. A message is considered as a *package* containing the message in its original data format, until it is transformed to the preferred format. Additional information with regard to the message can be part of the package as well.

3.2 Profiling

To be able to filter messages of different media-types, the actual filtering is divided among two agents. These are the extractor agent and the profile agent. The former performs a data analysis, while the latter should take care of filtering, modelling and learning when compared with common information filtering [3].

Extractor Agent The first agent performs (media-specific) extractions on messages. It annotates the message with information implicitly present in the message. This information is made explicitly available as meta-data. Meta-data can be any kind of descriptive information about the actual message, the sender or any other relevant information for the addressee. Note that these meta-data need to consist of machine-processable types.

Profile Agent All user profiling is performed by the Profile Agent. This agent decides on two topics; whether messages are important and in which format they

should be shown. The former is accomplished by assigning applicable situations to a message, placing messages on virtual *piles*. Each situation is represented by a pile, for instance home, work and travel, and a message can be on multiple piles. The format decides how messages should be shown, based on the capabilities of the user's current device and the preferences of the user in the past.

Users have to provide their *current situation*, which determines whether messages are important. Only messages that are on the pile representing the user's current situation will be delivered to the user. Users also have to provide *feedback*, so the profile can be improved and learn the user's interests for each situation. The profile is based on this feedback, combined with the meta-data of messages. Since it is based on this meta-data, the profile can be established independent of the used medium.

This user profile learning is a rather complex problem:

- The present fields (meta-data) can vary per message.
- The fields are heterogeneous (numerical, textual, labels, etc. . .).
- At design and implementation the actual fields are unknown.
- Both fields and target values (interests) can change over time.
- Users will not always be consistent and make mistakes.

3.3 Routing

A separate *mobile* agent is used as a router. It is separate of the other agents to isolate all functions that require knowledge of the environment, such as the other agents and the network. The router dynamically locates, *per user*, the agent where the message has to be delivered. This closely resembles the idea of the *mail transfer agent* in regular e-mail. The agent suggested here is mobile however, and each message is handled by its own agent. Messages are thus encapsulated by a router agent, effectively creating an autonomous message.

It consults the agents described previously. The order in which they are consulted is determined by the router agent however. All processing — like extraction, profiling and transformation — is delegated to these other agents. A router agent thus acts as a *semantic router*, that uses other agents as *consultants* [2].

Figure 2 shows the standard process a message is subjected to. After meta-data is extracted, applicable situations are assigned. Unimportant messages are saved, but important messages are assigned a format as well, which is acquired next. After the message is shown to the user, the message can be saved. The additional transitions provide alternative scenarios in case failures occur. Each step in this task-model can invoke a move of the router agent, if the agent with the required service is located remotely.

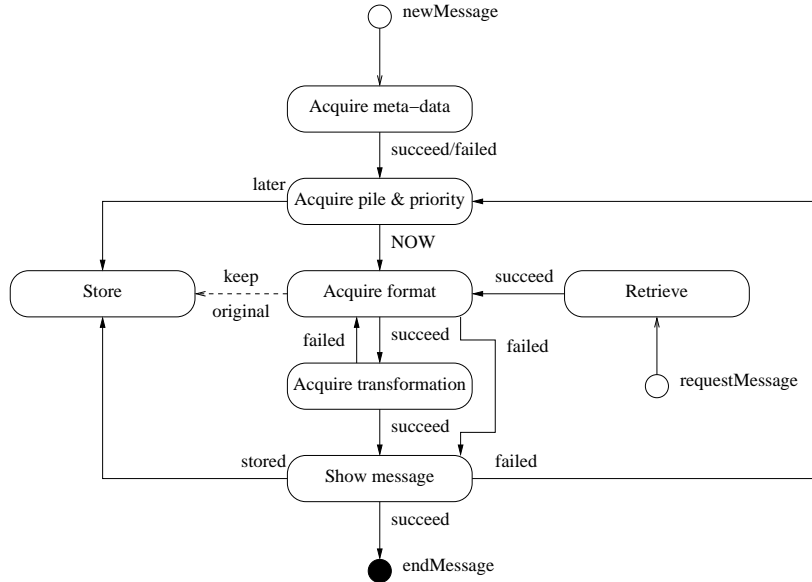


Figure 2: Task model for the Router Agent

4 Prototype

Based on the previously presented design, a prototype has been implemented. It was created as proof-of-concept and for evaluation of the architecture. As basis for the implementation the Java-based Tryllian Agent Development Kit (ADK) [9] has been used. The ADK supports the FIPA Agent Communication Language, and the agents communicate according performatives of this standard. Some elements have been simulated for practical reasons, using the modalities instead of the full devices or processing.

4.1 Description

Of all types of agents, a few different instances were built. Included in the implementation are e-mail, SMS and news-headlines as inputs. As output for the user a desktop application, telephone and SMS are available. The prototype is described in further detail in [6]. Two interesting aspects will be further described below, and will start with the routing.

The task-state transition of figure 2 is almost directly implemented, using the task-model behaviour of agents in the Tryllian ADK [8]. Each message in the design is handled by its own mobile agent, thus equipped with its own individual code of this model. The process a message is subjected to is determined by this code, so it can be varied per addressee, originating location or other criteria. This

creates a highly flexible system, where the processing can be modified when needed. These routing agents can locate other agents through standard Domain Name System. New components can thus be added dynamically, with the same scalability as DNS. Failings of the network or other agents are handled by alternative scenarios and only affect a single message. This robustness is due to the task based behaviour of an agent and the usage of one mobile agent per message.

4.2 Profiling

For the profile two main alternatives were implemented. The first is a rule-based one, somewhat similar to the `procmail` program for e-mail [1]. This allows users to have full control, but lacks self-learning capabilities. Feedback given by the user is used to create statistics, so the user can evaluate and improve rules.

This rule-based profile is used to proof the concept of cross-media profiling based on meta-data. As meta-data in the prototype three distinctive types of meta-data are used. The senders device-address is changed to a real name, the relation of the sender with the recipient is determined and a keyword is assigned to the message based on its content. An evaluation with this profile shows that messages can be classified based on their meta-data, rather than their content. Furthermore, it is shown that the usage of more extractions, resulting in more available meta-data, can provide a more fine-grained classification. When the same meta-data is available for different media, profiling originally designed for one medium can directly be applied to another.

Another created profile is based on the k -nearest-neighbour algorithm. The distance in this profile is based on comparing meta-data, using the same meta-data as the static rule profile. As function the number of equal meta-data fields present in both compared messages is used. Fields missing in one of the messages are ignored, otherwise additional extraction (more available information) can have a negative influence. Due to object-oriented programming, each type of meta-data can have its own distance function, to overcome the heterogeneous nature of the learning problem. The k most recent user-classified messages with the highest score determine the set of situations in which the classified message is interesting.

Some functional tests were conducted to evaluate the properties of the agent-based design. One of these tested the capabilities of the system to learn the user's preferences. For this purpose a user was simulated, to behave inconsistent (providing feedback not representing the intended interests) with a certain chance. For an user that responded consistent for all messages, a profile can be learned across multiple media. As shown in Figure 3, the more inconsistent the user responded, the more incorrect classifications the learning profile made. Note that the shown numbers do not represent a quantitative analysis.

5 Conclusions

In this paper a multi- and mobile-agent design is described to handle unified messaging in a personalised manner. A proof-of-concept prototype has been imple-

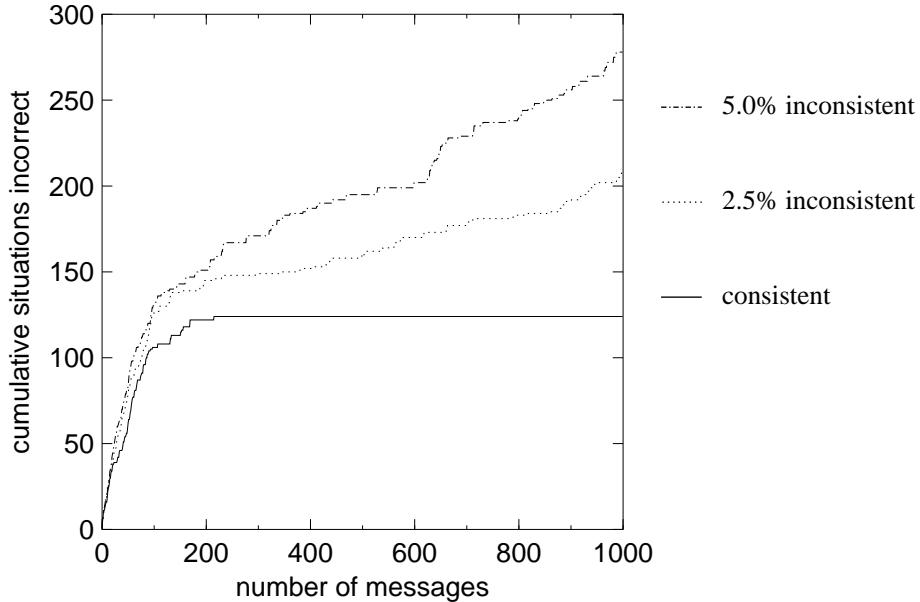


Figure 3: k -nearest-neighbour error for three different user consistencies.

mented, demonstrating the concept. This prototype was used as basis for an evaluation of the properties of the designed architecture.

First of all, automated transformation of different media can easily be accomplished. New devices can be added to the system without the requirement of modifications to existing devices. Messages from or for these new devices are converted to an available medium, so the user will receive the information rather than the data.

Although the messages in unified messaging are from all kinds of media, automated filtering to reduce a communication overload is possible. Each message is annotated with meta-data, describing characteristics of the message and the sender with relation to the addressee. Annotation is handled by a separate agent that can perform media-specific operations. This meta-data is used to classify new messages independent of the used medium. Adding new extractions can improve the granularity of the classification as well as filtering across different media.

The creation of a profile of the user with machine learning is investigated as well. Although further research is necessary to determine additional usable extractions of meta-data and improved profiling algorithms, learning user preferences based on this meta-data is shown to be achievable. The problem is marked down to its characteristics, and a possible solution is found in a simple variant of the k -nearest-neighbour algorithm. Easy integration and evaluation of an algorithm and application of new extractions is possible in the developed architecture.

Agents have proven to be good facilitators for personalised unified messaging. Each user can have its own cluster of agents, allowing adaptation of behaviour per functionality. Due to the routing of a message by its own mobile agent, the process a message is subjected to is flexible and robust. Individual messages can be handled differently this way, based on the addressee, location or origin. One of the main advantages is that each user can not only have its own profile, but apply an entirely different profiling method as well. A single algorithm that best fits all users is thus no longer necessary. Other benefits of the usage of agents are the distributed nature, allowing dynamic extensions and adaptations. Specific details of devices are encapsulated by the agent from the rest of the system as well.

References

- [1] S. R. van den Berg and P. A. Guenther. Procmail - autonomous mail processor. WWW site and program, 2002. <http://www.procmail.org/>.
- [2] D. Chess, C. Harisson, and A. Kershenbaum. Mobile agents: Are they a good idea? Technical Report RC 19887, IBM, December 1994.
- [3] U. Hanani, B. Shapira, and P. Shoval. Information filtering: An overview of issues, research and systems. *User Modelling and User Adaptive Interaction*, 11:203–259, 2001.
- [4] P. Helmersen et al. Impacts of information overload. Technical Report P947, Eurescom, January 2001. <http://www.eurescom.de/public/projects/P900-series/P947/>.
- [5] B. Raman, R. H. Katz, and A. D. Joseph. Universal inbox: Providing extensible personal mobility and service mobility in an integrated communication network. In *Proc. of the Workshop on Mobile Computing Systems and Applications (WMCSA'00)*, December 2000.
- [6] R. M. Schaar. Agent technology for personalised unified messaging. Master's thesis, Delft University of Technology, June 2002.
- [7] R. B. Segal and J. O. Kephart. Incremental learning in SwiftFile. In *Proceedings of the Seventh International Conference on Machine Learning*, June 2000.
- [8] Tryllian. *Tryllian ADK developers guide*, 2002. <http://www.tryllian.com/>.
- [9] Tryllian. Tryllian Agent Development Kit. Software Development Kit, 2002. <http://www.tryllian.com/>.
- [10] S. Whittaker and C. Sidner. E-mail overload: Exploring personal information management of e-mail. In *Conference proceedings on Human factors in computing systems*, pages 276–283, 1996.