Reachability Management

in a networked home

Master Thesis

by Johan Ensing

November, 2002

Delft University of Technology
Faculty of Information systems and Technology
Department of Mediamatics, Knowledge based systems

Title:	Reachability Management in a networked home
Author:	Johan Ensing

Supervision:	Prof. Dr. H. Koppelaar	- Delft University of Technology
	Drs. Dr. L.J.M. Rothkrantz	- Delft University of Technology
	Prof. Dr. Ir. E.J.H. Kerckhofs	- Delft University of Technology
	Ir. M.H. Verberkt	- Philips Research

Keywords: Reachability Management, Context awareness, Ambient Intelligence

Abstract: The topic of this graduation project is Reachability Management. The need for Reachability Management research results from the progress in (communication) technology that enables people to reach each other at all times, using all kinds of communication, no matter where people are. The drawback of this new technology is that you might receive communication when you don't want to be disturbed by communication, e.g. if your phone rings when you are sleeping. Reachability Management must give a person more control over her reachability.

This graduation project determines the important aspects of reachability and Reachability Management, and presents a model that a Reachability Management System (RMS) can use to determine whether someone is available for communication. This graduation project focuses on the architecture of a RMS. The main characteristics of this architecture are:

- It defines the functionality and the operation of the components that are necessary to obtain the functionality that a RMS must provide. The main functionality that a RMS must provide is the notification of incoming communication, monitoring of the progress of the notification, negotiation with the initiator of the communication, and starting communication.
- It takes into account that the RMS might need to execute multiple tasks at the same time. This is in particular of importance for the User Interface (UI), e.g. the RMS might show one UI at multiple devices, and it might show multiple UIs at one device. For example, at one device it might show a UI for the notification of new email and a UI for the notification of incoming telephone communication.
- It shows how the components of the RMS interact with the WWICE system. The WWICE system is developed at Philips Research. Among other things, the WWICE system deals with issues concerning distributed computing in a home environment.

PREFACE

Part of the education to become a Master of Computer Science at Delft University of Technology is to perform, report, and present a research project. My major is Knowledge Based Systems; the department headed by Prof. Dr. H. Koppelaar.

I did my graduation project at Philips Research, Eindhoven, the Netherlands. I worked in the Media Interaction group under supervision of Mark Verberkt. I would like to thank Mark Verberkt for his valuable feedback. I would also like to thank my direct supervisor at Delft University of Technology, Leon Rothkrantz, for his feedback and enthusiasm.

As part of my graduation project, I have interviewed eleven people. The interviews have helped a lot to determine when people are available for a particular type of communication, and to determine what functionality people expect from a Reachability Management System. I would like to thank all these people for their feedback: mrs. A. Matthijssen, mr. E. den Boef, mr. J. Hu, mr. J. Aerts, mr. C. Bartneck, mr. M. Ferreira, mr. B. Vaessens, mr. J. Kleinhout, mr. M. Sommeijer, mr. H. Ensing, and mrs. J. Ensing.

During my graduation project, especially the last weeks of the project, I have been quite busy and I was possibly not the best company one could have. Therefore, I would like to thank my roommates and above all my girlfriend, Wieteke Conen, for their support.

Johan Ensing
Eindhoven, November 2002

CONTENTS

1 INTRODUCTION

This chapter explains and defines my graduation project that is carried out within the Media Interaction group at Philips Research. Section 1.1 explains the research problem that the project deals with. Section 1.2 explains the relevance of this problem in more detail. Section 1.3 states the objectives of this graduation project, which are refined in section 1.4 that explains the scope. Section 1.5 explains my approach to this graduation project and section 1.6 concludes with the outline of this thesis.

1.1 PROBLEM DESCRIPTION

The topic of this graduation project is Reachability Management. The reachability of a person indicates what types of communication can be used to communicate with a person, and to what extent a person is willing to start a particular type of communication (e.g. telephone communication, video communication, or email). In other words, the reachability of a person indicates whether it is technically possible to start communication with a person (e.g. there must be enough bandwidth available to start video communication), and whether she[1] is available for communication.

Figure 1 Logo Media Interaction group

I have defined a paradigm, ‘Reachability Management‘, to indicate the control of a person over her reachability. Reachability Management research investigates the important aspects of reachability and the functionality that systems must have to deal with reachability. A system that manages the reachability of a person is called a Reachability Management System (RMS). An important aspect of reachability is the context information that is of importance to determine one’s reachability (e.g. the activities and the location of people)[2]. Other important aspects are privacy (e.g. should a RMS inform a caller that the callee is having a holiday), and trust (will people trust a RMS to manage their communication). However, the latter two aspects are social consequences of using a RMS that are not in the scope of this project.
The need for Reachability Management research results from the progress in (communication) technology that enables people to reach each other at all times, using all kinds of communication (e.g. video communication, telephone communication, and email), no matter where people are. The drawback of this new technology is that you might receive communication when you don’t want to be disturbed by communication, e.g. if your mobile phone rings during a meeting. Reachability Management must solve this problem. More specific, the central research question of this thesis is:

“ What are important aspects of reachability and how can an ambient system take reachability into account? ”

The aim of this graduation project is to determine the important aspects of reachability and Reachability Management, to design a RMS, and to integrate the RMS into an ambient system. An ambient system is the name for a system that consists of multiple interconnected devices, is integrated into its environment, is adaptive to the user, and might even show emotion[3]. On the one hand, the RMS must make sure that a person is not disturbed by communication, and on the other hand the RMS must take care that the person is aware of communication in time. In other words, the system must notify a person of incoming communication at the right time, in the right way (if it must notify a person at all).

[1] This report uses ‘she’ to refer to a person. Of course, a person can also be male.
[2] The definition of context is explained in more detail in the preliminary study of this graduation project.
[3] Section 1.2.1 describes ambient systems in more detail.

Figure 2 Ambient Intelligence: ubiquity, transparency, and intelligence

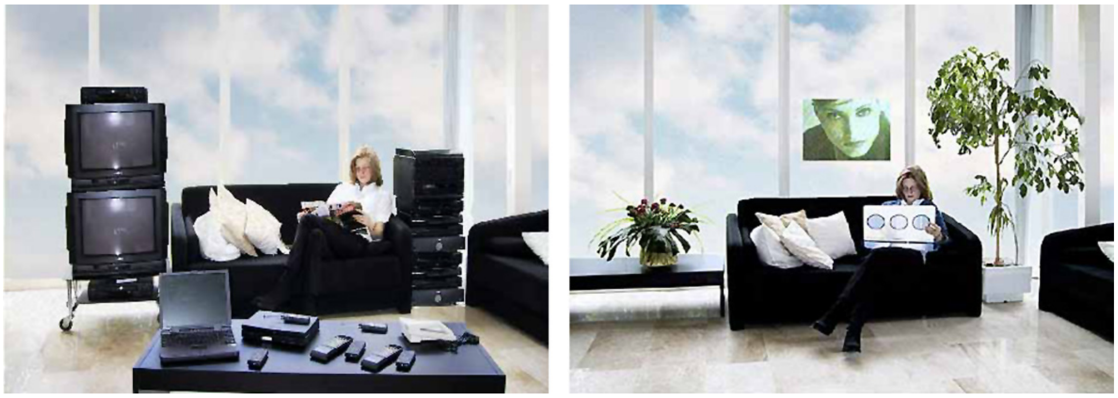


Figure 3 Present and future situation according to the Ambient Intelligence vision

1.2 RELEVANCE

This section explains the relevance of this graduation project. Section 1.2.1 explains the Ambient Intelligence vision that stimulates Reachability Management research, section 1.2.2 explains Reachability Management in more detail, and section 1.2.3 explains the link between Reachability Management and Knowledge Based Systems.

1.2.1 Ambient Intelligence

Ambient Intelligence is one of the visions that guide the research at Philips Research. According to [1], [2], and [3], there are two major developments in the computing and user interface industry that led to the emergence of Ambient Intelligence: Ubiquitous Computing and Intelligent Social Interfaces. Ubiquitous Computing envisions a world in which a user is surrounded by a large number of 'embedded devices' that are connected to each other. These embedded devices will probably not look like the personal computer we know now, but they can be small devices, like a PDA, connected through a wireless network, or a physical object with a built-in processor. Intelligent Social Interfaces research investigates user interfaces. It investigates user interfaces not only from a cognitive oriented approach (button dimension, coding and layout, display layouts and interaction schematics), but it also incorporates natural interaction, intelligence and emotion into user interfaces.

The Ambient Intelligence vision emerges from the convergence of these two developments. Ambient Intelligence research investigates systems that respond to and even anticipate our needs according to our tone of voice, gestures and expressions. These systems are called ambient systems. Ambient Intelligence is characterised by ubiquity, transparency and intelligence. Ubiquity because the user is surrounded by a multitude of interconnected embedded systems, transparency because these systems are invisible and moved into the background of the user's surroundings, and intelligence because the system is able to recognise the inhabitants, adapts itself to them, learns from their behaviour, and even shows emotion. Figure 2 shows an example of ambient intelligence: a child interacting with her environment, using only gestures and speech.

There are two important stimuli for Ambient Intelligence research. The first one is illustrated by Moore's law that predicts that the number of transistors that can be put on a chip doubles every eighteen months and thus the embedded computing power in our homes looks set to continue increasing exponentially [2]. The increase of embedded computer power enables the 'ubiquity'- and 'transparency'-concept of Ambient Intelligence. Another important stimulus is the technological progress in sensor technology. Sensors have become cheaper, smaller, more diverse, and easier to use. As stated in the 1997 ten-year Forecast of the Institute of the Future (quote): "One can build an accelerometer on a single chip for a couple of dollars, creating a device that is not only cheaper than today's sensors, but also smarter and more reliable" [4]. Figure 4 shows that sensors are the enabling technology of the next decade, according to the Institute of the Future.

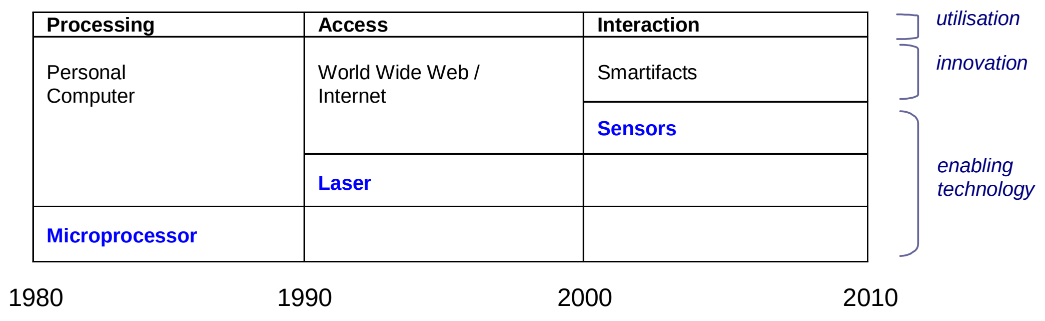


Figure 4 Sensors: the Next wave of Infotech Innovation [4]

Sensors are an enabling technology for the ‘Intelligence’-concept of Ambient Intelligence. For example, sensors can be used to recognise people or to determine the location of people or devices. The part of the Ambient Intelligence research that investigates how sensor technology can make interacting with computers easier is called Context Awareness research[4]. Context Awareness is an important research topic within the Ambient Intelligence research at Philips research since it provides the information that an ambient system needs for more intelligent behaviour.

1.2.2 Reachability Management

The popularity of mobile phones shows that people really enjoy and even start to rely on the ability to communicate whenever they want and wherever they are. The technology that is being developed as part of the Ambient Intelligence research, enables people to reach each other at all times, using all kinds of communication (e.g. video communication, telephone calls, and email), no matter where people are. However, it turns out that people want to have more control over their reachability since it is not *always* pleasant to be reachable. Philips Research wants to gain more insight in the important aspects of reachability, and in how an ambient system can take reachability into account. My definition of reachability is as follows:

“ The reachability of a person at a certain point in time, indicates what types of communication can be used to communicate with that person, and to what extent that person is willing to start a particular type of communication “

In other words, the reachability of a person indicates which types of communication *can* be used to reach that person, and for which types of communication the person is available. My definition of Reachability Management is as follows:

“ Reachability Management is the control of a person over her reachability. ”

An ambient system that takes care of one’s reachability is called a Reachability Management System (RMS). An example of the functionality of a RMS is to reject a communication request when it determines that it is not possible to start communication, or when it determines that the recipient is not available for communication. In this situation the system might propose other communication to the initiator of the communication (e.g. it might propose voicemail instead of telephone communication). If person A indicates that she wants to start communication with person B, person A is called the initiator of the communication and person B is called the recipient of the communication.

1.2.3 Reachability Management and Knowledge Based Systems

This section explains the link between Reachability Management and my mayor, Knowledge Based Systems (KBS). Knowledge technology models knowledge, develops artificial intelligence techniques (e.g. expert systems and neural networks), and uses them in decision problems such as regulating (technical) processes based on sensor information [5]. The KBS research that is performed at Delft University of Technology includes the application of knowledge technology to design the operation of airplanes, data interpretation techniques, multi-modal dialogs and interfaces, and emotional intelligence.

Figure 5 Cockpit environment of FlightGear[5]

[4] Context Awareness research is explained in more detail in the preliminary study of this graduation project.
[5] FlightGear is an open-source flight simulator.

Figure 5 shows an example of the research that is performed at Delft University of Technology. It shows the cockpit environment of a context aware system that is currently being developed for a flight simulator environment.

Reachability Management research uses knowledge technology to build a Reachability Management System. For example, multi-modal dialogs and interfaces, and emotional intelligence are of utmost importance to a RMS to ensure natural and intelligent interaction with the RMS. However, these topics are not in the scope of this graduation project. Nevertheless there is still a strong link between the issues that are dealt with by this graduation project and the field of Knowledge Based Systems. The goal of this graduation project is to determine the *desired functionality* of a Reachability Management System, to determine *the decisions* that the system must make to provide the user with this functionality, and to determine what *(context) information* is needed to resolve these decisions. These issues all have a strong link with the research area of Knowledge Based Systems, since knowledge technology is necessary to model and resolve the decision problems.

1.3 OBJECTIVES

This section states the objectives of this graduation project. As is mentioned in section 1.1, the aim of this graduation project is to determine the important aspects of reachability, and to develop a Reachability Management System that integrates Reachability Management into an ambient system. More concrete, the objectives of this graduation project are:

- Determine the important aspects of Reachability Management with respect to all types of communication (e.g. telephone-, video-, and email communication).
- Identify and prioritize the desired functionality of a Reachability Management System.
- Design the architecture of a RMS that shows which components are necessary to obtain the desired functionality. This design must show the decisions that these components should make to obtain the desired functionality, and the information and knowledge that is required to make these decisions.
- Design the architecture of a RMS such that the RMS can be incorporated into the WWICE system[6]. This means that the design must show how the components of the RMS interact with the entities of the WWICE system.

1.4 SCOPE

This section refines the objectives that are stated in section 1.3.

§ ***The social consequences of using a RMS are not in the scope of this project***
Some important social consequences are the privacy of a person (e.g. what information that the RMS must provide to an initiator), and whether a person trusts a RMS to manage her communication. These aspects are of importance when one wants to employ a RMS. However, one must first develop a prototype to test these aspects. Therefore, the focus of this graduation project is on a model that enables a person to indicate whether she is available for communication, and the architecture of a RMS.

§ ***The functionality of the RMS must focus on the in-home environment.***
The RMS must be incorporated in the WWICE system. The WWICE project explores new applications in the electronic market that will offer more entertainment, comfort, and flexibility *in the home*. This means that the RMS does not need to provide any functionality that would be desired in an office environment, such as making an appointment with one of the users of the WWICE system. Obviously, it must be possible to add new functionality (such as an appointment-system) to the RMS.

[6] The WWICE system is an ambient system that is developed as part of the Ambient Intelligence research at Philips Research. Chapter 5 explains the WWICE system in detail.

§ ***The model for reachability should be simple.***
The WWICE project focuses on the *consumer* domain. This means that average consumers must be able to use the RMS. Therefore, the conceptual model for one’s reachability must be easy to understand.

§ ***The RMS must adapt to the user behavior***
The RMS must learn from, and adapt to user behavior. The design of the system must take learning into account. However, learning does not have to be worked out in detail. This means that the design must show how learning could be incorporated in the architecture of the RMS, but the exact implementation (*how* the system learns from the user behavior) is not in the scope.

§ ***The functionality of the RMS must not become WWICE-specific.***
The RMS must be incorporated in the WWICE system. This means that the RMS can use all services that are part of the WWICE system. However, the *functionality* of the RMS must be independent from the WWICE system. This means that the focus of the system architecture must be on the desired functionality with respect to Reachability Management.

§ ***Initiator or recipient of communication might not have a RMS.***
The system architecture of the RMS must deal with the situation that both the initiator and the recipient of the communication have a RMS, and with the situation that only the recipient or the initiator has a RMS. The part of the architecture that explains the interaction with the WWICE system does not have to deal with the situation that either the recipient or the initiator does not have a RMS.

§ ***The RMS must work in a networked environment.***
The WWICE system works in a network of interconnected devices. The design of the RMS must take this into account. For example, it must take into account that part(s) of the network might be disconnected and that the available bandwidth is not unlimited.

§ ***The RMS must work in a Consumer Electronics environment.***
The WWICE project focuses on the *consumer electronics* domain. This means that there is no unlimited processing power and memory. The design of the RMS must take technical limitations into account.

§ ***Multiple (simultaneous) users in the home***
The RMS must deal with the situation that multiple users communicate at the same time. This is of importance with respect to the distribution of the (functionality of the) RMS over the network.

§ ***The WWICE system can provide user- and device location.***
The location of devices and users are important aspects of one’s reachability. The Presence Monitor service is the part of the WWICE system that keeps track of the physical locations of persons and devices that are in the home. The Presence Monitor service does not know the location with 100 % certainty: it appoints a certainty factor to every location of a device or person. The RMS can obtain the user and device location with the corresponding certainty-factor from the Presence Monitor.

§ ***The user activity is known to some extent.***
An aspect of one’s reachability is the activities that one is involved in. Some examples of activities are ‘watching television’, ‘having a shower’, ‘sleeping’, and ’having dinner’. In this graduation project we assume that the WWICE system has an Activity system that keeps track of the current activities of the users of the WWICE system. The design of the RMS must define the output to and input from the Activity system.

§ ***The user interface of the RMS is not in the scope.***
The design of the User Interface (UI) of the RMS system is not part of this graduation project. Examples of the design-issues of a UI are the size and the color of a window, and whether the system should use a Graphical User Interface (GUI), or speech (recognition) to interact

with the user. Obviously, it *is* important to identify what aspects of a UI are important with respect to Reachability Management.

1.5 APPROACH

My approach to this graduation project is as follows. First I have investigated the research that has a link with Reachability Management (Appendix I). From this research, I have extracted all information that is of importance to Reachability Management. Furthermore, I have determined new information that is of importance to Reachability Management. After this analysis, I have created a model for Reachability Management that describes how a RMS can help its' users to control their reachability. This model is based upon discussions with my supervisors, some interviews (Appendix II), and my own opinion. Next, I have developed the system architecture that a RMS needs to implement the model for Reachability Management. Since the system architecture must be incorporated into the WWICE system, I have investigated the WWICE system. I have determined the operation of the WWICE system based upon the available documentation and discussions with my supervisor at Philips Research. Based upon my analysis of the WWICE system, I have extended the system architecture of the RMS such that it can be incorporated into the WWICE system.

1.6 OUTLINE

Chapter 1 explains and defines my graduation project that is carried out within the Media Interaction group at Philips Research. It explains the problem that the project deals with, the relevance of this problem, the objectives of my graduation project, and the scope. Chapter 2 explains which information is of importance to Reachability Management and what functionality a RMS must have. Chapter 3 explains the model for Reachability Management. This chapter explains how the RMS determines whether it should notify a recipient of incoming communication, how it notifies the recipient, and what actions it undertakes if it determines that it should not notify the recipient. Chapter 4 explains the system architecture of the RMS. This chapter explains what components a RMS needs to deal with a communication request, and how these components interact with each other. Chapter 5 explains the WWICE system. It introduces the goals and the functionality of the WWICE system, and it deals with the entities of the WWICE system that are of importance to the RMS. Chapter 6 explains how the RMS can be incorporated in the WWICE system. Finally, chapter 7 states my conclusions and recommendations with respect to Reachability Management.

2 ANALYSIS

This chapter explains the aspects of Reachability Management that are identified in this graduation project. These aspects are based upon a literature survey (Appendix I), discussions with my supervisors, my own opinion, and some interviews (Appendix II). Section 2.1 describes a scenario that introduces some of the problems that a RMS must deal with. The reachability of a person indicates whether it is *possible* to reach that person using a particular type of communication, and whether that person *is willing* to use this type of communication. Section 2.1 explains what information is necessary to determine whether it is technically *possible* to start a particular type of communication. Section 2.3 explains what information is of importance to determine whether someone is available for communication. Note that these sections only explain *what* information is of importance. Chapter 3 explains the *relationship* between these pieces of information. Section 2.4 explains the functionality of the RMS that is worked out in detail in this graduation project.

2.1 SCENARIO

The main functionality of a RMS is on the one hand, to make sure that a person is not disturbed by communication, and on the other hand, to take care that the person is aware of communication in time. In other words, the system must notify a person of incoming communication at the right time, in the right way (if it must notify the recipient at all). The RMS must work as part of a home system (e.g. the WWICE system[7]) that connects all devices in the home. Via this home system the RMS can start communication or notify a recipient of incoming communication, using any device that is part of the home system (if the necessary resources are available, e.g. enough bandwidth and enough User Interface capabilities).

In this scenario, an initiator indicates to her RMS that she wants to start a telephone conversation with a recipient. Therefore, the RMS of the initiator sends a so-called communication request to the RMS of the recipient. This communication request indicates the type of communication that the initiator wants to start, the identity of the recipient and the initiator, and some additional information (e.g. the subject of the call). The RMS of the recipient must determine whether it is, at that moment, technically possible to set-up a telephone connection between the device that the initiator is using, and a device in the home system of the recipient (preferably a device near the recipient). Furthermore, it must determine whether the recipient is willing to start a telephone conversation with the initiator. For example, if the recipient is sleeping, or if she is already engaged in a telephone conversation, she might not be available for the call. If the RMS determines that it is technically possible to start telephone communication, and the recipient is available for the call, it determines that it should notify the recipient of the incoming call. Now the recipient can indicate whether or not she wants to accept the call. If the RMS determines that it should not notify the recipient, or the recipient does not accept the call, it must execute an appropriate action. For example, it might start interaction with the initiator to negotiate another type of communication, e.g. it might propose to record a voicemail message. When the initiator and the RMS of the recipient are negotiating, the initiator might increase the importance of the call (e.g. by indicating a priority-value). This might cause the RMS to notify the recipient (again). However, sometimes it will not be possible to start communication (e.g. because the recipient is not present in the house). In this situation, the RMS might forward important communication to another communication channel (e.g. to the mobile phone of the recipient).
It is clear that there are many aspects that influence the behavior of the RMS. This chapter explains the aspects that are of importance to determine whether it is technically possible to start communication, and to determine whether a recipient is available for communication. Furthermore, this chapter explains the functionality of a RMS that is worked out in detail in this graduation project.

[7] Chapter 5 explains the WWICE system in detail.

2.2 DETERMINE WHETHER IT IS POSSIBLE TO START COMMUNICATION

This section explains what aspects are of importance to determine whether it is technically possible to start a particular type of communication, e.g. to set-up a telephone connection between two devices. The research that is described in Appendix I identifies the following aspects:

- Type of communication.
- Location of initiator, recipient, and devices.
- Type of device.
- Network.

The type of communication determines the requirements of the communication, e.g. video communication needs a screen to display the video stream. The RMS determines the location of the initiator and the recipient (if possible). It uses these locations to determine which devices are nearby the recipient and the initiator. The type of a device determines its capabilities, and thus whether it is possible to setup the desired type of communication using that device. For example, if the only device that is near the recipient is a plain telephone, it is not possible to setup video communication using that device (as shown in Figure 6). The communication applications that are installed on a particular device are also capabilities of that device. A communication application takes care of a particular type of communication, e.g. an application to send and read email messages. The network between the devices that can be used for communication, also influences the types of communication that the system can start (e.g. the available bandwidth and the network protocol).

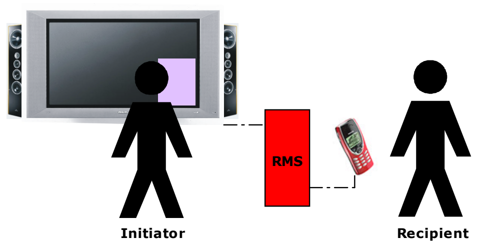


Figure 6 The RMS must determine whether communication is possible

The aspects that are enumerated above are all important aspects that determine to a large extent whether it is possible to setup a particular type of communication. However, there are two important aspects that are not mentioned by the research that is dealt with in the literature survey: the availability of capabilities, and the possibility to notify the recipient.

§ ***Availability of capabilities***

When a device has the capabilities to start a particular type of communication (e.g. it has a microphone and speakers to support telephone communication), this does not mean that it is possible to use that device. The resources that are needed to start the communication might not be available. For example, a screen in the living room might be used for a television application and thus it might not be possible to start video communication using that screen. Therefore, the availability of capabilities is important to determine which types of communication the RMS can start. The WWICE system already takes the availability of devices into account.

§ ***Possibility to notify the recipient***

The research groups that are involved in the research that is described in Appendix I, assume that a RMS should always set-up communication using a device near the recipient. However, the RMS might be able setup the communication using devices that are *not* near the recipient (e.g. using devices in another room). In this situation, the communication can proceed if the RMS is able to *notify* the recipient of incoming communication. The RMS might be able notify the recipient using devices near the recipient, but it might also use devices that are not near the recipient. In this last situation, the system might need to use an obtrusive UI to attract the recipient's attention, e.g. a loud sound. When the RMS notifies the recipient on a device that the recipient cannot use for the proposed type of communication, a new problem is introduced: the recipient might not know which devices she can use to start the communication. The RMS must somehow solve this problem. For example, it could show a blinking icon on every device that the recipient can use for the desired type of communication. Note that a person might want to receive a notification, even if the RMS cannot start the corresponding communication application. For example, a user might want to know that new email has arrived, even if it is not possible to start an email application at that moment.

2.3 DETERMINE WHETHER THE RECIPIENT IS AVAILABLE FOR COMMUNICATION

This section explains what information is of importance to determine whether a recipient is available for communication. The first eight subsections explain the aspects that are identified by the literature survey that is described in Appendix I. The next subsections explain new aspects.

§ ***Priority of communication***

The priority is the importance of communication. According to the research that is described in the literature survey, the recipient might indicate the priority of communication as follows:

- The assertion of urgency[8]. The initiator can indicate a certain degree of urgency.
- The specification of a function. The initiator can give details about the reason for her call, about her position, or even her qualification. For example, she may call as a member of a particular project or company.
- The specification of a subject.
- The provision of a reference. The initiator mentions the recommendation of a third person.
- The presentation of a voucher: The voucher differs from the reference in that the recipient has issued it herself. It may increase the chance of a return call.
- Offering a surety. The initiator may remit to the recipient an (possibly negotiated) amount as a surety. If the recipient does not agree with the initiator's evaluation of the urgency of his call, she has the potential to withhold this amount.

I distinguish two types of priority: the priority from the initiator's point of view (priority^{i}), and the priority from the recipient's point of view (priority^{r}). The initiator can indicate the priority^{i} by allocating a priority-value to a message. The priority^{i} and the priority^{r} can have the following values: 'Normal', 'High', and 'Emergency'[9]. The recipient can indicate preferences that determine the priority^{r}. From interviews (Appendix II) results that people want to indicate preferences with respect to the following aspects:

- Priority^{i}. The priority^{i} might be used as the default value for the priority^{r}.
- Identity of the initiator. This information might pose upper or lower bounds to the priority^{r}, e.g. your girlfriend has at least the priority^{r} "High" and your mother in law has at most the priority^{r} "Normal".
- Subject of the message. This information might pose upper or lower bounds to the priority^{r}, e.g. the subject "WIN A FREE TELEVISION" has at most the priority^{r} 'Normal', and the subject "Meeting" has at least the priority^{r} "High".

The other aspects that are mentioned by the literature survey (specification of a function, provision of a reference, presentation of a voucher, and offering a surety) are aspects that merely increase the security of communication (can I trust the initiator, is the initiator who she says she is), and aspects that indicate the priority^{i} in another way (how much money is the initiator willing to pay). These aspects are merely of importance when one would actually want to employ a RMS. These aspects are not taken into account in this graduation project.

[8] One research group uses 'urgency' instead of 'priority'. I think that priority is a better name, since people are accustomed to indicate the importance of communication (e.g. email) by allocating a priority.
[9] Future research must determine what classification is the most convenient for people.

§ ***Status of the recipient***
A recipient can indicate her personal status. The status is the most important aspect of one's reachability: the status allows a user to control her reachability. A person must indicate which statuses she wants to have, how a particular status can become active, and what the RMS should do with incoming communication when a particular status is active. From user tests at Microsoft research results that users do not want to use many statuses [6]. Therefore, this graduation project uses the statuses 'Available', 'Busy', 'Working', and 'Not available'[10]. For example, the status 'Busy' might mean that a recipient is only available for communication that has a high priority[r].
Not only a person can activate a status, but also the RMS. For example, there might be a system that detects activities and these activities might cause the RMS to activate a certain status (e.g. during the activity 'sleeping' the status is 'Not available'). Furthermore, the system might activate a status during some period(s). For example, a person might want to have the status 'Not available' between 0:00h and 7:00h.

§ ***Type of communication***
Obviously, the type of communication is very important to determine whether or not a recipient is willing to start communication. Some examples of types of communication are 'email', 'SMS', 'direct messaging', 'telephone communication', and 'video communication'. The different types of communication are grouped into two classes: real-time communication and non real-time communication[11]. Real-time communication is communication whereby two persons have direct interaction. In other words, the interval of time between one person sending a message (asking a question) and the other person replying with a message (answering the question) is negligible. Some examples of real-time communication are telephone communication and video communication. An important property of real-time communication is that the initiator of the communication will not wait very long for a response to her communication request (e.g. one does not want to wait very long for someone to answer the phone). Non real-time communication is communication that is not real-time, for example email and SMS.

§ ***Time***
The time might influence the availability of the recipient in two ways. The recipient might indicate that during some periods, she is not available for particular types of communication. For example, she might indicate that her status is 'Busy' between 12:00h and 12:30h (e.g. because she is always having lunch between 12:00h and 12:30h). Furthermore, the elapse of time might influence the priority of communication[12]. For example, a user can indicate that the priority of an email message increases over time. This might cause the RMS to try different ways to notify the recipient (e.g. it might forward an email message to another email address).

§ ***Identity of the initiator***
The recipient might indicate that she *is* available for communication from some (groups of) initiators, and that she is *not* available for communication from other (groups of) initiators. For example, when your status is 'Working' you are available for communication from colleagues, but you might not want to be disturbed by your family. Of course, it must also be possible to put some initiators in an ignore-list (e.g. marketing companies).

§ ***Subject of the communication***
A recipient might have indicated that she is more (or less) available for communication with a particular subject. For example, she might be more available for communication with the subject 'meeting' or 'urgent', and she might be less available for communication with the subject 'Visit our homepage' (since this is typically the subject of junk mail). An initiator might indicate the subject of the communication, e.g. 'meeting', 'appointment', or 'social call'. However, the RMS might also automatically extract the subject, e.g. it might parse an email

[10] Future research must determine what classification is the most convenient for people. Of course, users must be able to create and define their own statuses.
[11] One could argue that there is a need for more classes, since direct messaging and chatting are neither real-time nor non real-time. To keep things simple, I distinguish only two classes. Direct messaging and chatting are assumed to be real-time communication since the communicating parties expect a reply in a short time.
[12] The priority of communication is explained below.

message to extract pre-defined subjects from this message. The influence of the subject on one's reachability might depend on, among other things, the status, the activities that one is involved in, and possibly even the identity of the initiator. For example, when your status is 'Working' you might indicate that you are available for communication if the subject is 'meeting'. However, from the interviews results that people do not want to indicate a lot of preferences. Furthermore, the system might become too complex if they must indicate a lot of preferences. To keep things simple, the model for Reachability Management that is explained in chapter 3 does not take many dependencies into consideration. This model assumes that the subject of communication only influences the priority[r] of communication. Of course, the implementation of the RMS must be flexible such that a user can easily indicate more dependencies.

§ ***Preferences***
The research that is described in Appendix I came up with User specific reachability rules and Common reachability rules. User specific reachability rules are indicated by the user and define how the RMS should deal with incoming communication requests. Common reachability rules are rules that the user cannot change. In this graduation project, the User specific reachability rules are called 'Preferences'. From the interviews results that users want to have control over their reachability. Therefore, there should not be any 'Common reachability rules'. However, there should definitely be a difference between 'normal' preferences that are easy to adjust, and 'expert' preferences for people who want to adjust many preferences. This is a UI-problem that is not in the scope of this project.
Unfortunately, the literature survey does not clarify which preferences a RMS should use. In other words, the research groups did not come up with a model for Reachability Management that describes *how* a RMS must determine whether a recipient is willing to start communication.

§ ***User Interface of the notification***
If a RMS determines that it is possible to start communication and that the recipient is available, it notifies the recipient. It is important to realize that the User Interface (UI) of the notification might influence whether or not the recipient is available for the communication. For example, the UI of the notification of new e-mail messages might be a small icon on the kitchen screen. A user is might not feel disturbed by such a notification (especially if she is not in the kitchen). Therefore, she might always be available for new e-mail communication. This means that the RMS must be able to adapt its' UI, based upon (among other things) the type of communication.

§ ***Identity of the recipient***
The identity of the recipient determines which preferences a RMS must use to determine whether the recipient is available for communication. The research that is described in the literature survey does not mention this aspect since this research assumes that every user has a RMS, and thus the RMS automatically uses the appropriate preferences. However, one RMS might also serve a group of users (e.g. there might be one RMS per home system). In this situation, the RMS needs the identity of the recipient to determine which preferences it must use.

§ ***Activities***
An ambient system might be able to automatically detect the activities that a user is involved in (e.g. sleeping, cooking, having a bath, having a shower, and watching a television program, having a telephone conversation). These activities might influence whether or not someone is available. For example, when a person is sleeping she might not be available for communica-tion. Furthermore, activities might influence the User Interface of the RMS. For example, the RMS might use speech recognition when the recipient is cooking.

§ ***Additional aspects***

This section introduced aspects that are of importance to determine whether or not a person is available for communication. These are the aspects that are used in this graduation project. Of course, one could come up with (many) other aspects, for example:

- The history. If an initiator repeatedly sends a communication request, the RMS might determine that the communication is important and thus that the recipient is willing to start that communication. This aspect is only of importance if the initiator does not have a RMS. If the initiator does have a RMS, she can indicate herself that the communication is important.
- The mood of the recipient. For example, if the RMS can detect that the recipient is tired, it might determine that the recipient is not available for communication.

The goal of this project is not to determine *all* aspects that determine whether someone is available for communication. The goal of this graduation project is to determine the most important aspects of reachability, and to design a RMS that integrates Reachability Management into an ambient system. This means that the focus of this project is on the system architecture of a RMS. When this architecture is implemented, it can be used to test the RMS with users. These user tests must determine which other aspects influence the reachability of a person.

2.4 FUNCTIONALITY REACHABILITY MANAGEMENT SYSTEM

This section explains the functionality of the RMS that is worked out in detail in this graduation project. It is definitely possible to come up with more functionality (especially when you include the office environment in the scope). However, the functionality that is described in this subsection already enables the user of the RMS to control her reachability to a large extent. The first six sections describe the functionality that is identified by the research that is described in Appendix I. The last four sections explain new functionality.

§ ***Reachability Management***

Obviously, the main task of the RMS is Reachability Management. The RMS must determine the reachability of a user, and based upon this reachability, it must determine whether it should notify a recipient of incoming communication. If it determines that it should not notify the recipient, it must execute appropriate actions, e.g. forward the communication. A user must be able to indicate preferences that determine how the RMS takes care of Reachability Management.

§ ***Notification***

On the one hand, the RMS must make sure that the user is not disturbed by communication. On the other hand, the RMS must take care that the recipient is aware of communication in time. In other words, the system must notify the recipient of incoming communication at the right time, in the right way. The RMS notifies the recipient to determine whether it should start the communication that the initiator wants to start (e.g. set-up a telephone connection).

§ ***Monitor progress***

The system must monitor the progress of the notification, and, if necessary, switch to alternative communication channels. For example, when the recipient is not aware of an email message with a high priority[r] for twenty-four hours, the system might decide to transform the message to SMS, and forward it to the mobile phone of the recipient.

§ ***Negotiation***

In order to determine whether it should notify the recipient, the RMS might need to negotiate with (the RMS of) the initiator. Furthermore, it must take care of any negotiation between the recipient and the initiator, e.g. the initiator might propose to start video communication and the recipient might propose to start telephone communication. An important issue that the RMS must consider is what options it must propose, e.g. the RMS might propose to an initiator to record a voicemail message.

§ ***Communication rejection***
The RMS might determine that it is not possible to start communication, or the recipient is not available for communication. In this situation it must execute an appropriate action, e.g. it might forward the communication (e.g. forward a phone call to the recipient's office), or it might propose other actions to the initiator (e.g. propose e-mail communication instead of telephone communication).

§ ***Transformation of communication***
The system must allow the user to use *every* device for communication (if possible). Therefore, the system must take care of the transformation from one type of communication into another. For example, it might transform an e-mail message into a SMS message such that a user can read the message on via her mobile phone. This graduation project does not explain *how* the RMS can transform communication. It only indicates how the system architecture of the RMS can take transformation into account.

§ ***Adapt information***
A RMS must adapt the information that it provides to a recipient or an initiator. For example, when the RMS determines that the recipient is not reachable, the RMS might inform the initiator of the location of the recipient, or it might tell when the recipient is likely to be available again. The RMS must adapt the information that it provides based upon the preferences of the user.

§ ***Adapt the User Interface of the notification***
The RMS must adjust the UI of the notification of incoming communication. There are many ways that the RMS can adjust the UI. For example, it could adjust the size of icons, or use an animated character instead of a text-based interface. This graduation project only deals with the aspects of a UI that influence the reachability of the recipient:

- The devices that the RMS uses for the UI. For example, the RMS might create a UI near the user when it has to deal with a telephone call, and it might create a UI at the kitchen screen when it has to deal with a new e-mail message.
- The obtrusiveness of a notification. The obtrusiveness of a UI indicates to what extent the UI stands out. For example, the RMS can create an obtrusive UI if the recipient is not near any devices (e.g. use a loud sound if the recipient is in the cellar).

§ ***Learning***
The RMS might learn from and adapt to the user behavior. The system might learn implicitly (the system observes the behavior of the user) and/or explicitly (the user indicates an appreciation-value when the system executes automatic behavior). This project does not determine *what* data the RMS must store, or *how* the RMS could extract new preferences from this data, since this is not the focus of this graduation project. This project only indicates how learning can be incorporated in the architecture of a RMS.

§ ***Pro-actively indicate status to initiator***
The RMS can pro-actively indicate the status of the user to the user herself, and to initiators. This way, the RMS would have MSN® Messenger[13]- or ICQ[14]-like functionality: the initiator can see the status of a user and thus she can guess whether or not the user is available for communication, even before she tries to start communication. If a user can see her own status, she can adjust her status if the current status is not correct. This functionality is definitely a nice function that would increase the usability of a RMS. However, it does not influence one's reachability and thus it is not worked out in detail in this graduation project.

[13] For more information concerning MSN ® Messenger, see http://messenger.msn.com/
[14] For more information concerning ICQ, see http://web.icq.com/

3 REACHABILITY MANAGEMENT

The main functionality of the RMS is Reachability Management. The RMS must enable its users to control their reachability. On the one hand, the RMS must make sure that the user is not disturbed by communication. On the other hand, the RMS must take care that the recipient is aware of communication in time. In other words, the system must notify the recipient of incoming communication at the right time, in the right way. Section 3.1 explains how the RMS determines whether the recipient wants a notification of incoming communication. Two important aspects that determine whether she wants a notification, are whether she is available for the communication and whether the RMS can start that type of communication. These aspects are dealt with in respectively section 3.2 and section 3.3. Section 3.4 explains how the RMS notifies the recipient. If the RMS determines that it should not notify the recipient, the RMS must undertake an appropriate action. Section 3.5 explains what actions the RMS can undertake, and how the RMS determines the appropriate action.

3.1 DETERMINE WHETHER THE RECIPIENT WANTS A NOTIFICATION

This section explains how the RMS determines whether it should notify the user of incoming communication. Another word for incoming communication is a ‘communication request’. The reachability of a person at a certain point in time, indicates what types of communication can be used to communicate with that person, and to what extent that person is willing to start a particular type of communication. The RMS determines whether or not it should notify a user of a communication request, based upon the type of communication and the reachability of the user.

If the communication request regards real-time communication, the RMS determines that it should notify the user if the user is available for *a* type of real-time communication, and if it is possible to start that type of communication. If the recipient is available for a particular type of communication, and it is possible to start that type of communication if the recipient stops another application (e.g. she stops a television application), the RMS could also notify the recipient. However, the RMS must be incorporated in the WWICE system and, at this moment, the WWICE system cannot determine whether the necessary resources to start an application will become available when a person stops an application. Therefore, the RMS only notifies the recipient when it determines that there is a type of real-time communication that the recipient is willing to start, and it is possible to start that type of communication. This way, the RMS does not disturb the recipient for communication that it cannot start. Of course, a user must be able to indicate she wants a notification, even if it is not possible to start the corresponding communication application.

If the RMS determines that the recipient is not available for the type of real-time communication that the initiator proposes, but the recipient is available for a different type of real-time communication, the RMS still notifies the recipient (if it is possible to start that type of communication). The RMS could also decide to propose this type of communication to the initiator. However, the initiator usually does not really care what type of real-time communication she starts, thus the RMS notifies the recipient in order to find out if the recipient wants to start real-time communication[15].

From interviews (Appendix II) results that a recipient wants to know whether she has new messages, regardless of the fact whether she can start the corresponding communication application. Thus if the communication is not real-time, the RMS notifies the recipient if she is available for the communication. Of course, the recipient must be able to indicate that it must also be possible to start the corresponding communication application.

[15] If the recipient wants to start a different type of communication than the initiator proposes, the RMS first proposes this type of communication to the initiator, before it starts the communication.

3.2 DETERMINE WHETHER THE RECIPIENT IS AVAILABLE FOR COMMUNICATION

This section explains how the RMS determines whether someone is available for a particular type of communication. Section 3.2.1 summarises this decision process and sections 3.2.2 to 3.2.4 explain the decision process in more detail. The decision processes that are explained in this section are based upon interviews (Appendix II), discussions with my supervisors, and my own opinion.

3.2.1 Overview

In a nutshell, the RMS determines as follows whether a recipient is available for a particular type of communication. First, the RMS determines the status of the recipient. The status indicates to what extent the recipient wants to be disturbed. Based upon the status, the RMS determines the threshold of communication. The threshold is the minimum priority^r that a particular type of communication must have such that the recipient is available for that communication. Next, the RMS determines the priority^r of the communication. The priority^r is the importance of the communication from the recipient's point of view. If the priority^r is higher than the threshold, the recipient is available for that type of communication. Otherwise, she is not available. Figure 7 illustrates the decision process.

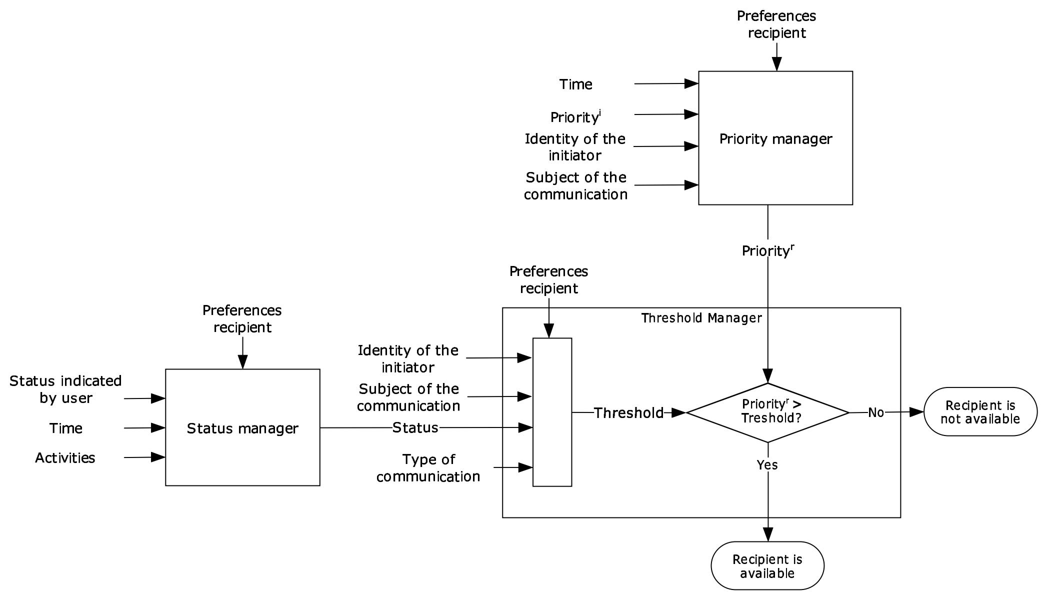


Figure 7 Decision process to determine whether recipient is available

3.2.2 Priority manager

An important aspect that determines whether someone is available for a particular type of communication is the importance of that communication from the recipient's point of view. This is called the priority^r. The Priority manager is the subsystem of the RMS that determines the priority^r. In this graduation project, the following classification for the priority^r is used: 'Normal', 'High', and 'Emergency'[16]. This classification is based upon the interviews that are explained in Appendix II.

[16] User tests must determine the best classification for the priority^r.

Figure 8 illustrates the information that the Priority manager uses to determine the priority^{r}[17].

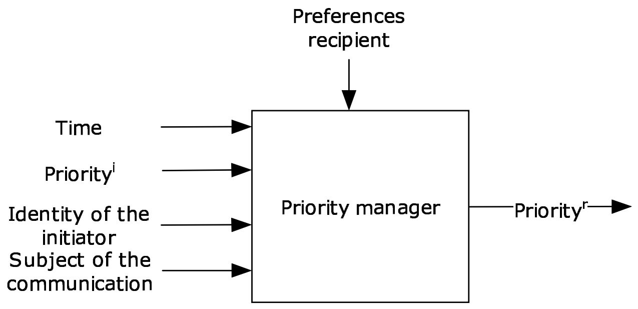


Figure 8 Information that determines the priority^{r}

The user preferences of the recipient determine how the Priority manager combines the information that is shown in Figure 8. Per default the Priority manager sets the priority^{r} to 'Normal'. Next, the Priority manager adjusts the priority^{r}, based upon the information that is shown in Figure 8. The Priority manager uses knowledge rules to adjust the default priority^{r}:

- Priority^{i}. The priority^{i} is the importance of the communication from the initiator's point of view. If the initiator has indicated a priority^{i}, this value is used as the default value for the priority^{r}.
- Identity of the initiator. This information might pose upper or lower bounds to the priority^{r}, e.g. your girlfriend has at least the priority^{r} 'High' and your mother in law has at most the priority^{r} 'Normal'.
- Subject of the communication. This information might also pose upper or lower bounds to the priority^{r}, e.g. the subject 'Meeting' has at least the priority^{r} 'High'.
- Time. Section 3.4 explains that the higher the priority^{r}, the more effort the RMS puts into notifying the recipient. Therefore, a recipient might indicate that the Priority manager must increase the priority^{r} over time. This way the recipient might be aware of the communication in time. For example, she might indicate that the Priority manager must increase the priority^{r} of communication to 'High', one day after the communication arrives at the RMS. Obviously, the recipient will only indicate this preference for non real-time communication. If the communication is real-time the initiator will not wait a day for the response of the recipient.

Of course, the Priority manager must contain a mechanism that detects inconsistencies in the preferences of the user, such that it can inform the user in advance whether her preferences cause conflicts. When conflicts occur due to other reasons, e.g. your girlfriend (lower bound "High") sends a message with subject "WIN 1 MILLION DOLLARS" (upper bound "Normal"), the Priority manager assigns the highest priority^{r} to the communication to ensure that the recipient does not miss important communication.

Another way to determine the priority^{r} is to allocate priorities and weights to all preferences. In this situation, every priority must have a number, e.g. 'Normal'=1, 'High'=2, and 'Emergency'=3. The weights are between 0 and 1. The Priority manager could use a formula like this: $Priority^{r} = CEILING\ (w1*p1 + w2*p2 + w3*p3) / (w1+w2+w3)$, where p1, p2, and p3 are priorities, and w1, w2, and w3 are weights. This formula determines the weighted average of the priorities, and it rounds up the result to the nearest integer. However, this formula increases the complexity of the algorithm and thus the user might not understand the working of the Priority manager anymore. Furthermore, the user has to indicate more preferences. From the interviews results that users prefer to have a rule-based algorithm. Therefore, the Priority manager just assigns the highest priority^{r} to the communication. Note that users do not *have* to indicate any preferences. Per default, the priority^{r} might be the priority^{i} of the incoming communication. However, it must be possible to indicate preferences such that (expert) users can have full control over their reachability. User tests must determine what preferences users want to indicate.

[17] Note that the type of communication does not influence the priority^{r} of communication. The Threshold manager takes the type of communication into account when it determines the threshold.

3.2.3 Status manager

The status is the concept that enables the user of the RMS to control her reachability. For example, if the status of the user is ‘Busy’ this might mean that she is not available for real-time communication. The Status manager is the subsystem of the RMS that determines the status. User tests that are performed at Microsoft research conclude that users do not want to use many statuses [6]. In this graduation project, the statuses ‘Available’, ‘Busy’, ‘Working’, and ‘Not available’ are used to explain how the concept ‘Status’ works. User tests must determine what classification is the most convenient for people. Of course, users must be able to create their own statuses. Figure 9 illustrates the information that influences the activation of statuses.

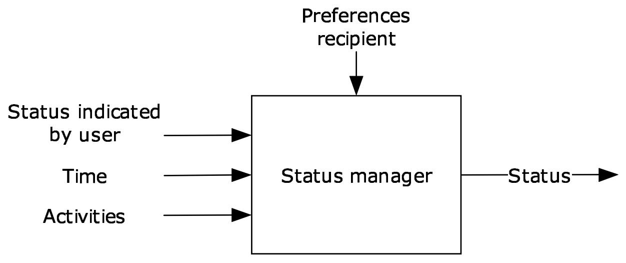


Figure 9 Information that might activate statuses

The preferences of the recipient determine how the Status manager combines the information that is shown in Figure 9. Per default, the Status manager sets the status to ‘Available’ (which means that the recipient is available for all communication). The Status manager adjusts the status, based upon the information that is shown in Figure 9:

- Status indicated by user. A user can indicate her status directly, using the UI of the RMS. For example, a user might indicate that for the next hour, her status is ‘Not available’.
- Time. The user might indicate in advance that she has a particular status during some periods, e.g. her status is always ‘Not available’ between 0:00h and 7:00h.
- Activities of the user. The user might indicate that she has a particular status during particular activities, e.g. her status is ‘Busy’ when she is in involved in the activity ‘Watching the news’, and her status is ‘Not available’ when her activity is ‘Sleeping’.

Of course, the Status manager must contain a mechanism that detects inconsistencies in the preferences of the user, such that it can inform the user when her preferences cause conflicts. The Threshold manager needs *one* status to determine the threshold. During a day, conflicts might occur because multiple statuses are activated at the same time. For example, the time is 0:30 (this means that the user’s status is ‘Not available’) and the user indicates, using the UI of the RMS, that her status is ‘Available’. In order to solve conflicts, the Status manager determines the ‘user status’. This is the status that the Threshold manager uses to determine the threshold. From the interviews results that the Status manager should determine the user status as follows: per default, the user status is ‘Available’. If the user indicates a status directly, then this is the user status. Otherwise, the Status manager uses the status that has the strongest impact on the user’s reachability. In other words, it uses the status that makes the user the least reachable. This implies that the user must indicate for each status the impact of that status on her reachability. For example, the user could order the statuses as follows: [1] Not available; [2] Working; [3] Busy; [4] Available.
This means that the status ‘Working’ is the user status, when both the status ‘Working’ and the status ‘Busy’ are activated, and the user did not indicate a status directly. The Status manager uses the status that has the strongest impact on one’s reachability since the user activates statuses to indicate that she does *not* want to be disturbed. Thus the status that makes sure that she is not disturbed, must become the user status. This way it is easy for the user to understand how the Status manager determines the user status.

Figure 10 shows an example of statuses that are directly activated by the user, e.g. she might indicate the status ‘Not available’ because she is tired, or the status ‘Available’ because she expects an important call.

		0:00h	1:00h	2:00h	3:00h	4:00h	5:00h	6:00h	7:00h	8:00h	9:00h	10:00h	11:00h	12:00h	13:00h	14:00h	15:00h	16:00h	1700h	18:00h	19:00h	20:00h	21:00h	22:00h	23:00h	24:00h
Status	Not available																		🟦	🟦						
	Working																									
	Busy																									
	Available																				🟦	🟦	🟦	🟦		

Figure 10 Example of the statuses that are activated by the user directly

Figure 11 shows an example of the periods that the Status manager might automatically activate a status because of user preferences that are based on the time.

		0:00h	1:00h	2:00h	3:00h	4:00h	5:00h	6:00h	7:00h	8:00h	9:00h	10:00h	11:00h	12:00h	13:00h	14:00h	15:00h	16:00h	1700h	18:00h	19:00h	20:00h	21:00h	22:00h	23:00h	24:00h
Status	Not available	🟥	🟥	🟥	🟥	🟥	🟥	🟥																	🟥	
	Working																									
	Busy														🟥											
	Available																									

Figure 11 Example of the statuses that are activated by the RMS due to the time

Figure 12 shows an example of statuses that are activated because the Status manager detects activities, e.g. the activity ‘Sleeping’, ‘Having breakfast’, ‘Having dinner’, or ‘Watching news’.

		0:00h	1:00h	2:00h	3:00h	4:00h	5:00h	6:00h	7:00h	8:00h	9:00h	10:00h	11:00h	12:00h	13:00h	14:00h	15:00h	16:00h	1700h	18:00h	19:00h	20:00h	21:00h	22:00h	23:00h	24:00h
Status	Not available	🟧	🟧	🟧	🟧	🟧	🟧																			
	Working																									
	Busy							🟧											🟧			🟧				
	Available																									

Figure 12 Example of the statuses that are activated by the RMS due to activities

Figure 13 combines Figure 10 to Figure 12 and shows the ‘user status’ that is based upon the preference rules that are explained above (e.g. a status that is directly indicated by the user has precedence).

		0:00h	1:00h	2:00h	3:00h	4:00h	5:00h	6:00h	7:00h	8:00h	9:00h	10:00h	11:00h	12:00h	13:00h	14:00h	15:00h	16:00h	1700h	18:00h	19:00h	20:00h	21:00h	22:00h	23:00h	24:00h
Status	Not available	🟪	🟪	🟪	🟪	🟪	🟪	🟪											🟪	🟪					🟪	
	Working																									
	Busy														🟪											
	Available								🟪	🟪	🟪	🟪	🟪	🟪		🟪	🟪	🟪			🟪	🟪	🟪	🟪		

Figure 13 Example of the ‘active status’

The Status manager could also determine the user status as follows: when there is a communication request, the Status manager could determine the threshold for every status, and use the status that causes the highest threshold. However, this would make the operation of the Status manager more complex and thus the user might not understand the working anymore. Furthermore, the Status manager cannot show the current user status to the user, since this status might be different for every communication request. Therefore, the Status manager uses the precedence rules that are explained above.

Users might want to be able to activate a status only for specific (groups of) initiators. For example, you might want to have the status ‘Not available’ for all your colleagues. However, this increases the complexity of the system, such that users might not understand the working anymore. Another disadvantage is that the Status manager cannot show the user status to the user, since this status might be different for every initiator. The user obtains the same system behavior when she creates a new status, and indicates preferences that adjust the threshold for that status for particular initiators. User tests must determine whether or not the user would like to have multiple active statuses that are valid for only some (groups of) initiators.

3.2.4 Threshold manager

The threshold is the minimum priority^r that communication must have such that the recipient is available for the communication. The Threshold manager determines the threshold and it determines, based upon the threshold and the priority^r, whether a recipient is available for a particular type of communication. If the priority^r is higher than the threshold, the recipient is available. Otherwise, she is not available. Figure 14 illustrates the information that influences this decision process.

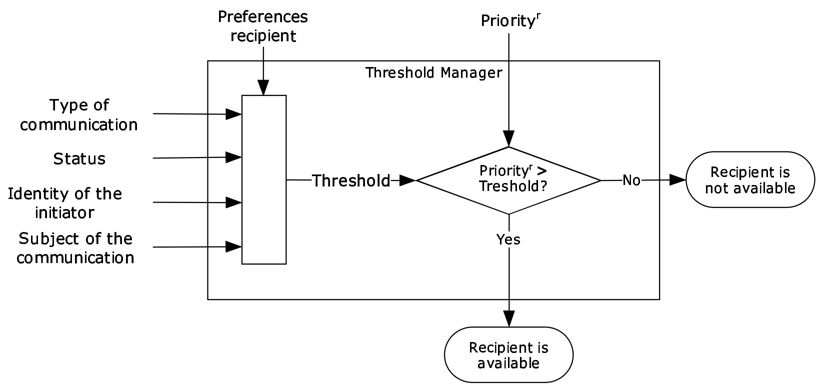


Figure 14 Information that determines the threshold

The activities that the recipient is involved in do not influence the threshold. One might think so since, for example, when the recipient is having a bath she might not be interested in video communication. However, the Threshold manager does not take activities into account, since activities already influence the status of the recipient. From interviews results that the default threshold value mainly depends on the type of communication and the status. Table 1 shows an example of preferences that the RMS uses to determine the threshold. Note that a user typically has the same preferences for all types of communication that belong to the same class of communication.

Table 1, Preferences table to determine the threshold

Status		**Available**	**Busy**	**Working**	**Not available**
Type of communication	**E-mail**	Normal[18]	Normal	Normal	Emergency
	SMS	Normal	Normal	Normal	Emergency
	Telephone	Normal	High	High	Emergency
	Video	Normal	High	High	Emergency

[18] This field indicates the threshold.

The Threshold manager determines the default threshold, based upon the user preferences with respect to the type of communication and the status (Table 1). Next, it adjusts the threshold, based upon preferences with respect to the status of the recipient in combination with the identity of the initiator and/or the subject of the communication. For example, a user might indicate that when her status is 'Working', the threshold for all communication from colleagues is 'Normal' (instead of 'High'). When conflicts occur, the Threshold manager uses the default threshold that is shown in Table 1.

The Threshold manager determines whether a recipient is (still) available for communication, every time it determines a new threshold, or when the Priority manager determines a new priority[r]. The Threshold manager determines a new threshold, every time the Status manager determines a new status.

3.3 DETERMINE WHETHER IT IS POSSIBLE TO START COMMUNICATION

The RMS must determine which types of communication it can start within the home system of the recipient, e.g. whether it can set-up a telephone connection. The RMS determines which devices it should use for a particular type of communication, and it investigates whether it can use these devices to start communication. Chapter 5 explains that the WWICE system can investigate whether it is possible to start a particular type of communication, using a particular device. This section explains how the RMS searches for the devices that the WWICE system must investigate. The search for devices that the RMS can use for communication is roughly the same as the search for devices that the RMS can use for notification. Therefore, the last paragraph of this section explains how the RMS searches for devices that it can use for notification.

The recipient must be able to indicate which device(s) she wants to use for a particular type of communication. From interviews results that a user might want to indicate three types of preferences:

- The RMS must use a device nearby the recipient. In this case, the RMS first investigates the device that the recipient is using. It keeps on increasing the search area until it finds a device that it can use, until it has investigated all devices in the whole house, or until there is no more time to investigate any more devices.
- The RMS must investigate a particular group of devices. For example, the recipient only wants to use a particular screen in the kitchen and a particular screen in the living room, for a particular type of communication.
- The RMS must try to find one device per room. The user might indicate for every room whether the RMS should try to find a device in that room, and which devices the RMS should preferably use.

From interviews results that a user wants to indicate preferences, based upon the priority[r] and the type of communication[19]. Table 2 shows an example of preferences that determine which devices the RMS must investigate.

Table 2, Preferences table to determine what devices to investigate for communication

Priority[r]		Normal	High	Emergency
Type of communication	**E-mail**	Group	Group	Device nearby
	SMS	Group	Group	Device nearby
	Telephone	Device nearby	Device nearby	Device nearby
	Video	Device nearby	Device nearby	Device nearby

[19] User tests must determine what kind of preferences people want to indicate (if any).

Per default, the RMS uses a device near the recipient such that the users do not *have* to indicate any preferences. The last type of preference needs some more explanation. This preference indicates that the RMS must search for a device in every room of the house. Figure 15 illustrates an example of this search.

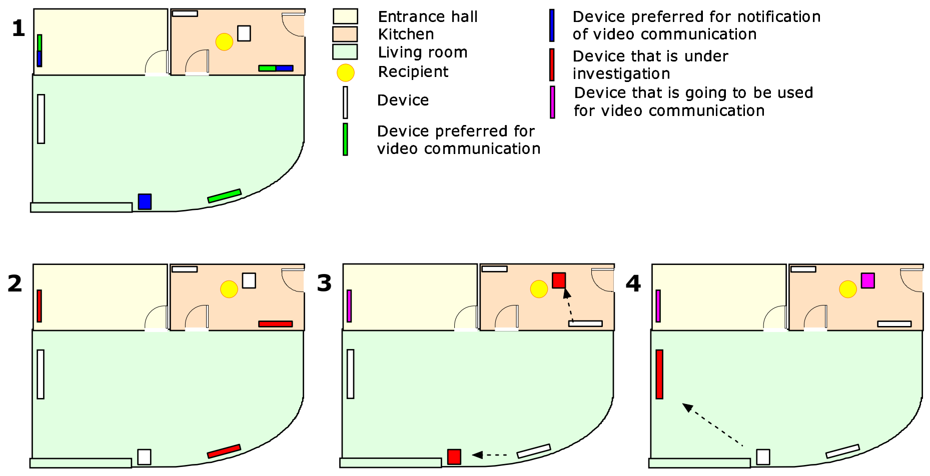

1
Entrance hall
Kitchen
Living room
Recipient
Device
Device preferred for video communication
Device preferred for notification of video communication
Device that is under investigation
Device that is going to be used for video communication
2
3
4

Figure 15 Search for devices that can be used for video communication

Per room, the RMS uses the following search order:

- First, the RMS investigates the devices that the user prefers to use for the proposed type of communication (if any).
- Next, it investigates the devices that the user prefers to use for communication in the same class of communication as the proposed type of communication (if any).
- Next, it investigates the devices that are preferred for notification (if any).
- Next, it investigates the device that the recipient is using (if any).
- Next, it investigates all remaining devices.

In the scenario that is depicted in Figure 15, the recipient has indicated which devices she wants to use for video communication, and which devices she wants to use for the notification of video communication (Figure 15, map1). The RMS starts investigating one device per room (Figure 15, map 2). It starts with the devices that the recipient prefers to use for video communication. For two rooms, the RMS determines that the device that it investigated is not available for video communication. Therefore, the RMS investigates a different device in the same room, according to the search order that is explained above (Figure 15, map 3). Again, it determines that it must search for another device in one room (Figure 15, map 4).

The RMS stops investigating devices in a room as soon as it finds one device in that room that it can use for video communication. As soon as the RMS has found *one* device that it can use for video communication (it doesn't matter in which room), it determines that it is possible to start video communication. Now the RMS can postpone the search for video communication, and start the notification (if it has determined that the recipient is available for video communication). When it has started notification, it can resume the search for devices that it can use for video communication and for other types of communication.

The search for devices that the RMS can use for notification is roughly the same as the search for devices for video communication. The user can indicate the same types of preferences. If the user indicates that the RMS should use a device in every room (the last type of preferences), the RMS investigates the devices in a room as follows:

- First, it investigates the devices that are preferred for the notification of the type of communication that the initiator proposes (if any).
- Next, it investigates the devices where the RMS can start the type of communication that the initiator proposes.
- Next, it investigates the device that the recipient is using (if any).
- Next, it investigates all remaining devices.

3.4 NOTIFICATION

When the RMS determines that it should notify the recipient of a communication request, it must determine *how* it must notify the recipient. The RMS needs to adapt the User Interface (UI) of the notification such that on the one side the recipient is not disturbed by the notification, and on the other side the recipient is aware of the notification in time. I consider two properties of a UI that the RMS should adapt: the device(s) that the RMS uses for the UI of the notification, and the obtrusiveness of the UI. Of course, the RMS must also adapt the UI to user preferences such as 'use large icons' or 'use an animated character for interaction'. However, these kinds of preferences are not in the scope of this graduation project since these preferences do not influence one's reachability. The last paragraph of section 3.3 explains how the RMS determines which device(s) the RMS must use for the notification. This section explains the term 'obtrusiveness', and it indicates how the RMS can determine the obtrusiveness of a UI.

3.4.1 Obtrusiveness

The obtrusiveness of a UI determines to what extent the UI stands out. For example, the RMS can create an unobtrusive UI (e.g. a small icon in the corner of a screen), or an obtrusive UI (e.g. a loud sound and a flashing pop-up menu). In this graduation project we assume that the RMS uses only two UI modalities: a Graphical User Interface (GUI) and Audio. This way it is possible to show how the RMS deals with the UI of a notification without going into too much detail[20]. The RMS determines the obtrusiveness for every UI-modality, since every UI modality influences the obtrusiveness of a UI in a different way. The obtrusiveness of a UI modality is fixed by a set of (appearance, start event, termination event)-tupels. The appearance determines the UI description of a UI modality. The appearance is classified as follows[21]: 'Unobtrusive', 'Normal', and 'Obtrusive'. Table 3 shows how the RMS can translate the values of appearance into UI descriptions. For example, when the appearance of the GUI is 'Obtrusive', the UI description of the GUI is 'Flashing pop-up menu'. Obviously, the real UI description is more detailed.

Table 3 Example of UI descriptions, based upon the appearance

UI modality		GUI	Audio
Appearance	**Unobtrusive**	Icon	Low volume
	Normal	Pop-up menu	Normal volume
	Obtrusive	Flashing pop-up menu	High volume

The start event indicates when the RMS must start to use a particular appearance for a particular UI modality, e.g. after 10 seconds. The termination event indicates when the RMS must stop to use a particular appearance for a particular UI modality, e.g. after 10 seconds or when the notification is obsolete. The notification is obsolete in the following situations:

- If the status of the recipient changes such that the recipient is not available for communication anymore. In this situation, the RMS must stop the notification until the recipient is again available for communication.
- If the recipient starts to interact with the UI of the RMS. In this situation, the notification has reached its' goal.
- If the communication is not real-time and the recipient starts the communication application that corresponds with the type of communication that the notification is about. For example, when the RMS tries to notify the recipient of a new e-mail message and the recipient starts an e-mail application, that application will show the new messages to the recipient. Therefore, the RMS does not need to show a notification anymore.

[20] If a device has only capabilities for one of these two modalities, the RMS might have to adjust the obtrusiveness. I leaf it to the UI designer to deal with this situation.
[21] Further research must determine the most convenient classification.

Table 4 shows an example of a set of tupels that might constitute the obtrusiveness of a UI modality for a particular notification.

Table 4 Example of the obtrusiveness of a UI modality

Tupel		Appearance	Start event	Termination event
Set of tupels	**Tupel 1**	Unobtrusive	Immediately	After 5 seconds
	Tupel 2	Normal	After termination of Tupel 1	After 7 seconds
	Tupel 3	Obtrusive	After termination of Tupel 2	After 5 seconds

3.4.2 Determine obtrusiveness

From interviews results that RMS should determine the obtrusiveness based upon preferences with respect the priority[r] of the communication, the type of communication, the location of people, the focus of people, the activities near the cluster, and the status of the recipient. The default obtrusiveness of a UI modality is based upon the type of communication and the priority[r] of the communication. Note that the user does not have to indicate any preferences: the designers of the RMS must create default preferences. Table 5 shows an example of the default preferences for non real-time communication: if the priority[r] is 'High', the RMS shows a GUI with appearance 'Normal' until the notification is obsolete. Furthermore, it plays an audio tune with appearance 'Unobtrusive' for one second. These default values are refined when the RMS takes other aspects into account (e.g. activities).

Table 5, Example of the obtrusiveness of a UI modality for non real-time communication

UI modalities		GUI			Audio		
		Appearance	Start Event	Termination event	Appearance	Start Event	Termination event
priority[r]	**Normal**	Unobtrusive	Immediately	Notification is obsolete	Unobtrusive	Immediately	After 1 second
	High	Normal	Immediately	Notification is obsolete	Unobtrusive	Immediately	After 1 second
	Emergency	Obtrusive	Immediately	Notification is obsolete	Normal	Immediately	Notification is obsolete

§ ***Location of people***

The location of people with respect to the device that is used for the UI, influences the appearance. For example, if the RMS knows that there are no people in the room where the device is located, and the RMS cannot create a UI in the room where the recipient is located, it might change the appearance of the UI modality 'Audio' into 'Obtrusive'. This way the recipient might still notice the notification although she is not near any devices that can be used for the UI. This situation might occur when the communication is real-time and the recipient is in a room without any devices (e.g. the cellar).

§ ***Focus of people***

When someone is using the device that the RMS must use for the UI, this poses an upper bound to the appearance of the UI modalities. For example, the upper bound of the GUI is 'Normal' and the upper bound for Audio is 'Unobtrusive'. This way, the RMS makes sure that the user of the device is not disturbed by the notification. The RMS might also change the appearance of a UI-modality over time. For example, it might use a GUI with appearance 'Normal' for the first 5 seconds of the notification, and it might use a GUI with appearance 'Unobtrusive' for the rest of the time. This way the GUI no longer bothers a person after 5 seconds.

§ ***Activities near the cluster***

Every activity might influence the obtrusiveness of the UI of the notification in a different way. User tests must determine during which activities the RMS must adapt the obtrusiveness of the UI of the notification. Figure 16 illustrates one example: if someone in the room is vacuum cleaning, the RMS might use the appearance 'Obtrusive' for the GUI and the Audio, such that the recipient will notice the notification.

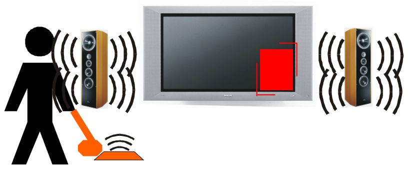


Figure 16 RMS might change the obtrusiveness during the activity 'Vacuum cleaning'

§ ***Status of the recipient***

The user might want the RMS to adapt the obtrusiveness, based upon the status of the user (e.g. because there is no system that can detect activities). Figure 17 shows an example. The status of the user is 'Sleeping'. Therefore, the RMS starts the UI modality 'Audio' with the appearance 'Unobtrusive'. After 5 seconds it changes the appearance to 'Normal', and after 15 seconds it changes the appearance to 'Obtrusive'.

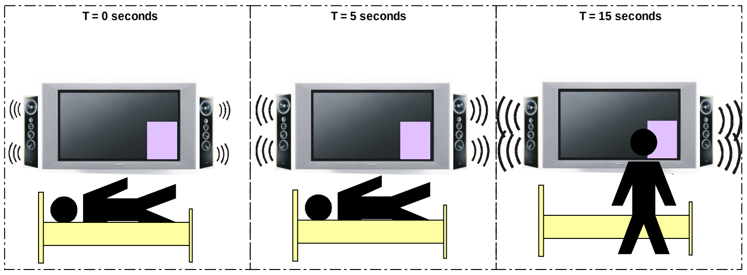


Figure 17 RMS might change the obtrusiveness during the status 'Sleeping'

§ ***Conflicts***

When conflicts occur, e.g. the RMS determines that the appearance must be both obtrusive and unobtrusive at the same time, the RMS uses the default values for obtrusiveness that are shown in Table 5. In other words, the RMS does not adapt the obtrusiveness of the notification in case of conflicts.

3.5 DETERMINE APPROPRIATE ACTION

When the RMS receives a communication request, it first determines whether it should notify the recipient. If the RMS determines that it should notify the recipient, it tries to notify the recipient for a specific period. If the recipient does not respond within that period, the RMS undertakes one of the actions that are explained below. While the RMS is trying to notify the recipient, it might determine that it should stop the notification, e.g. because the recipient is not available for communication anymore (e.g. because her status changed to 'Not available'). In this situation, the RMS stops the notification and undertakes one of the actions that are explained below.

If the RMS should not notify the recipient (anymore), or it is not possible to notify the recipient, the RMS undertakes one of the following actions:

- Start monitoring. The RMS waits until it can start notification. The RMS will typically do this when the communication request regards non real-time communication. The RMS monitors whether the recipient is aware of the communication in time. If the recipient is not aware of the communication in time, it undertakes one of the other actions, e.g. it increases the priority^r of the communication.

- Forward communication. The RMS can forward the communication to another communication channel. The RMS might need to transform the communication to be able to forward the communication. For example, it might need to transform an email-message into a SMS message to be able to forward it to a mobile phone.
- Start interaction with the initiator. The RMS can inform the initiator that the recipient is not available. In this case, it might propose other types of communication to the initiator (e.g. propose to record a voice-mail message). The RMS will typically do this when communication request regards real-time communication.
- Increase priority^r. Section 3.2 explains that the RMS might increase the priority^r of communication when the recipient is not aware of the communication in time. This might make the priority^r higher than the threshold, thus this might make the recipient available for communication. If the recipient is available for communication, the RMS might start notification (as explained in section 3.1). Otherwise, it undertakes one of the other actions, e.g. start monitoring.

The RMS determines the appropriate action based upon user preferences, e.g. preferences with respect to the type of communication, the priority^r of communication, and the identity of the initiator. Per default, the RMS starts interaction with the initiator when the communication is real-time, and it starts monitoring when the communication is *not* real-time. User tests must determine what preferences the users would like to indicate and what actions the RMS should undertake per default. Table 6 shows an example of the preferences that a recipient might indicate for non real-time communication.

Table 6 Table to determine appropriate action when RMS should not start notification

Non real-time communication				
Identity		**Unknown**	**Girlfriend**	**Colleague**
priority^r	**Normal**	Start monitoring	Start monitoring	Start monitoring
	High	Start monitoring	Start monitoring Increase priority^r after 1 day	Start monitoring Increase priority^r after 1 day
	Emergency	Forward to mobile phone	Forward to mobile phone	Forward to mobile phone

4 ARCHITECTURE OF RMS

This chapter explains the system architecture of the RMS. Section 4.1 introduces the components that the RMS consists of. Sections 4.2 to 4.9 explain the operation of these components in detail. These sections explain the operation of the components for the situation that both the recipient and the initiator of communication have a RMS. Section 4.10 explains how the RMS deals with the situation that either the initiator or the recipient does not have a RMS. The components that are introduced in this chapter are the components that a RMS needs to deal with *one* communication request. Chapter 6 explains how the RMS deals with multiple communication requests at the same time, and how the RMS can be incorporated in the WWICE system.

4.1 OVERVIEW OPERATION RMS

This section gives an overview of the operation of the RMS. Section 4.1.1 introduces the components of the RMS that are shown in Figure 18[22]. Section 4.1.2 gives an impression of the interaction between these components when an initiator negotiates with (the RMS of) a recipient to start communication. Section 4.1.3 gives an overview of the operation of the RMS of the initiator, when an initiator indicates that she wants to start communication. Section 4.1.4 gives an overview of the operation of the RMS of the recipient, when it must deal with a communication request.

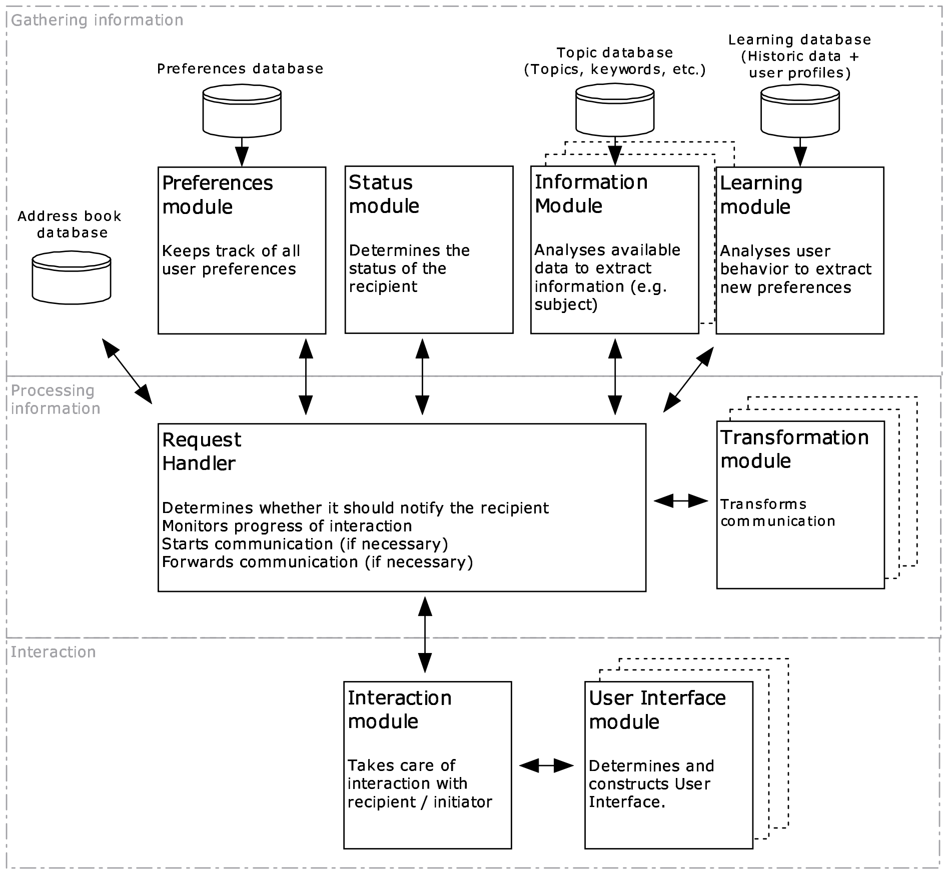

Gathering information
Preferences database
Topic database (Topics, keywords, etc.)
Learning database (Historic data + user profiles)
Address book database
Preferences module
Keeps track of all user preferences
Status module
Determines the status of the recipient
Information Module
Analyses available data to extract information (e.g. subject)
Learning module
Analyses user behavior to extract new preferences
Processing information
Request Handler
Determines whether it should notify the recipient
Monitors progress of interaction
Starts communication (if necessary)
Forwards communication (if necessary)
Transformation module
Transforms communication
Interaction
Interaction module
Takes care of interaction with recipient / initiator
User Interface module
Determines and constructs User Interface.

Figure 18 Global system architecture of the RMS

[22] Figure 18 shows only the most important interaction between the components. This interaction is explained in detail in sections 4.2 to 4.10.

4.1.1 Components of RMS

This section introduces the functionality of the components of the RMS that are shown in Figure 18. The functionality is based upon the functionality of a RMS that is described in section 2.4, and the model for Reachability Management that is explained in chapter 3. Sections 4.2 to 4.9 explain the components of the RMS in detail.

Request Handler
The Request Handler is responsible for dealing with communication requests (e.g. a telephone call or new email). The Request Handler determines whether it should notify the recipient. If it determines that it should notify the recipient, it instructs the Interaction module to take care of this. If it determines that it should not notify the recipient, it undertakes a different action, e.g. it might instruct the Interaction module to inform the initiator that the communication request is rejected. Furthermore, the Request Handler monitors the progress of the notification and it undertakes actions when the recipient does not notice the communication in time. If the recipient and initiator agree to start a particular type of communication, the Request Handler starts the corresponding communication application. Finally, the Request Handler is responsible for starting communication when a user of the RMS wants to *start* communication. In this situation, the user of the RMS is the initiator of the communication.

Status module
The Status module determines the status of the recipient. The status is the concept that allows a user of the RMS to control her reachability. The Status module implements the ‘Status manager’ that is introduced in section 3.2.3.

Information module
An Information module provides a service that extracts information from available data. For example, there might be an Information module that is able to extract the subject of a communication request from a text.

Preferences module
The Preferences module takes care of personalization. The Preferences module stores all preferences of the users of the RMS in the Preferences database. The components of the RMS can request these preferences. An example of a preference is a preference that indicates when someone wants to have a particular status, e.g. a user wants to have the status ‘Not available’ between 0:00h and 7:00h.

Interaction module
The interaction module takes care of the interaction between the RMS and a person. Therefore, it determines a so-called interaction scheme. The interaction scheme consists of the options that a User Interface module must propose to a person (e.g. it can propose to an initiator to record a voice-mail message), and the information that it must provide (e.g. it can inform an initiator that the recipient is available within an hour).

User Interface module
The User Interface modules take care of the User Interface of the RMS. Each User Interface module can provide a particular UI. For example, there might be a User Interface module that uses the UI capabilities of the devices of the home system of the recipient, to interact with a person. There might also be a module that can interact with a person using a telephone connection.

Transformation module
Every Transformation module provides a service that can transform one type of communication into another. For example, there might be a Transformation module that can transform a voicemail message into an e-mail message.

Learning module
The learning module stores all kinds of data in a Learning database (e.g. the response of a user when a particular initiator is calling), and tries to extract new user preferences.

4.1.2 Interaction between RMS initiator and RMS recipient

This section gives an impression of the interaction between the RMS of the initiator and the RMS of the recipient, when the RMS of an initiator sends a communication request to the RMS of the recipient[23]. More specific, this section deals with the interaction between the components of the RMS-es that take care of the negotiation process between the initiator and (the RMS of) the recipient. These components are the Request Handler, the Interaction module, and the User Interface module of the two RMS-es[24]. Figure 19 illustrates one possible sequence of events that leads to the set-up of a particular type of communication[25]. Section 4.2 and section 4.3 explain the interaction between the Request Handlers and the Interaction modules in detail.

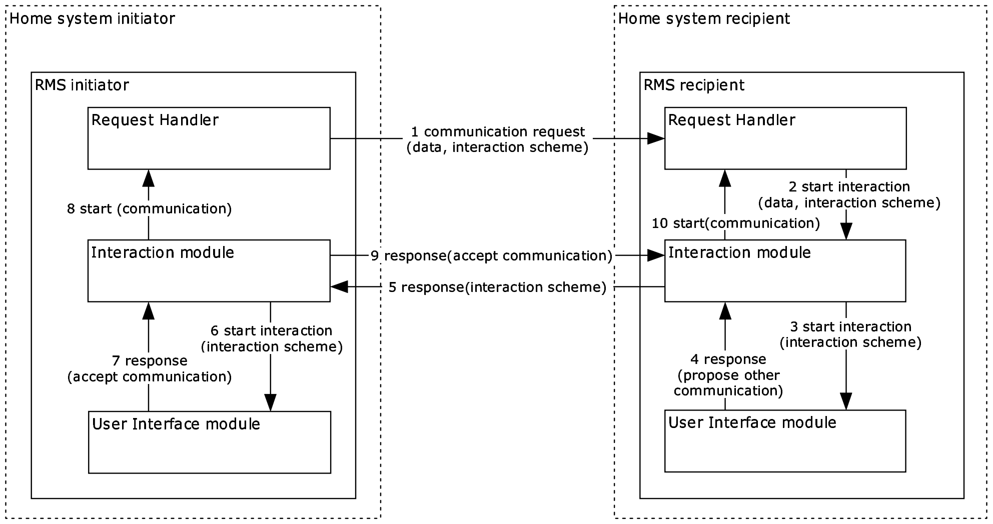

Home system initiator
RMS initiator
Request Handler
8 start (communication)
Interaction module
6 start interaction (interaction scheme)
7 response (accept communication)
User Interface module
1 communication request (data, interaction scheme)
9 response(accept communication)
5 response(interaction scheme)
Home system recipient
RMS recipient
Request Handler
2 start interaction (data, interaction scheme)
10 start(communication)
Interaction module
3 start interaction (interaction scheme)
4 response (propose other communication)
User Interface module

Figure 19 Global interaction between RMS initiator and RMS recipient

When an initiator indicates to her RMS that she wants to start communication, she uses the UI of her RMS to indicate with whom she wants to communicate. The Request of that RMS starts an Interaction module to gather all information that is necessary for a communication request. Meanwhile, the Request Handler determines whether it is possible to start the communication at the WWICE system of the initiator. In this scenario, the Request Handler is able to start the communication that the initiator wants to start. The Interaction module constructs an interaction scheme for the RMS of the recipient and passes that scheme to the Request Handler. The Request Handler sends a communication request (that contains the interaction scheme) to the RMS of the recipient (message 1).

The Request Handler of the RMS of recipient receives the communication request. In this scenario, it determines that it should notify the recipient, thus it instructs the Interaction module to start notification (message 2). The Interaction module asks the User Interface module to construct a UI (message 3). The recipient chooses the option ‘Propose other type of communication’. The User Interface module sends this response to the Interaction module (message 4). The Interaction module constructs an interaction scheme that proposes the type of communication that the recipient wants to start, to the initiator. It sends this scheme to the Interaction module of the RMS of the initiator (message 5).

[23] Note that the RMS of an initiator only sends a communication request to the RMS of the recipient, when she wants to start real-time communication. Otherwise the RMS of the initiator just starts the appropriate communication application, e.g. an e-mail application.
[24] Note that both the initiator and the recipient have a RMS. Section 4.10 explains the situation that one of these two parties does not have a RMS.
[25] For simplicity, Figure 19 shows direct interaction between the two Interaction modules. Section 6.6.3 explains that Interaction modules actually communicate via the Request Handlers.

The Interaction module of the RMS of the initiator forwards the interaction scheme to the User Interface module (message 6). The initiator selects the action 'accept communication'. The User Interface module sends this response to the Interaction module (message 7). The Interaction module instructs the Request Handler to start this type of communication (message 8), and informs the Interaction module of the RMS of the recipient (message 9). The Interaction module of the RMS of the recipient also instructs the Request Handler to start communication (message 10). Now, the Request Handlers can cooperate to set-up the communication.

4.1.3 Global operation of the RMS of the initiator

This section globally explains how the components of the RMS of the *initiator* cooperate when an initiator indicates that she wants to start communication. Figure 20 illustrates the operation of the RMS of the initiator[26].

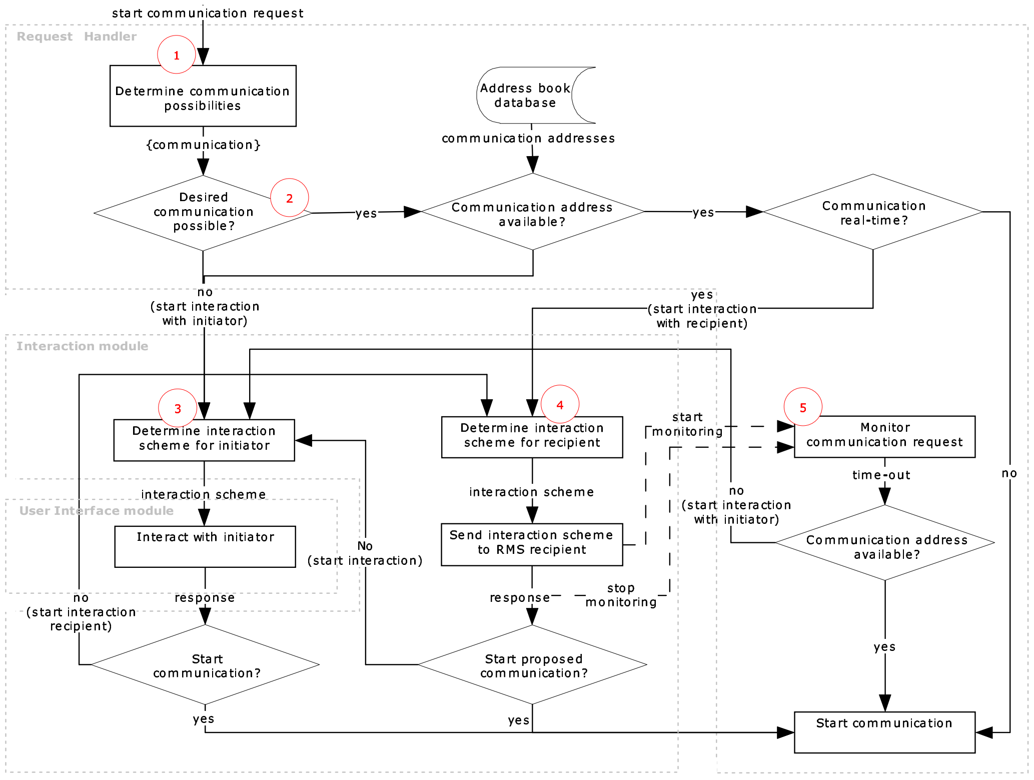

start communication request
Request Handler
1
Determine communication possibilities
{communication}
Address book database
communication addresses
Desired communication possible?
2
yes
Communication address available?
yes
Communication real-time?
no
(start interaction with initiator)
yes
(start interaction with recipient)
Interaction module
3
Determine interaction scheme for initiator
4
Determine interaction scheme for recipient
5
Monitor communication request
start monitoring
time-out
no
interaction scheme
interaction scheme
no
(start interaction with initiator)
User Interface module
Interact with initiator
Send interaction scheme to RMS recipient
No
(start interaction)
Communication address available?
no
(start interaction recipient)
response
response
stop monitoring
yes
Start communication?
Start proposed communication?
yes
yes
Start communication

Figure 20 Flowchart of the operation of the RMS of the initiator

The flowchart of Figure 20 starts at the moment that the Request Handler knows that a user of the RMS wants to start a particular type of communication (circle 1). The Request Handler first determines what types of communication it can start within the home system (e.g. the WWICE system) of the initiator. If the Request Handler is able to start the type of communication that the user wants to start, at the device that the user is currently using (circle 2), it retrieves the appropriate communication address from the Address book database. The communication address is either the address of the RMS of the recipient, or the address of the type of communication (e.g. the email address for email communication). If the Request Handler is able to retrieve the communication address, it can undertake two actions: if the communication is not real-time, the Request Handler starts the appropriate communication application. For example, it starts an email application when the initiator wants to send an email message. If the communication is real-time, the Request Handler asks the

[26] The figure contains circles to be able to reference particular parts of the flowchart.

Interaction module to start interaction with (the RMS of) the recipient. The interaction module constructs an interaction scheme and sends it (via the Request Handler) to the Request Handler of the recipient (Figure 19, message 1).

If the Request Handler determines that it cannot start the type of communication that the initiator wants to start, or it cannot retrieve the appropriate communication address, it asks the Interaction module to start interaction with the initiator to solve the problem (circle 3). The Interaction module constructs an interaction scheme and sends it to the User Interface module. If the initiator decides to start non real-time communication, the Request Handler starts the corresponding communication application. If the initiator provides the address of the RMS of the recipient, or she decides to start a type of real-time communication that the Request Handler *can* start, the Request Handler instructs the Interaction module to start interaction with the recipient.

If the Interaction module must start interaction with the recipient (circle 4), it constructs an interaction scheme and sends it to the RMS of the recipient[27]. Next, it waits for the response. If the recipient agrees to start a particular type of communication, the Interaction module instructs the Request Handler to start the appropriate communication application. Otherwise, it receives an interaction scheme from the Interaction module of the RMS of the recipient. It forwards this scheme to the User Interface module. Figure 19 illustrates the negotiation process that might start between the (RMS of the) recipient and the (RMS of the) initiator.

When the Interaction module sends a communication request to the RMS of the recipient, it asks the Request Handler to monitor the response from the RMS of the recipient (circle 5). If there is no response for a long time (e.g. 20 seconds), this means that the RMS of the recipient is not working properly. If the Request Handler has the communication address of the recipient for the type of communication that the initiator wants to start (e.g. it has the telephone number of the recipient when the initiator wants to start telephone communication), the Request Handler tries to start communication using that address (thus it tries to start the communication directly, without consulting the RMS of the recipient). Otherwise, it asks the Interaction module to inform the initiator that the RMS of the recipient is not responding.

[27] The first interaction scheme that the Interaction module sends to the RMS of the recipient, is wrapped in a communication request from the Request Handler of the RMS of the initiator to the RMS of the recipient.

4.1.4 Global operation of the RMS of the recipient

This section globally explains how the components of the RMS of the *recipient* cooperate to deal with a communication request. Figure 21 illustrates the operation of the RMS of the recipient[28]. The decisions that the components make are based upon the model for Reachability Management that is explained in chapter 3. Sections 4.2 to 4.9 explain the operation of the components in detail.

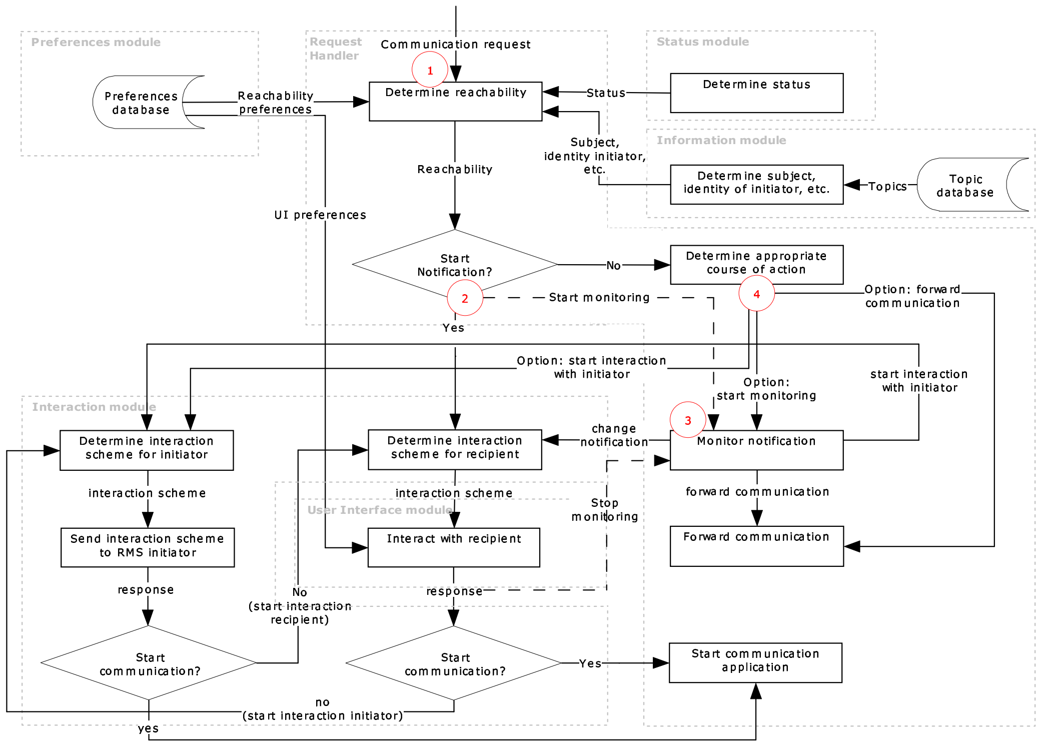


Figure 21 Flowchart of the operation of the RMS of the recipient

Section 4.1.3 explains that the RMS of the initiator sends a communication request to the RMS of the recipient, if the initiator wants to start real-time communication. If the initiator has started non real-time communication, this means that she sent a *message* to the recipient. For example, the initiator sent an e-mail message. A component of the RMS of the recipient that is running at the e-mail server, detects this new e-mail message, constructs a communication request based upon this message, and sends the communication request the Request Handler. This component is not explained in more detail, since its' only functionality is to construct and send communication requests.

Figure 21 shows that a communication request arrives at the Request Handler of the recipient (circle 1). The Request Handler determines the reachability of the recipient. In other words, it determines which types of communication the recipient is willing to start, and whether it is possible to start these types of communication. In order to determine what types of communication the recipient is willing to start, the Request Handler requests the recipient's preferences from the Preferences module, and the recipient's status from the Status module. Furthermore, it might need an Information module to obtain some information. For example, it might ask an Information module to determine the subject of the communication. Next, the Request Handler determines whether it should notify the recipient of the communication request.

[28] The figure contains circles to be able to reference particular parts of the flowchart

If the Request Handler determines that it should notify the recipient, it starts an Interaction module and instructs that module to notify the recipient (circle 2). The Interaction module uses the interaction scheme that is part of the communication request to constructs a new interaction scheme for the recipient. Next, it instructs a User Interface module to create a UI for the notification. The Interaction module waits for the response of the recipient. If the recipient is willing to start the communication, it instructs the Request Handler to start the communication. Otherwise, it constructs an interaction scheme and sends it to the RMS of the initiator.
If the Request Handler determines that it should not notify the recipient, it determines the appropriate action (circle 4). It might forward the communication (e.g. forward a phone call to another RMS system), it might wait until it can start notification (start monitoring), or it might ask the Interaction module to inform the initiator that the communication request is rejected.

While the Request Handler is monitoring, it continuously determines whether it should (still) notify the recipient. If it should not notify the recipient anymore (e.g. because the user status changes to ‘Not available’), it instructs the Interaction module to (temporarily) stop the notification and keeps on monitoring. If it should start notifying the recipient (e.g. because the user status changes to ‘Available’ or because the Request Handler has increased the priority[r]), it instructs the Interaction module to start notification and keeps on monitoring. Furthermore, the Request Handler monitors the notification such that it can undertake action if the recipient is not aware of the communication in time. The Request Handler can undertake the following actions:

- Increase the priority[r] of the communication. This action might cause the Request Handler to determine that it should start notification. If the Interaction module was already trying to notify the recipient, this action might cause a different type of notification (e.g. the User Interface module might create a more obtrusive UI).
- Start interaction with the initiator. The Request Handler might determine that it should inform the initiator that the recipient does not respond to the communication request. The Request Handler will typically choose this action when the communication request regards real-time communication.
- Forward the communication. For example, the Request Handler might forward an e-mail message to another e-mail address, or it might forward a communication request for real-time communication to another RMS.

If the Interaction module must start interaction with the initiator, it constructs an interaction scheme and sends it to the RMS of the initiator (via the Request Handler). There are two situations that cause the Interaction module to start interaction with the initiator:

- The Interaction module is already interacting with the recipient and the recipient does not accept the communication that the initiator proposes. For example, she proposes a different type of communication.
- The Request Handler determines that it should not notify the recipient and that it should start interaction with the initiator. For example, if an initiator wanted to start telephone communication, the Interaction module might send an interaction scheme that proposes to record a voicemail instead.

The Interaction module waits for the response from the initiator. If the initiator is willing to start communication that was proposed by the interaction scheme, the Interaction module instructs the Request Handler to start communication. If the initiator increases the priority[i] of the communication such that the Interaction module should notify the recipient, the Interaction module constructs an interaction scheme and instructs a User Interface module to create a UI for the notification. If the initiator proposes a new type of communication to the recipient, the Interaction module receives an interaction scheme from the RMS of the initiator. It forwards this scheme to the User Interface module such that the recipient can respond. The following sections explain in detail how the components of the RMS interact to deal with a communication request.

4.2 REQUEST HANDLER

The RMS starts a Request Handler when it must deal with a task, e.g. to deal with a communication request. As soon as the Request Handler has dealt with the task, the Request Handler stops its' execution. If the Request Handler determines that it must start interaction with a person, it starts an Interaction module to take care of this. It indicates what type of interaction the Interaction module must start (e.g. notification of a recipient or negotiation with an initiator), and it provides all data that is of interest for the interaction (e.g. the identity of the recipient and the priority[r] of the communication). Since the Interaction module is able to take care of the interaction with both the recipient and the initiator, the Request Handler starts only one Interaction module, as illustrated in Figure 22[29].

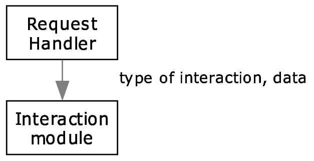


Figure 22 A Request Handler starts only one Interaction module

The main tasks of the Request Handler are dealing with a communication request, and starting communication when the user indicates that she wants to start communication (in this situation, the user is the initiator of the communication). Section 4.2.1 explains how the Request Handler starts communication, and section 4.2.2 explains how the Request Handler deals with an incoming communication request.

4.2.1 Starting communication

If a user wants to start communication, this means that the user is the initiator of the communication. The RMS starts a Request Handler to deal with this task. Figure 23 shows the state diagram of the Request Handler when it deals with this task[30]. The following subsections explain each state in detail.

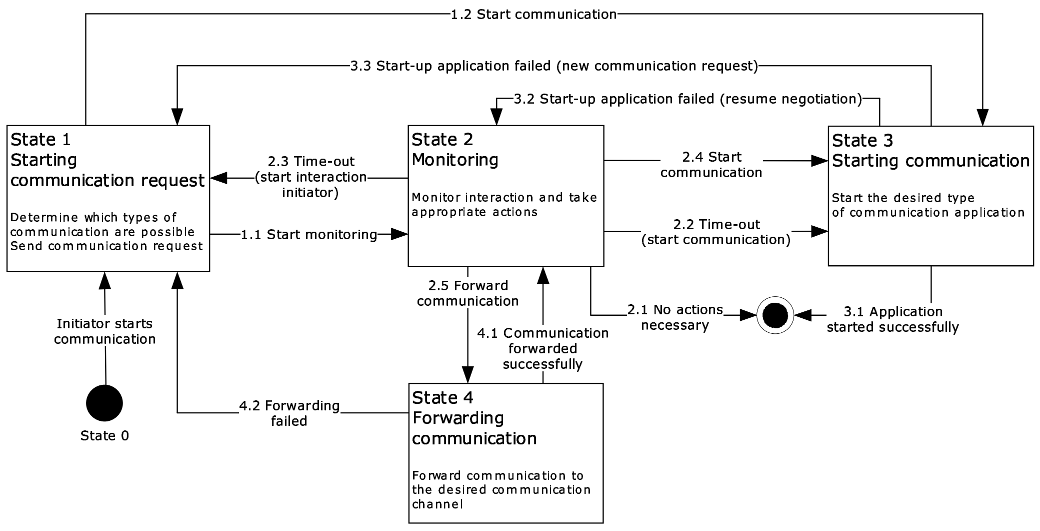

1.2 Start communication
3.3 Start-up application failed (new communication request)
3.2 Start-up application failed (resume negotiation)
State 1
Starting communication request
Determine which types of communication are possible
Send communication request
2.3 Time-out (start interaction initiator)
1.1 Start monitoring
State 2
Monitoring
Monitor interaction and take appropriate actions
2.4 Start communication
2.2 Time-out (start communication)
State 3
Starting communication
Start the desired type of communication application
2.5 Forward communication
4.1 Communication forwarded successfully
2.1 No actions necessary
3.1 Application started successfully
Initiator starts communication
State 0
4.2 Forwarding failed
State 4
Forwarding communication
Forward communication to the desired communication channel

Figure 23 State diagram 'Start communication'

[29] Note that Figure 25, Figure 30, Figure 35, and Figure 36 extend Figure 22.

State 0
When a user wants to start communication, the Request Handler starts an Interaction module and instructs that module to gather all information that is necessary to start communication, e.g. the type of communication and the identity of the recipient. Next, it goes into the 'Starting communication request' state to wait for the results from the Interaction module.

State 1: Starting communication request
When the Request Handler is in this state, it determines what types of communication it can start within the home system (e.g. the WWICE system) of the initiator. Furthermore, it interacts with the Interaction module to gather all information that it needs for a communication request. For example, if the initiator wants to start communication, but the Request Handler cannot retrieve the appropriate communication address from the Address book database, it instructs the Interaction module to inform the recipient and to ask for the appropriate address. The following paragraphs explain which events can occur when the Request Handler is in the 'Starting communication' state.

Event 1.1: Start monitoring
If the initiator wants to start real-time communication and the Request Handler has all information that it needs to send a communication request (e.g. the address of the RMS of the recipient, and an interaction scheme from the Interaction module), it constructs a communication request and sends it to the RMS of the recipient. Next, it goes into the 'Monitoring' state.

Event 1.2: Start communication
There are two situations that cause the Request Handler to start communication:
- The initiator wants to start non real-time communication. In this situation, the Request Handler does not need to negotiate with the RMS of the recipient. It can just start the appropriate communication, e.g. an email application if the initiator wants to send an email message.
- The Request Handler does not have the address of the RMS of the recipient, but it does have communication address for the type of communication that the initiator wants to start, e.g. the telephone number of the recipient. In this situation, the RMS tries to start communication without consulting the RMS of the recipient.

State 2: Monitoring
The Request Handler is in the 'Monitoring' state when it has sent a communication request to the RMS of the recipient and when the initiator and the (RMS of the) recipient are negotiating to start communication. The following paragraphs explain which events can occur when the Request Handler is in the 'Monitoring' state.

Event 2.1: No actions necessary
The initiator decides to cancel the communication request. The Interaction module informs the Request Handler of this event. The Request Handler does not have to do anything anymore thus it stops dealing with the communication request.

Event 2.2: Time-out (start communication)
The RMS of the recipient does not respond to the communication request of the initiator. In this situation, the Request Handler does not only have the address of the RMS of the recipient, but also the communication address for the type of communication (e.g. the phone number for telephone communication). It instructs the Interaction module to inform the initiator that it is going to start the communication without consulting the RMS of the recipient, and goes into the 'Starting communication' state.

[30] The events and states have a number to be able to reference them easily. This number does not indicate the order of occurrence.

Event 2.3: Time-out (start interaction initiator)
The RMS of the recipient does not respond to the communication request of the initiator. In this situation, the Request Handler does *not* have the communication address for the type of communication (e.g. the telephone number). Therefore, it determines that the communication request failed, thus it instructs the Interaction module to start interaction with the initiator (to inform her that the RMS of the recipient does not respond). Now, the initiator might provide a communication address or she might start a different type of communication (e.g. email). The Request Handler goes into the 'Starting communication request' state to await a new communication request from the initiator.

Event 2.4: Start communication
The Interaction module instructs the Request Handler to start communication, for example because the recipient and the initiator have agreed to start some type of real-time communication, or because the initiator decides to start non real-time communication. The Request Handler goes into the 'Starting communication' state to start the appropriate communication application.

Event 2.5: Forward communication
The RMS of the recipient proposed to the initiator to forward the communication, and the initiator accepted that proposal. The Interaction module instructs the Request Handler to forward the communication, thus the Request Handler goes into the 'Forwarding communication' state.

State 3: Starting communication
The Request Handler is in the 'Starting communication' state when the Interaction module has instructed the Request Handler to start a particular type of communication. If the Request Handler must start real-time communication, it cooperates with the RMS of the recipient to start the communication. In this situation, the Request Handler might need the services of a Transformation module. For example, if the Request Handlers must start video communication, the video stream might need to be transformed from mpeg-2 to mpeg-4 if the home system of the recipient can only deal with mpeg-4. The following events can occur when the Request Handler starts communication.

Event 3.1: Application started successfully
If the Request Handler has successfully started the communication application, the Request Handler informs the Interaction module that it can stop the interaction with the initiator and (the RMS of) the recipient. Next, it stops dealing with the communication request.

Event 3.2: Start-up application failed (resume negotiation)
In this situation, the Request Handler is collaborating with the Request Handler of the recipient to start the communication. Something goes wrong during the start-up of the communication application (e.g. the necessary resources are not available anymore). The Request Handlers start to communicate with each other to determine whether the recipient or the initiator may do the next communication proposal. For example, if the reason for the failure is that the device that the recipient is using is not working properly, the recipient could start to use a different device. Therefore, the recipient may do the next communication proposal, e.g. she might start to use a different device and propose the same type of communication again, or she might propose a different type of communication to the initiator. The Request Handlers instruct their Interaction modules to take care of the interaction, such that the initiator and the recipient can resume their negotiation. The Request Handler goes into the 'Monitoring' state to wait for the result of the negotiation.

Event 3.3: Start-up application failed (new communication request)
In this situation, the Request Handler is not collaborating with the Request Handler of the recipient, e.g. it is trying to start an email application. Something goes wrong during the start-up of the communication application. The Request Handler determines that the communication request failed, thus it instructs the Interaction module to start interaction with the initiator. The Request Handler goes into the 'Starting communication request' state to await a new communication request from the initiator.

State 4: Forwarding communication

If the Request Handler is in this state, this means that it must forward a communication request regarding real-time communication to a new RMS[31]. Either the Request Handler of the recipient has provided the address of the new RMS to which the communication request should be forwarded, or it has not provided this address[32]. In the last situation, the Request Handler of the recipient acts as a proxy between the RMS of the initiator and the new RMS. The following events can occur when the Request Handler forwards the communication request.

Event 4.1: Communication forwarded successfully

If the Request Handler has successfully forwarded the communication, it receives a response from the new RMS that contains an interaction scheme. It forwards that interaction scheme to the Interaction module and goes into the monitoring state.

Event 4.2: Forwarding failed

When something goes wrong while the Request Handler is forwarding communication (e.g. the new RMS does not respond), the Request Handler determines that the communication request failed, thus it instructs the Interaction module to inform the initiator and goes into the 'Starting communication request' state to wait for a new communication request.

4.2.2 Incoming communication

This section explains how a Request Handler deals with an incoming communication request. Note that the Priority manager and the Threshold manager that are described in section 3.2 are subcomponents of the Request Handler. The Request Handler uses these subcomponents to determine whether it should (still) notify the recipient. Figure 24 shows the state diagram of the Request Handler for dealing with incoming communication[33]. The following subsections explain each state in detail.

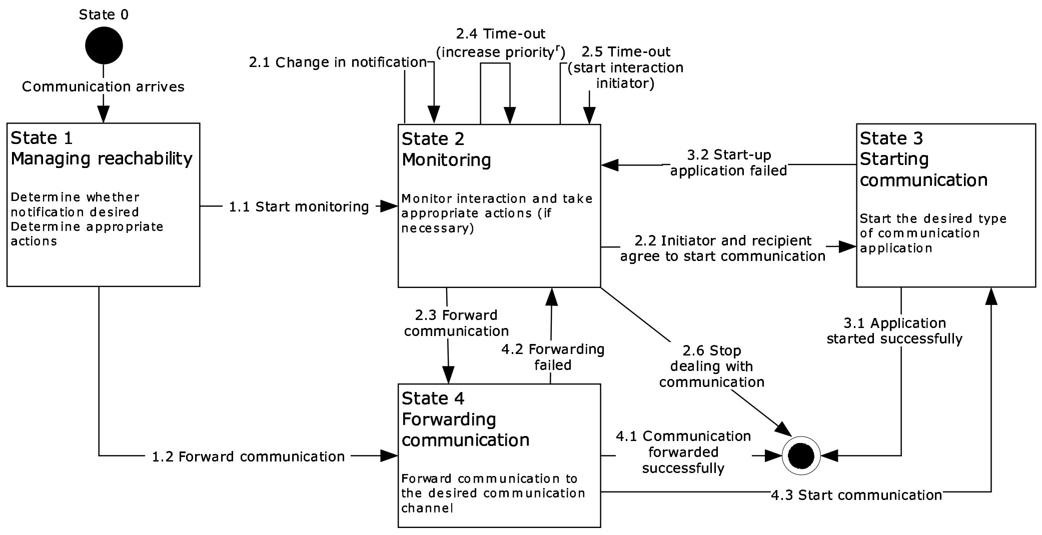

State 0
Communication arrives
State 1
Managing reachability
Determine whether notification desired
Determine appropriate actions
1.1 Start monitoring
1.2 Forward communication
2.1 Change in notification
2.4 Time-out (increase priority)
2.5 Time-out (start interaction initiator)
State 2
Monitoring
Monitor interaction and take appropriate actions (if necessary)
3.2 Start-up application failed
2.2 Initiator and recipient agree to start communication
State 3
Starting communication
Start the desired type of communication application
2.3 Forward communication
4.2 Forwarding failed
2.6 Stop dealing with communication
3.1 Application started successfully
State 4
Forwarding communication
Forward communication to the desired communication channel
4.1 Communication forwarded successfully
4.3 Start communication

Figure 24, State diagram 'Deal with incoming communication'

State 0

When the Request Handler must deal with a communication request, it goes into the 'Managing Reachability' state.

[31] Only the Request Handler of the *recipient* might need to forward non real-time communication, e.g. forward an e-mail message to another e-mail address.
[32] For privacy reasons, the RMS of the recipient might not provide the address of the new RMS.
[33] The events and states have a number to be able to reference them easily. This number does not indicate the order of occurrence.

State 1: Managing reachability
When the Request Handler is in this state, it determines the reachability of the recipient. In other words, it determines which types of communication the recipient is willing to start, and which types of communication it can start in the home system of the recipient. Based upon the reachability of the recipient, it determines whether it should notify the recipient of the communication request. Section 3.1 explains how the Request Handler determines whether it should notify the recipient. If it determines that it should not notify the recipient, it determines the appropriate action. Section 3.5 explains how the Request Handler determines the appropriate action. The following events can occur:

Event 1.1: Start monitoring
The following situation might cause the Request Handler to start monitoring:
- If the Request Handler determines that it should notify the recipient, it starts an Interaction module and instructs that module to notify the recipient. It provides all information that the Interaction module needs to start interaction with the recipient, e.g. the types of communication that it can start, the priority[r] of the communication, the identity of the recipient, and the identity of the initiator. The Interaction module takes care of the negotiation between the initiator and the recipient. The Request Handler goes into the monitoring state to monitor the progress of the notification and to wait for the result of the negotiation.
- If the Request Handler determines that it should not notify the recipient, it might determine that it should start interaction with the initiator. It starts an Interaction module and instructs that module to start interaction with the initiator. It provides all information that the Interaction module needs to start interaction, e.g. the types of communication that the recipient *is* willing to start, and the minimum priority[r] that communication needs such that the recipient is available for that communication. The Interaction module takes care of the negotiation between the initiator and the recipient. The Request Handler goes into the monitoring state to wait for the result of the negotiation.
- If the Request Handler determines that it should not notify the recipient, it might determine that it should wait until it can start notification. The Request Handler goes into the 'Monitoring' state.

Event 1.2: Notification is not desired (forward communication)
If the Request Handler determines that it should not notify the recipient, it might determine that it should forward the communication. The Request Handler goes into the 'Forwarding communication' state.

State 2: Monitoring
The Request Handler is in the 'Monitoring' state when it waits for the result of the negotiation between the initiator and the recipient, or when it waits until it can start notification. The following events can occur:

Event 2.1: Change in notification
The Request Handler not only monitors whether the recipient is aware of the communication in time, but it also monitors whether it should (still) notify the recipient. If it should not notify the recipient anymore (e.g. because the user status changes to 'Not available'), it instructs the Interaction module to stop the notification and keeps on monitoring. If it should start notifying the recipient (e.g. because the user status changes to 'Available' or because the Priority manager has increased the priority[r]), it instructs the Interaction module to start notification. In both situations, the Request Handler stays in the monitoring state.

Event 2.2: Initiator and recipient agree to start communication
When the Interaction module determines that both the initiator and recipient agree to start a particular type of communication, it informs the Request Handler such that the Request Handler can start the corresponding communication application. The Request Handler goes into the 'Starting communication' state.

Event 2.3: Forward communication
There are two situations that cause the Request Handler to forward communication:
- The Interaction module might propose to the initiator to forward the communication. If the initiator agrees, the Interaction module instructs the Request Handler to do so. The Request Handler goes into the ‘Forwarding communication’ state.
- While the Request Handler is monitoring, it might determine that it should forward the communication. In this situation, it instructs the Interaction module to inform the initiator that the communication is going to be forwarded. The Request Handler goes into the ‘Forwarding communication’ state.

When the Request Handler is in the ‘Monitoring’ state, a time-out can occur because the recipient is not aware of the communication in time. The Request Handler determines the appropriate action. Section 3.5 explains how the RMS determines the appropriate action.

Event 2.4: Time-out (Increase priority^r)
The Priority manager (which is a subsystem of the Request Handler) might increase the priority^r of the communication. If the Interaction module is notifying the recipient, the Request Handler informs the Interaction module of the new priority^r. Note that a change in the priority^r might cause the Request Handler to start notification. The Request Handler stays in the ‘Monitoring’ state.

Event 2.5: Time-out (start interaction initiator)
The Request Handler might determine that it should inform the initiator that the recipient is not aware of the communication in time. If the communication request regards real-time communication, the time-out occurs for instance 15 seconds after the arrival of the communication request, since an initiator that wants to start real-time communication does not want to wait very long for a response from the RMS of the recipient. The Request instructs the Interaction module to start interaction with the initiator and stays in the ‘Monitoring’ state.

Event 2.6: Stop dealing with communication
The following situation cause the Request Handler to stop dealing with the communication request:
- The Request Handler determines that it can stop dealing with the request. For example, if the communication request regards email communication, the Request Handler subscribes with the email application of the home system of the recipient, to be notified when the recipient starts using that application. When the Request Handler is notified that the user started that application, the Request Handler does not need to deal with the communication request anymore, since the application already shows the new message to the recipient. The Request Handler instructs the Interaction module to stop the notification and stops dealing with the communication request.
- The Interaction module can determine that it is not necessary to deal with the communication request anymore (for example, because the initiator has cancelled the request). It informs the Request Handler that it can stop dealing with the communication request.

State 3: Starting communication
When the Request Handler is in this state, it must start communication. Event 3.1 corresponds with event 3.1 that is explained in section 4.2.1. Event 3.2 corresponds with event 3.2 that is explained in section 4.2.1.

State 4: Forwarding communication
When the Request Handler is in this state, it must forward communication. The Request Handler first determines whether the communication must be transformed. If so, it asks the Transformation module to take care of this. For example, the Transformation module might transform an email message into an SMS message such that the Request Handler can forward it to the mobile phone of the recipient. The following events can occur.

Event 4.1: Communication forwarded successfully
The following situation can occur:

- The Request Handler has successfully forwarded non real-time communication. It informs the Interaction module that there is no need for any interaction anymore, and it stops dealing with the communication request.
- The Request Handler must forward a communication request to another RMS. In this situation, it either provides the address of that RMS to Request Handler of the initiator, or it starts acting as a proxy between the Request Handler of the initiator and the other RMS. If it must start to act as a proxy, it stays in the 'Forwarding state' until it does not need to act as a proxy anymore. If it provides the address of the other RMS to the Request Handler of the recipient, it has successfully forwarded the communication and stops dealing with the communication request.

Event 4.2: Forwarding failed
If the Request Handler cannot forward the communication, there are two possible situations. If the communication is not real-time, the Request Handler goes into the 'Monitoring' state to determine the appropriate action, e.g. it might start interaction with the initiator. If the communication is real-time, the Request Handler instructs the Interaction module to start interaction with the initiator and goes into the 'Monitoring' state.

Event 4.3: Start communication
The Request Handler might need to forward real-time communication to a device out-side the home network, that does not have a RMS installed. This means that it must start the communication that the initiator wants to start, between the device that the initiator is using and a device out-side the home network of the recipient. For example, it must forward a telephone call to the mobile phone of the recipient (which in this scenario does not have a RMS installed). In this situation, it goes into the 'Start communication' state to start the communication between the two devices

4.3 INTERACTION MODULE

The Interaction module takes care of the interaction between a person and a RMS, and the negotiation process between (the RMS of) the initiator and the RMS of the recipient. This section deals with the situation that both the initiator and the recipient of the communication have a RMS. Section 4.10 explains the situation that either the initiator or the recipient does not have a RMS. The Request Handler starts an Interaction module when it needs to start interaction with a person. The Request Handler stops the Interaction module when interaction is no longer necessary. Figure 25 illustrates that the Interaction module might start two User Interface modules. For example, the Interaction module that is part of the RMS of the recipient, might start a User Interface module to negotiate with an initiator who does not have a RMS (e.g. an User Interface module that is able to start interaction via a telephone connection), and it might start a User Interface module to notify the recipient (e.g. a User Interface module that uses the UI capabilities of the home system of the recipient, to create a UI).

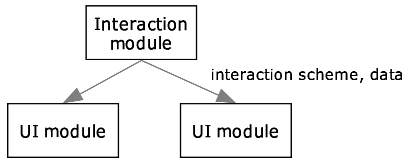


Figure 25 An Interaction module might start multiple User Interface modules

If an Interaction module must start interaction with a person via a User Interface module, it starts the User Interface module, constructs an interaction scheme, and sends it to the User Interface module such that this module can start the UI. If the Interaction module needs to start interaction with an initiator who has a RMS, it sends the interaction scheme to that RMS[34]. An interaction scheme consists of the following parts:

- The information that must be transferred, e.g. the Interaction module can inform the recipient of a call, of the identity of the initiator and the priority[f] of the communication.
- The options that a person can choose from, e.g. a recipient can choose to start communication or she can choose to propose a different type of communication to the initiator.

For most tasks, the operation of the Interaction module is quite straightforward. For example, when the Request Handler instructs the Interaction module to propose a new preference to a user, the Interaction module constructs an interaction scheme, and starts a User Interface module that can construct the appropriate UI. The interaction scheme contains two options: the user can either accept or to reject the preference. The Interaction module sends the result to the Request Handler. There are two tasks that are more complicated: to start interaction with a user who wants to start communication (the user the initiator of the communication), and to start interaction to notify a user of a communication request (the user is the recipient of the communication). The following two subsections explain the behaviour of the Interaction module for these two tasks.

4.3.1 Starting communication

When the user of the RMS wants to start communication, this user is the initiator of the communication. The Interaction module takes care of the interaction with initiator, and the interaction with the external party (the recipient)[35]. Figure 26 shows the operation of the Interaction module when it deals with this task[36]. The states are explained in detail in the subsections that follow.

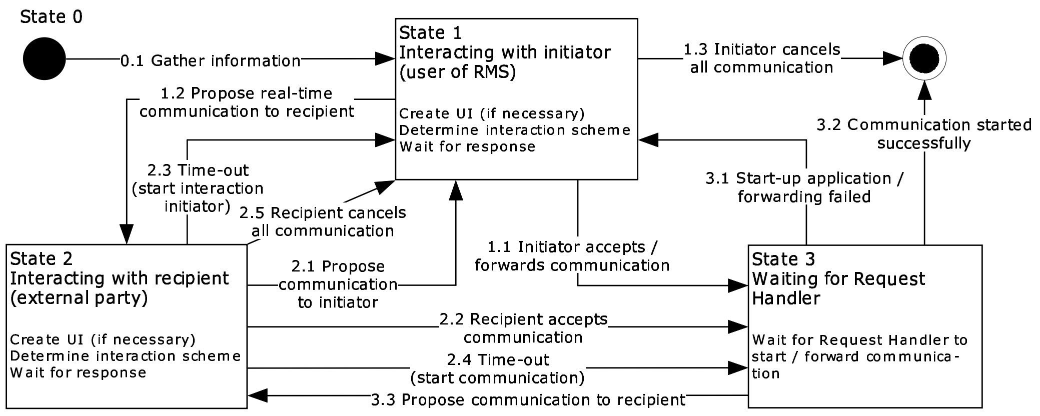

State 0
0.1 Gather information
State 1
Interacting with initiator (user of RMS)
Create UI (if necessary)
Determine interaction scheme
Wait for response
1.3 Initiator cancels all communication
1.2 Propose real-time communication to recipient
3.2 Communication started successfully
3.1 Start-up application / forwarding failed
2.3 Time-out (start interaction initiator)
2.5 Recipient cancels all communication
1.1 Initiator accepts / forwards communication
State 2
Interacting with recipient (external party)
Create UI (if necessary)
Determine interaction scheme
Wait for response
2.1 Propose communication to initiator
State 3
Waiting for Request Handler
Wait for Request Handler to start / forward communica-tion
2.2 Recipient accepts communication
2.4 Time-out (start communication)
3.3 Propose communication to recipient

Figure 26, State diagram 'Starting communication'

State 0

State 0 indicates that a Request Handler asks the Interaction module to gather all information that is necessary to send a communication request (e.g. the type of communication and the identity of the recipient). The Interaction module starts a User Interface module and sends the interaction scheme to that module. Next, it goes into the 'Interacting with initiator' state. The interaction scheme is explained in detail in Appendix III (Section III.I, Scheme 4).

[34] In fact, the Interaction module sends an interaction scheme to the Request Handler. The Request Handler forwards the scheme to the Request Handler of the remote RMS. Section 6.6.3 explains the interaction in detail.

[35] In this section, I assume that the recipient is located in a different home system. The situation that the recipient is located in the same WWICE system is the same to a large extent. However, in this situation the RMS must start two Request Handlers. Chapter 6 deals with the situation that there are multiple Request Handlers that are running as part of the same RMS.

[36] The events and states have a number to be able to reference them easily. This number does not indicate the order of occurrence.

State 1: Interacting with initiator
When the Interaction module of the RMS of the initiator is in this state, this means that it sent an interaction scheme to a User Interface module. It might have constructed this interaction scheme itself (e.g. to gather information for a communication request), or it might have forwarded an interaction scheme that it received from the Interaction module of the RMS of the recipient (e.g. an interaction scheme that informs the initiator that her communication request is denied). If the Interaction module is still gathering information for a communication request, it interacts with the Request Handler to determine whether the Request Handler has enough information to send the request. The Interaction module waits for the response from the initiator. The following events can occur:

Event 1.1: Initiator accepts / forwards communication
Three situations are possible:

- If the initiator decides to start *non real-time* communication and the recipient and the initiator were already negotiating, the Interaction module constructs an interaction scheme to inform the recipient, and sends it to the RMS of the recipient. This interaction scheme does not contain any actions[37]. Next, it instructs the Request Handler to start the communication.
- If the initiator accepts *real-time* communication that is proposed by the recipient, the Interaction module instructs the Request Handler to start the appropriate application.
- If the initiator decides to forward the communication, the Interaction module instructs the Request Handler to take care of the forwarding, and goes into the 'Waiting for Request Handler' state.

In all situations, the Interaction module goes into the 'Waiting for Request Handler' state.

Event 1.2: Propose real-time communication to recipient
The initiator decides to propose a particular type of real-time communication to the recipient. The Interaction module constructs an interaction scheme to propose this communication to the recipient (Appendix III, Section III.I, Scheme 5). If this is the first proposal, the Interaction module sends the interaction scheme to the Request Handler such that it can send a communication request. If the proposal is part of a negotiation between the initiator and (the RMS of) the recipient, the Interaction module sends the interaction scheme to the RMS of the recipient[38]. In both situations, it goes into the 'Interacting with recipient' state.

Event 1.3: Initiator cancels all communication
The initiator cancels the communication request. If the recipient and the initiator were negotiating, the Interaction module constructs an interaction scheme and sends it to the RMS of the recipient to inform the recipient of the cancellation. This scheme does not contain any actions. The Interaction module informs the Request Handler that no further actions are necessary and it stops the interaction.

State 2: Interacting with recipient
When the Interaction module is in this state, this means that the initiator proposes to the recipient to start a particular type of communication (possibly as part of a negotiation process). The Interaction module is waiting for the response of (the RMS of) the recipient. The following events can occur:

Event 2.1: Propose communication to initiator
The Interaction module receives an interaction scheme from the RMS of the recipient (Appendix III, Section III.I, Scheme 2). This scheme proposes a different type of communication than the initiator proposes. The Interaction module sends this scheme to the User Interface module and goes into the 'Interacting with recipient' state.

[37] Of course, the recipient can decide that she *does* want to start real-time communication. The Interaction module of the recipient will handle this as a new communication request.
[38] As mentioned before, the Interaction module sends this scheme via the Request Handler.

Event 2.2: Recipient accepts communication
The recipient accepts the communication that is proposed by the initiator. The Interaction module instructs the Request Handler to start the appropriate communication application and goes into the 'Waiting for Request Handler' state.

Event 2.3: Time-out (start interaction initiator)
The RMS of the recipient is not responding. The Request Handler has determined it should start interaction with the initiator, thus it instructs the Interaction module to start interaction with the initiator. The Interaction module constructs an interaction scheme to inform the initiator that the RMS of the recipient does not respond, and to propose appropriate actions (Appendix III, Section III.I, Scheme 6). The Interaction module goes into the 'Interacting with initiator' state.

Event 2.4: Time-out (start communication)
The RMS of the recipient is not responding. The Request Handler has determined that it should start the communication that the initiator wants to start, without consulting the RMS of the recipient. The Interaction module sends an interaction scheme to the User Interface module to inform the initiator that the RMS of the recipient is not responding and that the RMS is going to start the communication. Next, it goes into the 'Waiting for Request Handler' state to wait for the results of the Request Handler.

Event 2.5: Recipient cancels all communication
If the Interaction module receives this event, this means that the recipient rejects all communication. The Interaction module sends the scheme that it received from the RMS of the recipient to the User Interface module (section 4.3.2, event 1.3). Next, it goes into the 'Interacting with initiator' state.

State 3: Waiting for Request Handler
When the Interaction module is in this state, this means that the Request Handler is starting a communication application, or the Request Handler is forwarding communication. The following events can occur:

Event 3.1: Communication started successfully
If the Interaction module receives this event, this means that the Request Handler has successfully started the appropriate communication application. The Interaction module does not need to do anything anymore and thus it stops interaction.

Event 3.2: Start-up application / forwarding failed
If the Interaction module receives this event, this means that the Request Handler failed to start the appropriate communication application, or to forward the communication. The Request Handler instructs the Interaction module to propose other actions to the initiator (Appendix III, Section III.I, Scheme 2).

Event 3.3: Propose communication to recipient
If the Request Handler must forward a communication request to a new RMS, this is in fact a new communication request. It asks the Interaction module to construct a new interaction scheme. The Interaction module constructs an interaction scheme, and sends it to the RMS of the recipient (Appendix III, Section III.I, Scheme 5). Next, the Interaction module goes into the 'Interacting with recipient' state.

4.3.2 Incoming communication

When the Request Handler needs to take care of incoming communication, it can determine to start interaction with the recipient (the user of the RMS) to notify her of the communication, or to start interaction with the initiator (the external party) to inform her that the recipient is not reachable. It asks the Interaction module to take care of the interaction. Figure 27 shows the state diagram that explains the operation of the Interaction module when it deals with incoming communication[39]. The states are explained in detail in the subsections that follow.

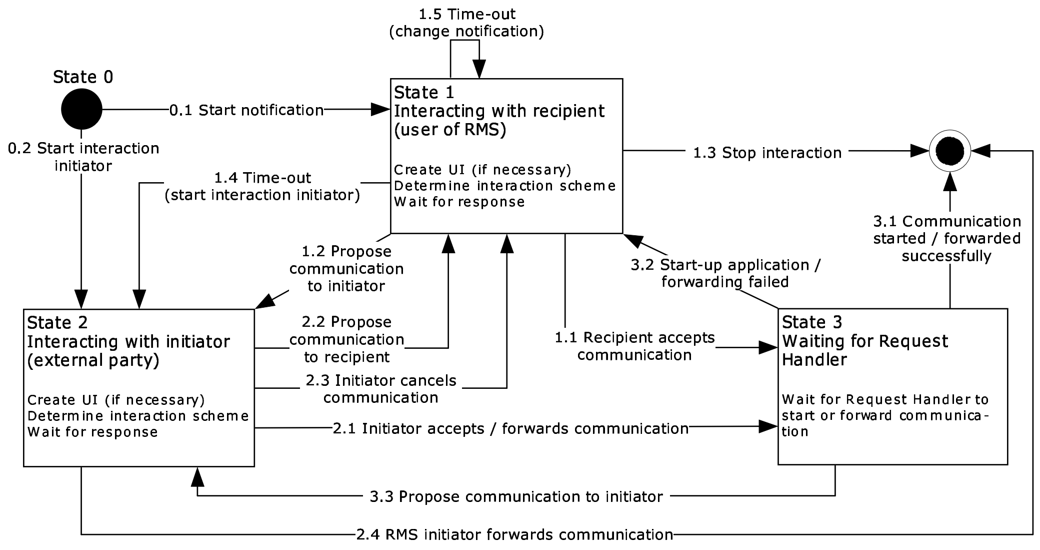

1.5 Time-out (change notification)
State 0
0.1 Start notification
State 1
Interacting with recipient (user of RMS)
Create UI (if necessary)
Determine interaction scheme
Wait for response
1.3 Stop interaction
0.2 Start interaction initiator
1.4 Time-out (start interaction initiator)
3.1 Communication started / forwarded successfully
1.2 Propose communication to initiator
3.2 Start-up application / forwarding failed
State 2
Interacting with initiator (external party)
Create UI (if necessary)
Determine interaction scheme
Wait for response
2.2 Propose communication to recipient
1.1 Recipient accepts communication
State 3
Waiting for Request Handler
Wait for Request Handler to start or forward communication
2.3 Initiator cancels communication
2.1 Initiator accepts / forwards communication
3.3 Propose communication to initiator
2.4 RMS initiator forwards communication

Figure 27, State diagram 'Dealing with incoming communication'

State 0: Starting point
State 0 is the 'idle' state. This means that the Interaction module is not dealing with any interaction. Two events can occur: the Request Handler can ask the Interaction module to start notification or to start interaction with the initiator of the communication. These events are explained below.

Event 0.1: Start notification
The Request Handler has determined that it should start notification. It instructs the Interaction module to take care of this. The Interaction module starts a User Interface module, constructs an interaction scheme, and sends it to the User Interface module. Next, it goes into the 'Interacting with recipient state. Appendix III, section III.I explains the interaction scheme in detail (Scheme 1). As soon as the recipient starts interaction, the Interaction module informs the Request Handler that it does not need to monitor the notification anymore.

Event 0.2: Start interaction with initiator
The Request Handler has determined that it should start interaction with the initiator. It instructs the Interaction module to take care of this. The Interaction module constructs an interaction scheme and sends it to the RMS of the initiator (Appendix III, Section III.I, Scheme 2). Next, the Interaction module goes into the 'Interacting with initiator' state to wait for the response of the (RMS of the) initiator.

State 1: Interacting with recipient
When the Interaction module is in this state, this means that the User Interface module is trying to notify the recipient (the user of the RMS) of new communication, or that the User Interface module is already interacting with the recipient. The following events can occur:

Event 1.1: Recipient accepts communication
The recipient accepts the communication that the initiator proposes. The Interaction module informs the RMS of the initiator, asks the Request Handler to start the appropriate communication application, and goes into the 'Waiting for Request Handler' state.

[39] The events and states have a number to be able to reference them easily. This number does not indicate the order of occurrence.

Event 1.2: Propose communication to initiator
The recipient wants to negotiate a new type of communication with the initiator. The Interaction module creates an interaction scheme and sends it to the RMS of the initiator (Appendix III, Section III.I, Scheme 2). Next, it goes into the 'Interacting with initiator' state to wait for the response of the initiator.

Event 1.3: Stop interaction
There are two reasons why the Interaction module might stop the interaction:
- The Request Handler has determined that notification is not necessary anymore, e.g. because the Request Handler has forwarded the communication. The Interaction module instructs the User Interface module to stop the UI and stops the interaction.
- The recipient cancels all communication. If the recipient and the initiator were negotiating, the Interaction module constructs an interaction scheme to inform the initiator that the recipient is not available, and sends it to the RMS of the initiator. This scheme only informs the initiator that the recipient is not available. It does not contain any actions. In both situations, the Interaction module informs the Request Handler that no further actions are necessary, it instructs the User Interface module to stop the UI, and it stops the interaction.

Event 1.4: Time-out (start interaction with initiator)
The Request Handler has detected that the recipient is not aware of the communication in time, and it has determined that it should start interaction with the initiator. It instructs the Interaction module to take care of this. The Interaction module constructs an interaction scheme, sends it to the User Interface module, and goes into the 'Interacting with initiator' state (Appendix III, Section III.I, Scheme 3).

Event 1.5: Time-out (change notification)
The Request Handler has determined that it should (temporarily) start or stop notification, or it has increased the priority[r] of the communication. The Interaction module informs the User Interface module and stays in the 'Interacting with recipient' state.

State 2: Interacting with initiator
When the Interaction module is in this state, this means that the Interaction module is waiting for a response from the initiator upon an interaction scheme that it sent to the RMS of the initiator. The following events can occur:

Event 2.1: Initiator accepts/ forwards communication
There are two situations that cause this event to happen:
- The initiator wants to start the type of communication that the (RMS of the) recipient proposes. The Interaction module instructs the Request Handler to start the communication and goes into the 'Waiting for Request Handler' state.
- The initiator wants to forward the communication and the Request Handler of the recipient must take care of the forwarding (e.g. it must act as a proxy). The Interaction module instructs the Request Handler to forward the communication and goes into the 'Waiting for Request Handler' state.

Event 2.2: Initiator proposes new communication to recipient
The initiator proposes a new type of communication to the recipient. Prior to this proposition, the (RMS of the) recipient has rejected the communication that the initiator proposed (e.g. the initiator proposed 'video communication). Now, the initiator proposes a different type of communication (e.g. 'telephone communication'). The Interaction module of the RMS of the recipient receives an interaction scheme from the RMS of the initiator (Appendix III, Section I.III, Scheme 5). It sends this scheme to the User Interface module and goes into the 'Interacting with recipient' state.

Event 2.3: Initiator cancels communication
If the Interaction module receives this event, this means that the initiator does not want to communicate anymore, or that the initiator decides to start non real-time communication. In both situations, the Interaction module receives an interaction scheme from the RMS of the

initiator. This interaction scheme informs the recipient what action the initiator undertakes. The Interaction module sends this scheme to the User Interface module and goes into the 'Interacting with recipient' state to wait for the response of the recipient.

Event 2.4: RMS initiator forwards communication
If the initiator decides to forward communication, and the Request Handler of the initiator takes care of this, the Interaction module of the RMS of the recipient informs the Request Handler that it can stop dealing with the communication request. If the initiator and the recipient were negotiating, the Interaction module constructs an interaction scheme to inform the recipient that the initiator has chosen to forward the communication request. Next, it stops the interaction.

State 3: Waiting for Request Handler
When the Interaction module is in this state, this means that the Interaction module is waiting for the Request Handler to start communication or to forward communication. The following events can occur

Event 3.1: Communication started / forwarded successfully
The Request Handler has successfully started or forwarded communication. The Interaction module does not need to do anything anymore and thus it stops the interaction.

Event 3.2: Start-up application / forwarding failed
The Request Handler failed to start the appropriate communication application, or to forward the communication. The Request Handler has determined that it should start interaction with the recipient (e.g. it determines to resume the notification of the recipient after it failed to forward the communication) The Interaction module constructs an interaction scheme, and sends it to the User Interface module (Appendix III, Section III.I, Scheme 5).

Event 3.3: Propose communication to initiator
The Request Handler failed to forward communication and it determined that it should start interaction with the initiator. The Interaction module to constructs an interaction scheme to propose other options to the initiator and sends it to the RMS of the initiator (Appendix III, Section III.I, Scheme 2). Next, the Interaction module goes into the 'Interacting with initiator' state.

4.4 USER INTERFACE MODULE

A User Interface module takes care of the UI of the RMS. The Interaction module starts a User Interface module when it needs to construct a UI (e.g. to notify a person that there is a telephone call), and it stops the User Interface module when it does not need its' services anymore. There are two types of User Interface modules that each provides a particular type of interaction. The first type is the 'internal' User Interface module that uses the UI capabilities of the home system where the RMS is installed, to create a UI. The second type is the 'external' User Interface module. This User Interface module is actually not a component of the RMS, but it is an external component that provides some additional UI-functionality that extends the interaction-possibilities of the RMS. For example, there might be an external User Interface module that is able to interact with someone using a telephone connection. This section does not deal with the external User Interface module in more detail, since the operation of this type of module is not in the scope of this graduation project. This section deals with the User Interface module that uses the UI capabilities of the home system to create a UI. There are four situations that require this User Interface module to construct a UI:

1. A user of the RMS wants to start communication.
2. A user of the RMS wants to adjust preferences.
3. A user of the RMS wants to indicate her status.
4. The Interaction module must notify a user.

In the first three situations, the User Interface module hardly needs to adapt the UI. It has to personalize the UI (e.g. adjust font size) and it might adjust the UI to the activities that the user is involved in (e.g. use speech recognition when the user is cooking). The fourth situation is a bit more complicated. In this situation, the User Interface module needs to adapt the UI such that on the one side, the user is not disturbed by the notification, and on the other side the user is aware of the notification in time. This graduation project considers two aspects of a UI that the User Interface module should adapt: the device(s) that it uses for the UI of the notification, and the obtrusiveness of the UI. Section 3.3 explains how the User Interface module determines which device(s) it must use for the notification and section 3.4 explains how the User Interface module determines the ‘obtrusiveness’ of the UI.

The Interaction module provides the content of the UI (the interaction scheme) and other data that the User Interface module can use to determine the UI, e.g. the identity of the user for whom the User Interface module must create a UI. Furthermore, the User Interface module retrieves the user preferences from the Preferences module. The User Interface module uses all this information to create UI descriptions. These UI descriptions fix the UI. Figure 28 shows an example of a UI description.

```
UI-widget = button
Data = { 'button-properties'}

UI-widget = sound
Data = {  'sound-properties' }
```

Figure 28 Example of the UI description of an icon

Figure 28 shows that a UI description consists of the following set of tupels: {UI widget, data}. The ‘UI widget’-item indicates a particular type of UI building block, e.g. a button. The ‘data’-item gives more information with respect to the UI widget, e.g. it indicates the colour. If the User Interface module must use an icon for the UI, it uses an icon that shows all the information that it must show according to the interaction scheme. Figure 29 illustrates an example.

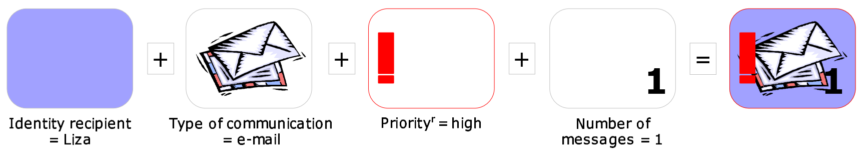


Figure 29 Example of an icon

Section 6.2 explains how the User Interface module uses UI descriptions to create a UI in the WWICE system.

4.5 STATUS MODULE

The Status module determines the status of the users of the RMS. The Status module implements the Status manager that is explained in detail in section 3.2.3. The Status manager is not part of the Request Handler, like the Priority manager and the Threshold manager, since the status of a user does not depend on communication requests.

The Status module is started by a component that needs a status. This component first determines whether there is a Status module running. If it cannot find a Status module, it starts a new Status module. Next, it subscribes with the Status module to be notified of changes in the status of a particular user. When another component needs the services of a Status module, it subscribes with the same Status module. When a component does not need the status anymore, it unsubscribes with the Status module. The Status module keeps on running until it has no more subscriptions. Probably, the RMS will continuously show the status to the user (just like ICQ and MS ® Messenger). In this situation, the Status module is running continuously. The information that the Status module provides is of interest to several components of the RMS. Therefore, only one instance of a Status module per RMS is sufficient. Figure 30 illustrates that the Status module might send status updates to several components of the RMS.

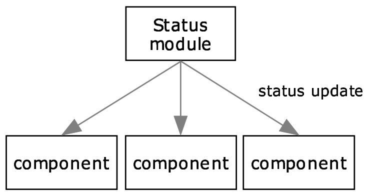


Figure 30 Status module might provide status information to multiple components

In order to determine the status, the Status module retrieves the preferences that it needs from the Preferences module. The Status module might use the activities that a user is involved in, to determine the status of the user. The Status module retrieves this information from the Activity Monitor that is explained in section 6.6.7. Since the Status module implements the Status manager, and section 3.2.3 explains the Status manager in detail, there is no need to explain the Status module in more detail.

4.6 INFORMATION MODULE

An Information module provides a service that extracts information from available data. Every Information module provides one type of service. For example, there might be an Information module that provides an “Extract subject from text”-service. This Information module is able to extract the subject of a communication request from a text that it receives as input, and it determines whether this subject is equal to one of the subjects that are pre-defined by the user. A component of the RMS might need the service of an Information module in the following situations:

- When non real-time communication arrives at the home system of the recipient. For example, a new e-mail message arrives at the e-mail server. The component of the RMS that detects this message must construct a communication request for the Request Handler. It might need an Information module to obtain some of the information that it needs to construct the communication request. For example, it might need an Information module that is able to determine the identity of the initiator, based upon the e-mail address of the initiator.
- The Request Handler might need an Information module if it must deal with a communication request regarding real-time communication. For example, it might ask an Information module to determine whether the subject of the communication request is equal to one of the pre-defined subjects that are identified by the recipient.
- The User Interface module might need an Information module if it must deal with a communication request regarding real-time communication, and the initiator does not have a RMS. This situation is explained in section 4.10.

In order to extract information, an Information module might parse the data that it receives as input, in order to find specific, pre-defined keywords that correspond to a list of pre-defined topics. These keywords and topics are stored in the so-called Topic database. For example, if the Information module must extract the subject of a message from a text that it receives as input, the possible topics might be ‘Meeting’, ‘Girlfriend’, ‘Emergency’, and ‘Cancellation’. The pre-defined keywords that correspond to the topic ‘Meeting’, might be ‘meeting’, ‘appointment’, and ‘date’. The usefulness of this type of Information module depends to a large extent upon the willingness of the user to identify topics and keywords. Personally, I do not think that many people will do this, since it takes too much effort to identify these topics and keywords. However, the designers of the RMS might define useful topics and keywords in advance. This would improve the usefulness of an Information module.

Another example of an Information module is a module that offers an “Extract-identity-from-video” service. The input for this module is a video stream, and the output is the identity, or identities, of the person(s) that are shown by the video stream. This module might use pattern recognition or speech recognition to identify the person(s).

The Information module is not dealt with in more detail since there is no need for any adaptive behavior from the Information module. The module just indicates what service it provides and the components of the RMS can use this service when needed. This is a straightforward process that does not need further explanation. *How* the Information module implements this service is not in the scope of this project.

4.7 PREFERENCES MODULE

The Preferences module stores the preferences of (the users of) the RMS. The components of the RMS can request these preferences from the Preferences module when needed. The exact preferences that are stored at the Preferences module are not in the scope of this project (e.g. whether there is a preference ‘Adjust color’ for the UI of a notification). This document only indicates what type of preferences there should be (e.g. preferences to determine the priority[r]). User tests must determine what kind of preferences a user would like to indicate. All preferences that are of importance to the operation of the RMS are explained in the sections that deal with the functionality of the RMS that needs these preferences. For example, section 3.2.3 explains the preferences that the Status module needs to determine the status of a recipient. It is important to note that users do not *have* to indicate any preferences. The RMS can have default preferences that result in quite good system behavior. The sections that explain the different types of preferences also indicate the default preferences.

At start-up, the RMS starts a Preferences module. Appendix IV explains that the Preferences module might store some preferences at a different device than the device where the RMS is installed. Therefore, the Preferences module must manage the preferences. This means that the Preferences module determines how much memory and/or storage is available for preferences, and next it retrieves the most important preferences in advance such that these preferences are immediately available for components. An example of important preferences, are the preferences that components need in order to deal with real-time communication. The Learning module determines which preferences are important, since the Learning module keeps track of historic data that can be used to determine this information. The Learning module informs the Preferences module which preferences are important. The Preferences module stores this information in its’ preferences database. At start-up, the Preferences module retrieves this information, and uses it to manage the preferences.

The Preferences module is not dealt with in more detail since there is no need for any adaptive behavior from the Preferences module: the module just indicates what preferences it provides and other components of the RMS can use these preferences when necessary. This is a straightforward process that does not need further explanation. Note that the Learning module tries to extract new preferences, as is explained in section 4.9.

4.8 TRANSFORMATION MODULE

A Transformation module provides a service that can transform one type of communication into another (e.g. a Transform-voicemail-to-email service)[40]. Each Transformation module offers an interface that indicates the input that it needs to perform the transformation, and what kind of transformation it performs (e.g. a text-to-speech transformation). Appendix III,

[40] Note the difference between the Transformation module and the Information module. The Transformation module transforms one type of communication into another. The Information module tries to extract *new* information from the data that it receives as input.

section III.II explains some examples of the interface of a Transformation module. There are three situations that require a Transformation module:

Transform real-time communication into other real-time communication
When a Request Handler must start real-time communication, it might need a Transformation module to transform one type of real-time communication into another type of real-time communication. For example, if it must start video communication and the recipient is using a mobile device, the Transformation module might need to transform the mpeg-2 stream of the incoming video stream into an mpeg-4 stream that can be used in a wireless network[41].

Transform real-time communication into non real-time communication
When the Request Handler must record a message, e.g. it must record a voice-mail message or a video mail message, it needs a Transformation module to take care of this.

Transform non real-time communication into non real-time communication
If the Request Handler must forward a message, it might need a Transformation module to transform the message into another type of non real-time communication. For example, if the Request Handler wants to forward an e-mail message to a mobile phone, it needs a Transformation module to transform the e-mail message into a SMS message.

This chapter does not deal with the Transformation module in more detail since there is no need for any adaptive behavior from the Transformation module. The module just indicates what services it provides (e.g. a Voicemail-to-email service) and other components of the RMS can use these services when necessary. This is a straightforward process that does not need further explanation. *How* the Transformation module implements the service that it provides, is not in the scope of this project.

4.9 LEARNING MODULE

The Learning module is the component of the RMS that makes the system ‘intelligent’. The Learning module tries to determine new user preferences that make the system adapt to the user behavior. The Learning module might learn implicitly (the system observes the behavior of the user) and/or explicitly (the system executes automatic behavior and the user can indicate an appreciation-value). From the interviews results that the Learning module should propose a new preference to the user such that the user can indicate whether or not she wants that preference, if the preference changes the reachability of the user (e.g. the initiator ‘Dilbert’s Marketing company’ should be in the ignore-list). The Learning module should not automatically adjust user preferences that change the reachability of a user, since the user wants to have control over her reachability. If the system determines a preference that does not change the reachability of the user, the Learning module can automatically adjust the preference, e.g. if it determines a new search order to find devices that are to be used for communication.

In order to find new preferences, the Learning module constructs user profiles that model the behavior of a user. It is difficult to predict in advance how these user profiles should look like. This is typically something that one can only determine after some extensive user testing, since this way one can establish different patterns in user behavior. Anyway, the Learning module should contain for every preference, knowledge rules that determine what value that preference should have. Appendix III, section III.III deals with two examples. The Learning module subscribes with components of the RMS (and possibly even with entities that are part of the home system) to obtain the data that it needs. When the Learning module receives new information that it can use to determine some particular preference, it stores the data in the so-called History database and evaluates the corresponding knowledge rules to determine new preferences.

[41] Section 5.4 explains that within the WWICE system, the Graphmapper takes care that this type of transformation takes place.

The Learning module is not dealt with in more detail, since the operation of the Learning module depends to a large extent on the results of user tests that should be performed when the RMS is implemented.

4.10 OPERATION OF RMS WHEN RECIPIENT OR INITIATOR DOES NOT HAVE A RMS

This section explains the operation of the RMS when either the initiator or the recipient does not have a RMS. Whether or not the recipient and/or the initiator have a RMS is only of importance for *real-time* communication. If the initiator wants to start non real-time communication (e.g. email), she does not need a RMS: the can just start the appropriate communication application. If the recipient of non real-time communication does not have a RMS, this just means that there is no RMS that is able to start notification, or to forward the message to another communication channel. However, if the communication request regards real-time communication, there are two important situations that are dealt with in this section: either the recipient does not have a RMS, or the initiator does not have a RMS.

§ ***Recipient does not have a RMS***
If the user of a RMS wants to start real-time communication with someone who does not have a RMS, the Request Handler must deal with this situation. The Request Handler tries to set-up the communication by using the communication address of the recipient. For example, it uses the phone number of the recipient to set-up a telephone connection. When it turns out that it is not possible to start the communication (e.g. the recipient does not pick up the phone), it instructs the Interaction module to inform the initiator and to propose other communication (if possible). If it succeeds to start communication, it has accomplished its' task, thus it can stop dealing with the communication request of the initiator.

§ ***Initiator does not have a RMS***
This section explains how the RMS of a recipient deals with a communication request from an initiator who does not have a RMS. In this situation, a problem might occur when the Request Handler needs to start interaction with the initiator in order to obtain the information that it needs to determine whether it should notify the recipient. If the Request Handler must start interaction because the recipient rejected the communication request or the recipient did not respond to the notification, the Request Handler can just do nothing (if interaction with the initiator is not possible).

Start interaction with initiator
If the Request Handler must start interaction with the initiator, this is only possible if there is a User Interface module that is able to start interaction with the initiator. For example, if the initiator is using a telephone, there must be a User Interface module that can start interaction with a person via a telephone connection. This User Interface module receives interaction schemes from the Interaction module and uses these schemes for the interaction with the initiator. If the initiator provides particular information (e.g. the identity of the recipient), the User Interface module might transform this reply into text, e.g. "I would like to talk to Homer". The User Interface module needs an Information module that is able to extract the identity of the recipient from this text. Thus, if an initiator must provide information, there might be a need for an Information module that can extract the information that the User Interface module must obtain. Of course, the Interaction module might also propose different options to the initiator in order to determine the identity of the recipient. For example, it can construct an interaction scheme that contains three options that the initiator can choose: 'Homer', 'Liza', and 'Bart'. In this situation, there is not need for an Information module.
Note that if it is possible to start interaction with the initiator, this means that the Interaction module constructs interaction schemes for the User Interface module that interacts with the initiator, *and* it constructs interaction schemes for the User Interface module that interacts with the recipient (if interaction with the recipient is necessary).

Determine whether or not to start notification

If the Request Handler can start interaction with the initiator, it can determine whether it should start notification. If it cannot start interaction with the initiator, it must deal with this situation is good as possible. The Request Handler needs information to determine whether the recipient is available for communication, e.g. the identity of the initiator, the identity of the recipient, the priority^{i} of the communication, and the subject of the communication. The Request Handler can use Information modules to determine part of this information. For example, it can use an Information module that is able to determine the identity of the initiator based upon the telephone number of the initiator. The identity of the initiator might give information over the identity of the recipient, e.g. Mr. Burns always calls for Homer.

The Request Handler uses the information that it *does* have and it uses default values for the information that it does *not* have (e.g. it uses default values for the priority^{i} and the subject of the communication). If the Request Handler does not know the identity of the recipient (e.g. because the initiator is calling to a telephone that does not belong to one person, but to a whole family), and it cannot start interaction with the initiator, the Request Handler just starts notification such that the intended recipient does not miss an important call. Of course, the users of the RMS must be able to indicate other default behavior, e.g. they might indicate that the RMS must record a voice-mail if the identity of the recipient is unknown.

If the Request Handler determines that it should not notify the recipient (e.g. because the status of all users in the home is ‘Not available’), and it cannot start interaction with the initiator, it must undertake a different action. The Request Handler uses preferences to determine which action it should undertake. I distinguish the following actions:

- It could record the communication, e.g. it could record a voice mail.
- It could reject the communication. This means that it does not do anything, thus the initiator will give up after a while.
- It could reject the communication and wait for the initiator to call again. If the initiator calls again, the Request Handler determines that the priority^{r} is ‘Emergency’, and it starts notification (if a recipient wants a notification of communication with the priority^{r} ‘Emergency’).

5 WWICE

This chapter describes the WWICE project [7], [8]. The WWICE project is a project at Philips Research that explores new applications in the electronic market that offer more entertainment, comfort, and flexibility in a networked home environment. It focuses on new application concepts, the corresponding interaction concepts for easy access and control, and the system architecture needed to support these concepts. In particular, the system architecture has to deal with issues concerning distributed computing. Section 5.1 introduces the interaction- and application concepts. Section 5.2 explains the system stack of the WWICE system that introduces the WWICE services that are of importance to the RMS. Sections 5.3 to 5.8 explain these services in detail.

5.1 APPLICATION AND INTERACTION CONCEPTS

Figure 31 Sharing an activity[42]

This section explains the new application and interaction concepts that are demonstrated by the WWICE system. One important interaction concept is the *‘community’*-concept. This concept makes it easy to communicate, and to share and use digital content. A community is a group of users who might use different WWICE systems. Members of a community can interact with each other via their own WWICE system, in the following ways:

- Content sharing and communication: members can easily communicate (e.g. start telephone- or video communication) and share content (e.g. share pictures, documents, music, and movies).
- Shared activities (multi-user activities): members can experience one activity at different locations, at the same time. Figure 31 shows the Ambient Intelligence vision of a shared activity[42]: two persons are drawing a picture together. Each person uses her own, interactive screen. On this screen, they are able to see each other and the drawing that they are creating together.

One user can be a member of one or more communities. Within the WWICE system, a community is represented by a ‘Space’. A Space is a virtual place that can contain people and content. The Space Browser is the application concept that is the interface of the WWICE system to the user. The Space browser shows Spaces and enables the user to communicate with other persons that are also using the Space, and to start, edit, delete or add content. For example, in your ‘Family’-space you can see icons that represent persons that are currently using that Space.
The next interaction concept is the *‘activity’*. If a person is interacting with content and/or engaged in communication, this is denoted as an activity. Examples of activities are watching a movie, listening to a song, and having a telephone conversation. If two persons are engaged in a shared activity, this means that they are using the same functionality *and* the same content. For example, if they are watching a movie together (at different places), this is denoted as one activity. If one of these persons manipulates the movie (e.g. she pauses the movie), this is also

[42] The shared activity that is shown in Figure 31 is the Philips vision of a shared activity. The WWICE system has a different implementation of this activity.

visible for the other person. The activity concept improves the user's conceptual model of the home system by allowing users to deal with activities instead of separate tasks. This means that the user only needs to indicate that she wants to start a particular activity (e.g. start a telephone conversation), or join an activity (e.g. join someone in the activity 'watching a movie'). The WWICE system takes care of all actions that are necessary to enable the user to start or join an activity. For example, the user might indicate on a mobile device, that she wants to start the movie 'The Godfather I'. The WWICE system searches for the appropriate content (e.g. an AVI-file or a DVD-file), takes care of any necessary transformation (e.g. it might need to encode the AVI-stream into Mpeg4 in order to be able to send the data over a wireless network), and starts the appropriate application (e.g. a Media Player) at the appropriate device (in this case, the mobile device).

The last important interaction concept is the *'Linking'*-concept. Linking means that a user can start activities, using a (possibly dynamic) combination of portable and stationary devices. Furthermore, this concept gives the user the possibility to change this configuration of devices whenever she likes. For example, when someone is using a PDA and approaches a television, she can choose whether or not she wants the television to display the content that she activated on her PDA. The linking-concept makes it easy to interact with portable devices in combination with the devices in their environment.

5.2 WWICE SYSTEM STACK

This section introduces the entities of the WWICE system that can be used for the implementation of the RMS. The entities are introduced on the basis of the system stack of the WWICE system that is shown in Figure 32. A device does not need to have the whole system stack installed. For example, a device that has only storage capabilities has no need for the functionality that is provided by the third and fourth layer of the system stack. Sections 5.3 to 5.8 explain the entities that are introduced in this section in detail.

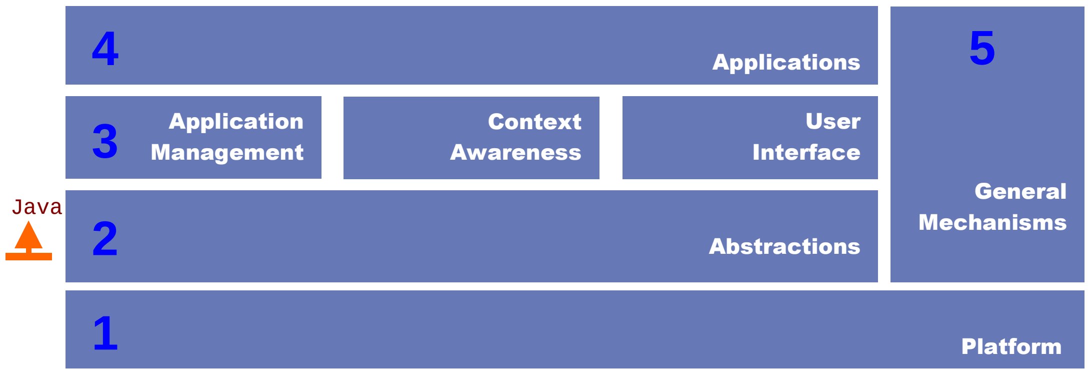


Figure 32 System stack of the WWICE system

The first layer of the system stack is the Platform layer. This layer consists of the hardware (e.g. hard disks, MPEG-encoders, and speakers) and platform software (e.g. low level drivers and operating system). The entities in the Platform layer are called *platform resources*.

The second layer is the Abstractions layer that consists of services. A service is an entity of the WWICE system that can provide a service to a client. Services are (remote) software objects with a type and an associated well-defined API that can be invoked by clients. Examples of services are resources, devices, spaces and persons. A 'resource' is a software object that represents a (set of) platform resource(s) in the Platform layer[43]. This software object offers an interface to the functionality of that resource. An example of a resource is an AVStorage. An AVStorage might represent a hard disk with streaming capabilities, and offers

[43] From now on, the term 'resource' is used to refer to a resource in the Abstraction layer, unless it is explicitly declared that the resource refers to a platform resource in the Platform layer.

store- and retrieve functionality for audio and video data. A ‘device’ is a software object that represents a physical device (e.g. the device that runs the WWICE system stack). This object takes care of creating the connections between resources, and it deals with the side effects that might occur when a particular resource is allocated. For example, a device that contains both an mpeg-4 decoder and an mpeg-2 decoder might not be able to use these two decoders at the same time. When one of these decoders is allocated for a particular application, the device object also allocates the other decoder such that WWICE subsystems know that this resource is not available. A ‘space’ is a software object that represents a Space. The Space is already introduced in section 5.1. Finally, a ‘person’ is a software object that represents a real-life person in the WWICE system. A person can have persistent attributes (e.g. the name of that person), as well as 'dynamic' attributes (e.g. the preferences of that person). The Abstractions layer also contains clusters. A cluster is a set of device-services that are logically grouped together, based on the location of the devices. For example, a cluster can contain all devices that are near a particular television. One could also define clusters based on a different property of a device, e.g. the capabilities of a device[44].

The third layer consists of three parts: the Application Management-, Context Awareness- and User Interface part. These parts contain WWICE subsystems. The most important subsystems of the Application Management part are the Application Manager and the Graphmapper. The Application Manager starts, given the content that is selected by the user, the corresponding application at the correct device. The Application Manager uses the Graphmapper to do this. The main subsystems of the Context Awareness part are the ‘Presence monitor’ and the ‘State monitor’. The Presence monitor mainly keeps track of the physical locations of persons and devices that are in the home. The State monitor monitors the state of ‘entities’. For example, the State monitor can monitor the state ‘location’ of the person ‘Homer’. The main subsystems of the User Interface layer are the ‘UI Manager’ and the ‘UI Mapper’. The UI Manager combines UI-descriptions from applications, in order to create one UI description for a particular device. The UI Mapper maps the UI-description of the UI Manager onto specific UI-descriptions for UI-resources.

The fourth layer is the Applications layer. This layer consists of applications that the users of the WWICE system can use, and system applications that provide some high-level functionality. An example of an application that the users of the WWICE system can use is the Space Browser that is introduced in section 5.1. An important system application is the UI Manager. Applications can query the UI Manager for the UI capabilities of the device that a UI Manager manages.

The ‘General Mechanisms’ support the whole system stack. These mechanisms take care of many problems that arise due to the distributed nature of the WWICE system. For example, they take care of resource discovery, URN resolving, and remote procedure calls.

5.3 APPLICATION MANAGER

A client[45] asks the Application Manager to start, given some content, the corresponding application at the correct device. The client indicates either the sink[46] or the (cluster of) device(s) that the Application Manager should use for the UI of the application. Furthermore, the initiating application must provide a URN that indicates what ‘content’ the Application Manager must start. For example, a URN might stand for a song (in this case, the Application Manager should start an application to play that song), or it might stand for a person (in this case, the Application Manager should start an application that allows the user of the WWICE system to communicate with this person[47]). The WWICE system uses URNs to be able to uniquely identify each individual entity (e.g. content and services). The system gives each

[44] At this moment, the WWICE system only uses clusters that are based upon location.
[45] The Space Browser is typically a client that asks the Application Manager to start an application.
[46] A sink is a resource that can receive data. For example, a Display-resource is a sink since it can receive data to display it on a screen.
[47] At this moment, the WWICE system automatically starts a video conferencing application when the URN stands for a person.

entity a unique, persistent name for referencing the entity, instead of using the location (URL) of the entity. In this way, the system obtains a de-coupling between references to entities and the actual instances of those entities.
Every application registers an application graph at the Application Manager. This graph indicates what resources an application needs. Figure 33 shows an example of the application graph of a Television application.

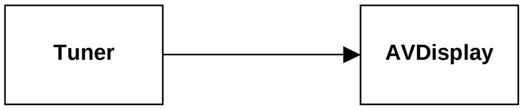


Figure 33 Example of an application graph of a Television application

The arrow between Tuner and AVDisplay shows that the Tuner-resource must be used as a source, and the AVDisplay resource must be used as a sink. A source is a resource that can provide data, and a sink is a resource that can receive data. The Application Manager asks the Graphmapper to test the realizability of an application graph. If the graph is realizable, the Application Manager asks the Graphmapper to realize the graph (e.g. to allocate all resources that the application needs). When the graph is realized, the Application Manager can start the application. For confidentiality reasons, this thesis does not provide more information concerning the Application Manager.

5.4 GRAPHMAPPER

A client (e.g. the Application Manager) can ask a Graphmapper to test, realize, and remove a graph[48]. When the Graphmapper tests a graph, this means that it checks whether it is possible to realize the graph and the ‘effort’ of this realization (e.g. bandwidth becomes scarce). When the Graphmapper realizes a graph, this means that it allocates the necessary resources (e.g. it allocates a tuner resource), initializes the resources (e.g. it sets a tuner to the correct content), and creates the necessary connections (e.g. it creates a TCP/IP connection between a tuner and an AVDisplay-resource). When the Graphmapper removes a graph, this means that it frees the allocated resources, breaks down the connection(s), and stops any device that it still running.
In order to test the realizability of a graph, the Graphmapper first maps the abstract resource descriptions from the application graph onto appropriate resources. Thus for each abstract resource, it chooses a real resource in such a way that all requirements are fulfilled. The Graphmapper tests for each resource whether it is available and whether it fulfills all the requirements. If not, it chooses a different resource for the abstract resource description, and tests that resource. The Graphmapper can realize a graph when it can find resources for all the abstract resource descriptions of the graph, when it is possible to allocate these resources, and when it is possible to create connections between the resources. In order to obtain the resources that provide the desired content, the Graphmapper sends the URNs that represent the desired content to the URN resolver. In order to obtain the resources that provide the UI, the Graphmapper asks the UI Manager to determine whether a particular (cluster of) device(s) can deal with UI of the application[49].

One of the requirements that resources must fulfill such that the Graphmapper can create connections between these resources, is that the resources must have compatible plugs. A plug describes whether a resource is a sink or a source, and the major and minor content type that the resource can handle. For example, suppose that the Graphmapper is testing the realizability of the graph that is illustrated in Figure 33, and the Tuner-resource only has an mpeg-2 source-plug available. Now, the Graphmapper must find an AVDisplay-resource that has the necessary UI-capabilities available and that has an available mpeg-2 sink-plug.

[48] A client talks to the Graphmapper that is installed on the same device as the client. The Graphmapper is in fact a distributed application. However, this is transparent to the client.
[49] Section 5.5 explains the UI Manager in detail.

The Graphmapper may enhance the graph that it got from the Application Manager. Figure 34 shows an example that enhances the graph that is shown Figure 33. The Graphmapper has introduced an mpeg-2-to-mpeg-4 transcoder (e.g. to decrease the required bandwidth or to obtain plug-compatibility).

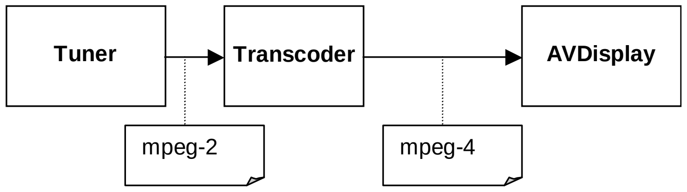


Figure 34 Example of an enhanced graph of a Media Player

5.5 UI MANAGER

Every device has one UI Manager that manages the UI-widgets of the resources of that device. At start-up, the UI Manager retrieves the list of widgets from the UI Mapper. The UI Manager provides the following functionality:

Manage resource
The UI Manager can allocate or de-allocate UI resources. If the Application Manager asks the UI Manager to allocate UI resources, it allocates widgets. It keeps track of the widgets that the application uses. If it must de-allocate the resources for a particular application, it de-allocates the appropriate widgets.

Provide UI-widgets
An application does not know beforehand which widgets are implemented on a certain device. It must query the UI Manager for the capabilities of the device. The UI Manager knows from the UI Manager, which UI-widgets are available for a particular application.

Combine UI's to create one UI for a device.
When an application wants to construct a UI, it sends a UI description to the UI Manager, based on the available UI capabilities of that device. This UI description contains among other things the UI-widgets with their corresponding minimum UI requirements. The UI Manager combines this UI with the UIs from the other applications in order to construct *one* UI for that device. It sends this UI to the UI Mapper such that the UI Mapper can map the UI to a resource-specific UI. The UI Mapper sends the resource-specific UI to the corresponding resource(s).

5.6 UI MAPPER

Every device has one UI Mapper. At start-up, the UI Mapper informs the UI Manager which UI widgets the resources in that device can provide. The UI Mapper determines which UI widgets a device can provide, based upon the UI widgets that are provided by the resources[50]. For example, an AVDisplay might provide a Video canvas-widget and Button-widgets. Furthermore, the UI Mapper takes care of the translation of the UI-description of the UI Manager onto specific UI-descriptions for a particular UI-resource. An example of such a mapping is that the UI Mapper determines the size of a button by taking into account the amount of pixels that are available at the physical resource (e.g. the physical display), and the standard distance of a user to that physical resource. The UI Mapper sends the resource-specific UI description to the resource such that this resource can create the UI.

[50] The UI Mapper might determine some 'composed widgets'. This is not explained in more detail.

5.7 PRESENCE MONITOR

The Presence monitor can provide the physical location and orientation of persons and devices. There can be multiple Presence monitors in one WWICE network (however, one would do). A client can subscribe with a Presence monitor to be notified of changes in the location of persons and devices. The Presence Monitor does not know the location with 100 % certainty: it appoints a certainty factor to every location. The Presence Monitor subscribes with services that can provide information about the physical locations or the orientation of devices and persons (e.g. resources that represent sensors). Whenever one of these services has some new information, it reports this to the Presence Monitor. The Presence Monitor adds this observation as a fact to its knowledge database, called the Presence Fact Base. After each addition of a new fact, the Presence Monitor uses an inference mechanism to look at the total content of the Presence Fact Base to draw new conclusions. Next, it sends a notification to its clients (if necessary).

5.8 STATE MONITOR

The State monitor monitors the state of ‘entities’. An entity can be a person or a device, but also an application. For example, the State monitor might monitor whether an application starts or stops. There can be multiple State monitors in one WWICE network. A client can subscribe at a State monitor with a (event, {conditions})-tupel. The ‘event’ is the event that the State monitor must use to determine when to evaluate the set of conditions. The set {conditions} is the set of conditions that should hold in order to cause the State monitor to notify the client. The State monitor subscribes with services that provide the information that it needs to determine when events occur, and to evaluate the conditions. For example, a client could subscribe with the event ‘Homer enters the kitchen’, and the condition ‘the time must be between 8:00h and 8:30h’. In this situation, the State monitor subscribes with the Presence monitor to be notified of changes in the location of the person ‘Homer’. If the location of the person ‘Homer’ becomes ‘Kitchen’, the State monitor evaluates the condition and sends a notification (if necessary).

6 ARCHITECTURE OF RMS IN WWICE

Chapter 4 explains how the components of a RMS cooperate to deal with one communication request. This chapter explains the deployment of these components, the deployment of RMS-es, and the operation of the RMS within the WWICE system. Furthermore, this chapter introduces two new aspects of a RMS. The first aspect is that a RMS must be able to deal with multiple tasks at the same time (e.g. multiple communication requests). Section 6.1 introduces the Task manager that deals with this aspect. The second aspect is that several components of a RMS might need to create a UI at the same device (e.g. one UI to notify a user of an incoming telephone call, and one UI to notify a user of new email). Section 6.2 introduces the Cluster manager that takes care of this aspect of a RMS. Section 6.3 explains the criteria for deployment. Section 6.4 explains the deployment of the components of the RMS. Section 6.5 explains the deployment of RMS-es in the WWICE system and finally, section 6.6 explains the operation of the RMS as part of the WWICE system.

6.1 TASK MANAGER

The RMS needs a Task manager to deal with multiple tasks at the same time, e.g. multiple communication requests. There is only one Task manager per RMS. For every task that a Task manager must deal with, it instructs the appropriate Request Handler to handle that task. If there is no appropriate Request Handler, the Task manager starts a new Request Handler. Figure 35 illustrates that the Task manager might start several Request Handlers to deal with tasks.

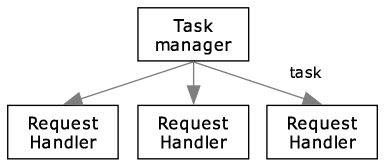


Figure 35 Task manager starts Request Handlers to deal with tasks

The main tasks that the Task manager deals with are ‘start communication’ and ‘handle communication request’. The task ‘start communication’ means that a user of the RMS wants to start communication with someone else (e.g. send an e-mail message). If the Task Manager must deal with this task, it starts a new Request Handler to execute the task. The task ‘handle communication request’ means that a communication request arrives at the RMS. If the request regards real-time communication, the Task manager starts a new Request Handler. If the request regards non real-time communication, it searches for a Request Handler that already handles that type of communication for the intended recipient of the communication. Thus the Task manager starts only one Request Handler per type of non real-time communication, per user. This Request Handler combines the information of all communication requests in order to create one notification per type of communication, per person. For example, if ten e-mail messages arrive for a particular person, the Task manager receives ten communication requests from a component of the RMS that is running at the e-mail server[51]. The Task manager sends all these requests to the same Request Handler. This Request Handler combines the information from these communication requests and instructs one Interaction module to take care of the notification.

[51] The functionality of this component is as follows: if it detects a new e-mail message, it obtains all information that it needs to construct a communication request (e.g. the identity of the recipient and the identity of the initiator). It sends the communication request to the Task Manager. This component is not dealt with in more detail.

The Task manager registers itself (with a URN) in the WWICE system as a ‘RMS’. The Task manager takes care of all communication between the internal components of the ‘RMS’ and entities in a different WWICE system. For example, it takes care of the communication between a Request Handler that has a local deployment with respect to itself, and a Request Handler that is part of a RMS in a different WWICE system. The reason for this is that a RMS must have *one* communication address (a URN) that a RMS in another WWICE system can use to communicate with the RMS. This is the URN that the Task manager uses to register in the WWICE system.
When the RMS is started, this means that the Task manager is started, since the Task manager registers itself in the WWICE system as a ‘RMS’. Per WWICE system, there might be multiple RMS-es that are running continuously[52]. However, there is only one ‘main’ RMS per WWICE system that deals with any tasks. At start-up, the Task manager searches for other RMS-es that are running. If there is another RMS that is running, it contacts that RMS to settle which RMS is the ‘main’ RMS. If there is no RMS running, the Task manager determines that it is the ‘main’ RMS.

6.2 CLUSTER MANAGER

The RMS needs a Cluster manager to deal with the situation that multiple User Interface modules must create a UI at the same device. The Cluster manager combines the UI descriptions of different User Interface modules in order to create one WWICE-specific UI description for a cluster. Furthermore, the Cluster manager determines the specific properties of a cluster (e.g. the available UI capabilities, whether someone is using the cluster, and the activities that occur nearby the cluster).

The Cluster manager combines UI descriptions as follows. Suppose that four User Interface modules want to create a UI at a particular cluster. The Cluster manager combines the four UI descriptions in order to create one WWICE-specific UI description for the cluster. For example, it depicts all icons in the corner of a screen, and it depicts all pop-up menus in the center of the screen. It sends the WWICE-specific UI description to the appropriate UI Manager(s). Figure 36 illustrates the operation of the Cluster manager.

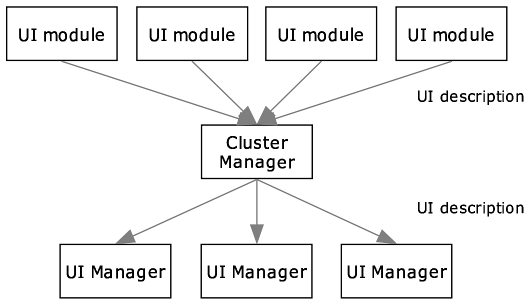


Figure 36 Cluster manager combines UI descriptions

The User Interface module could have dealt with the UI differently. The User Interface module could have instructed the UI Manager of the WWICE system to allocate a particular part of the UI resources that are allocated for the RMS, for its’ own UI. In this situation, the User Interface module would send its’ UI description immediately to UI Managers, instead of to a Cluster manager. However, the User Interface modules use Cluster managers since a Cluster manager can create an application-specific UI. For example, if there are several User Interface modules that want to create a pop-up menu, the Cluster manager might decrease the size of these pop-up menus, such that these windows do not use too much space in the GUI. A UI Manager cannot combine UI descriptions in this way.

[52] Section 6.5 explains how many RMS-es should run continuously.

A User Interface module starts the Cluster manager when that module needs to show a UI on a particular cluster. It first determines whether there is a Cluster manager running for the cluster that it wants to use for its' UI. If it cannot find a Cluster manager, it starts a new Cluster manager. The User Interface module creates one UI description per cluster, since every cluster might have different properties (e.g. every cluster might have different UI capabilities available). It sends the UI descriptions to the appropriate Cluster Managers. If several User Interface modules must create a UI in the same cluster, they use the same Cluster manager. The Cluster manager is running until there are no more User Interface modules that want to create a UI in the cluster that it manages.

6.3 CRITERIA FOR DEPLOYMENT

This section explains the criteria for deployment. These criteria are the criteria for the deployment of component B with respect to component A. Components A and component B can be any component of the RMS. A local deployment means that component A and component B are always installed at the same device. A distributed deployment means that component A and component B *can* be installed at different devices. Figure 37 illustrates the local and the distributed deployment of component B with respect to component A.

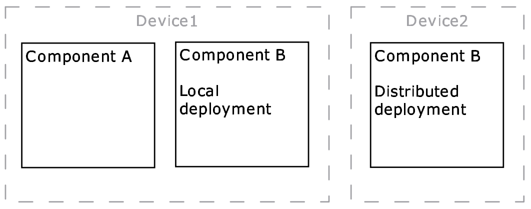


Figure 37 Example of local and distributed deployment

6.3.1 Types of failure

This section explains the types of failure that a RMS must deal with by means of a good deployment. Figure 38 shows these types of failures (indicated by a 'X'). It shows a component at every device (indicated by a square with a 'C'). This does not mean that there should be a component installed on every device. This only indicates that one could install components (or even a whole RMS), at every device in the network.

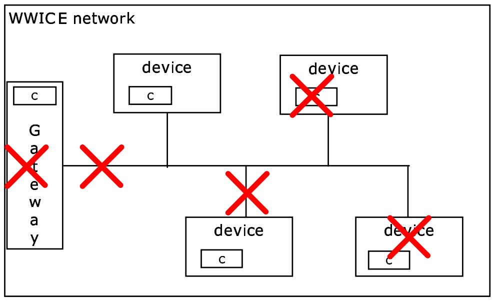


Figure 38 Type of failures in a network

Figure 38 shows that there are five types of failure:

- The device that acts as a gateway from the WWICE system to the out-side world, might fail. In this situation, a RMS that is installed at the Gateway also fails. A RMS that is installed at a device within the WWICE network, cannot handle any communication to or from the out-side world, and it cannot use any components that are installed at the Gateway.
- A network split between the Gateway and the rest of the network might occur. A RMS that is installed at the Gateway can still execute some tasks, e.g. it might reject all real-time communication. However, it cannot use any components that are installed at devices in the rest of the network. A RMS that is installed at a device within the network, cannot handle any communication to or from the out-side world, and it cannot use any components that are installed at the Gateway.
- A network split between a device and the rest of the network might occur. In this case, a RMS that is installed on that device cannot start or receive any communication. However, it might still be able to provide some functionality, e.g. a user could indicate preferences if preferences are stored locally. A RMS that is installed at a different device can still function properly, but it cannot use any components that are running at the device that is split-off the network.
- The RMS might fail. In this situation, a different RMS might still function properly. That RMS can still use the device where the other RMS was running, e.g. to display a UI.
- The device that runs the RMS might fail. In this situation, the RMS that is running on that device also fails. A RMS that is installed at a different device can still function properly. However, it cannot use any components that were running at the device that has failed.

6.3.2 Criteria for deployment

The following criteria are used to determine the deployment of component B with respect to component A. These criteria are based upon, among other things, the types of failure that are dealt with in section 6.3.1.

- If component B provides a service that component A really needs for its execution (e.g. it provides important information), this gives preference to a local deployment of component B such that no network split between component A and component B can happen.
- If component B uses a lot of CPU, memory, storage, or other resources, this gives preference to a distributed employment of that component. For example, because only one device in the network is powerful enough or has the necessary resources to execute that component, or because the component consists of multiple components that spread the load over several devices.
- If there is a high information flow between component A and component B, this gives preference to a local deployment. If there is a high information flow between component B and an external entity, this gives preference to a deployment at the device where the external entity is located. For example, if component B is able to encode an incoming mpeg-2 video stream into a mpeg-4 video stream, it might be good practice to install that component at the device where the mpeg-2 video stream originates. This way, the network is spared from transporting any mpeg-2 data.
- If a low communication delay is needed between component A and component B, this gives preference to a local deployment of component B.

6.4 DEPLOYMENT COMPONENTS OF RMS

This section explains the deployment of the components of the RMS that is shown in Figure 39. Appendix IV explains the reasons for the chosen deployment in detail. The components that are depicted at device 1 are the components that are part of the software application that is registered in the WWICE system as a 'RMS'[53]. In other words, these components have a local deployment with respect to the 'RMS'. The components that are depicted at device 2 are components that might run on a different device than the 'RMS'. In other words, these components have a distributed deployment with respect to the 'RMS'.

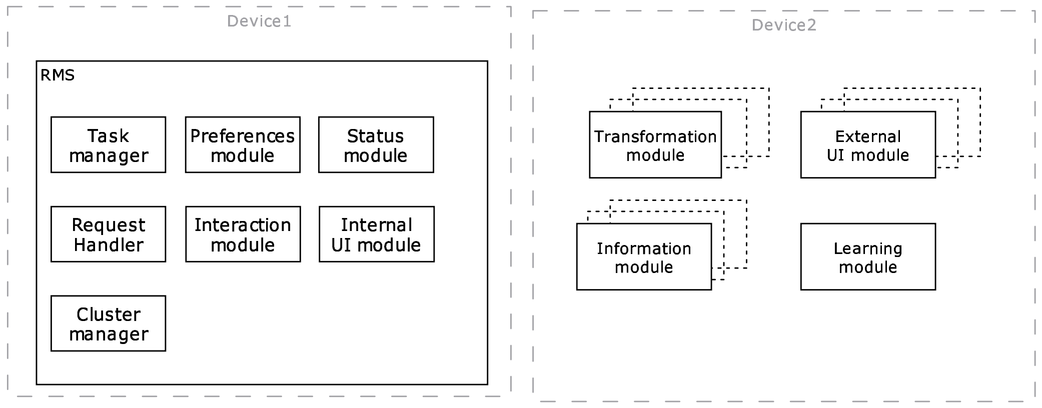


Figure 39 Deployment of the components of the RMS.

The more components are part of one software application (the 'RMS'), the easier it is for the user to install the RMS. This means that preferably, all components of the RMS are part of one application that a user can install at one device. The components that are depicted in Figure 39 as part of device 1, are all components that are part of the software application that is registered in the WWICE system as a 'RMS'. These components provide functionality that is not of interest to external entities (entities that are not a component of the RMS). Furthermore, these components do not need a lot of resources (e.g. CPU or storage). Therefore, it is possible to combine these components in one software application. However, this is not possible for the components that are depicted in Figure 39 as part of device 2. In a nutshell, this is for the following reasons:

- Some of these components provide functionality that is not RMS-specific, but more generic. In other words, other WWICE entities might need to use the functionality of these components. For example, a Transformation module that is able to transform mpeg-2 into mpeg-4 is typically a WWICE-service (a Transcoder-resource). Section 5.4 explains that the Graphmapper might use this service to enhance a graph.
- Some of these components need a lot of resources (e.g. CPU, memory) to execute their task. These components must have a distributed deployment with respect to the RMS, for example because only one device in the network is powerful enough or has the necessary resources to execute that component.
- Some of these components have a high information flow between itself and an external entity. This gives preference to a local deployment with respect to that external entity. For example, a User Interface module that is able to start video communication with a person out-side the WWICE system, needs a lot of bandwidth between itself and the Gateway.

The components that are depicted in Figure 39 as part of device 2, are all registered in the registry of the WWICE system. These components are WWICE services.

[53] Note that there are external and internal User Interface modules. The internal User Interface module uses the UI capabilities of the home system (WWICE) to create its' UI. The external User Interface modules use typically other resources, e.g. for interaction with a person via a telephone connection.

6.5 DEPLOYMENT OF RMS-ES IN WWICE

This section discusses the deployment of the RMS in WWICE. In other words, this section discusses how many RMS-es one should install per WWICE system, and where one should install these RMS-es. In this section, the ‘RMS’ is the software application that is registered in the WWICE system as a ‘RMS’. Since it takes effort to install many RMS-es, it has preference to install as few RMS-es as possible. There are two reasons to install multiple RMS-es. The first reason is to make sure that there is a RMS that is running continuously. The second reason is that RMS-es might cooperate to deal with the tasks that a RMS must execute, e.g. to deal with a communication request. Section 6.5.1 discusses how many RMS-es should run continuously. Section 6.5.2 discusses whether RMS-es should cooperate to deal with tasks. Section 6.5.3 summarizes the conclusions of this section.

6.5.1 Number of RMS-es that must run continuously

If an initiator indicates to her RMS that she is willing to start communication with a particular recipient, her RMS sends a communication request to the RMS of the recipient. In order to be able to send this request, the RMS of the initiator asks a URN resolver for a reference to the RMS of the recipient. This URN resolver must be able to find the RMS of the recipient. Since a URN resolver cannot start any applications, the RMS of the recipient must be already running. Therefore, there must be at least one RMS per WWICE system that is running continuously.
Section 6.3.1 explains the different types of failure that might occur in a network. From this section follows that if a user installs a RMS at the Gateway of the WWICE system, this results in the best availability of a RMS (it is assumed that the Gateway is a device that is always running). In this situation, the RMS can provide functionality to out-side parties, even if a network split between the Gateway and the rest of the WWICE system occurs (if it does not need to use components that are running at a different device). However, the RMS that is running at the Gateway may fail. Furthermore, a RMS must also take care of communication between persons who are located in the same WWICE system. A network split between the Gateway and the rest of the network would cause this functionality to be unavailable. This gives preferences to several RMS-es that are running continuously such that there is always a RMS that is running. A user can install as many RMS-es as she likes. The more RMS-es she installs, the better the chances that there is a RMS running continuously. However, I think that it is sufficient to have one or two RMS-es running continuously. If one RMS fails and there is another one running, this RMS can take over and possibly start a new RMS (if there are more RMS-es installed in the WWICE system).

6.5.2 Cooperation between RMS-es to deal with tasks

This section discusses whether RMS-es should cooperate to deal with the tasks that they must execute, e.g. to deal with a communication request. There are two types of cooperation. RMS-es could cooperate to deal with *one* task, and RMS-es could distribute tasks over several RMS-es. The following subsections deal with these two options.

§ ***Cooperation to deal with one task***
When a RMS deals with a task (e.g. it deals with a communication request), the only subtask that it could delegate to other RMS-es is the interaction with a person that is located within the WWICE system. The RMS could distribute this task over multiple RMS-es since it might need to create a UI on multiple devices. A user could install a RMS at multiple devices such that each RMS can take care of the UI of the RMS for a particular area, e.g. for one cluster of devices. This means that if a RMS determines that it is necessary to start interaction with a user within the WWICE system, it asks several RMS-es to take care of this interaction. Each RMS sends UI descriptions to the UI managers that are located in the area where it takes care of the UI. The other option is that one RMS sends UI descriptions to all (remote) UI managers that it needs for its’ UI. In this situation, the RMS has a centralized UI. A centralized UI has the following advantages:

- If only one RMS takes care of the UI in the whole WWICE system, the user needs to install only one RMS. Of course, she might install a few more RMS-es to make sure that there is always a RMS that is running continuously.
- The RMS does not need to start other RMS-es to take care of interaction. This saves time. Of course, the other RMS-es could run continuously. However, this costs CPU and memory. Section 6.5.1 explains that only a few RMS-es need to run continuously.
- If a new device enters the WWICE system, the RMS can automatically use that device for its’ UI. If the UI is distributed, the RMS-es need to settle which RMS takes care of the UI at the new device. This makes the operation of the RMS-es more complex.
- There is no need for synchronization between multiple RMS-es. For example, suppose that multiple RMS-se cooperate to deal with a communication request regarding real-time communication (they take care of the notification of the recipient). If one RMS determines that it should start interaction with the initiator, it must synchronize this task with the other RMS-es to make sure that only one RMS starts interaction with the initiator. Synchronization takes time and makes the operation of the RMS-es more complex.
- If only one RMS deals with the UI, this means that RMS-es do not cooperate to deal with *one* task. Therefore, only one RMS needs to retrieve the information that it needs to deal with this task (e.g. preferences). This saves some bandwidth.

Disadvantages of a centralized UI:

- One RMS must create all UI descriptions thus the computation of the UI descriptions is localized at one device. At this moment, it is not possible to determine whether this will cause any problems (e.g. a shortage of available CPU or memory). However, since the focus of this project is on the home environment, this will probably not cause any problems since the RMS does not need to create many UI descriptions. For example, if it uses 5 devices for it’ UI, and it must start 3 notifications at the same time, it needs to create 15 UI descriptions.
- The RMS must send UI descriptions over the network. The size of a UI description is approximately between 1 Kb and 20 Kb. However, there is probably enough bandwidth available. LAN networks provide a bandwidth of 10 Mbps to 100 Mbps, wireless LANs provide a bandwidth of 5 to 11 Mbps (and in the future 54 Mbps), and fiberglass networks provide a bandwidth of 10 Mbps to 10 Gbps. This means that the bandwidth that is necessary to send UI descriptions is negligible. Note that if multiple RMS-es deal with the UI, these RMS-es might also need to send UI descriptions over the network. However, since they only RMS deals with the UI for one particular area, these UI descriptions do not need to be transported over the whole network.

The main advantage of a distributed UI is that a RMS can divide the computation of UI descriptions over multiple devices. The main advantages of a centralized UI is that a user does not need to install many RMS-es, and that a RMS can easily deal with new devices that enter the WWICE system. Therefore, the centralized UI is the best option. This means that only one RMS deals with a task.

§ ***Cooperation to distribute tasks over several RMS-es***

RMS-es might either cooperate to distribute tasks over multiple RMS-es, or only one RMS might deal with all tasks. The option that one RMS deals with all tasks has the following advantages:

- If RMS-es cooperate to distribute tasks, a component of one RMS might need to use components that are part of another RMS. For example, only one Cluster manager must send UI descriptions to the UI Managers of a particular cluster. Therefore, either the User Interface module of one RMS sends its’ UI description to a Cluster manager that is part of another RMS, or two Cluster managers that manage the same cluster must cooperate. This means that the components of one RMS must know which components exist in the other RMS. This increases the complexity of the RMS and it might introduce synchronisation problems (e.g. what to do when two RMS-es create a Cluster manager at the same time). If one RMS deals with all tasks, a component of that RMS does not need to cooperate with components in other RMS-es.

- If a RMS must deal with a task, it retrieves information from other entities in the WWICE system. For example, it retrieves user preferences from Person services, and it asks an Application Manager whether it is possible to start a particular type of communication. One RMS can use this information for multiple tasks. If multiple RMS-es would execute tasks, they would need to retrieve the same information. This causes (WWICE-)entities to execute the same task several times.
- If Cluster managers in different RMS-es must cooperate, or a User Interface module must send its' UI description to a Cluster manager that is part of another RMS, this costs time since the UI descriptions are sent to the appropriate UI managers, via another RMS. This situation does not occur if one RMS deals with all tasks.

Disadvantages of one RMS that deals with all tasks:
- If one RMS executes all tasks, all computation is located at one device. At this moment, it is not possible to determine whether this will cause any problems (e.g. a shortage of available CPU or memory).

At this moment, it is not clear whether it will cause any problems if one RMS deals with all tasks. However, it is clear that if RMS-es cooperate to distribute tasks over multiple RMS-es, this increases the complexity of the operation of the RMS-es to a large extent. Therefore, I choose to have only one RMS that deals with all tasks. Tests with an implementation of the RMS must determine whether this will cause any problems, e.g. a shortage of available memory. The RMS-es that are running must settle among each other which RMS is the 'main' RMS that executes all tasks.

6.5.3 Conclusions

There must be at least one RMS that is running continuously. The RMS has a centralized UI. This means that RMS-es do not cooperate to deal with one task. Furthermore, there is only one RMS per WWICE system that deals with any tasks at one point in time. This means that the RMS-es that are running must settle among each other, which RMS is the 'main' RMS that deals with all tasks. Since one RMS can take care of all tasks, and only one RMS needs to run continuously, one does not need to install many RMS-es per WWICE system. I think that it is sufficient to install one or two RMS-es. However, if one wants to be sure that there is always a RMS running, one might install more RMS-es. Section 6.5.1 explains that the Gateway is a good location to have a RMS running (note that it is assumed that the Gateway is always running).

6.6 OPERATION OF RMS IN WWICE

This section explains how components of the RMS cooperate with each other, and with entities of the WWICE system (e.g. the Application Manager) to take care of the functionality of the RMS. This is the main functionality of the RMS:
- The RMS is able to create a UI to interact with a person. For example, such that a person can indicate that she wants to start communication with someone, such that she can indicate her status, and such that she can indicate preferences.
- A person can use the RMS to start non real-time communication. In this situation, the RMS starts the appropriate communication application, e.g. an email application.
- A person can use the RMS to start real-time communication. In this situation, the RMS sends a communication request to the RMS of the recipient. If the recipient does not have a RMS, the RMS of the initiator starts the appropriate communication application, e.g. it starts a telephone application.
- The RMS deals with communication requests. This means that it determines whether it should notify the recipient of incoming communication. If it determines that it should notify the recipient, it creates a UI for the notification. If it determines that it should not notify the recipient, it either starts monitoring, it starts interaction with the initiator, or it forwards the communication. If the recipient and the initiator agree to start communication, the RMS starts the appropriate communication application.

Section 6.6.1 explains how the RMS creates a UI. Section 6.6.2 explains how the user can indicate that she wants to start communication, and how the components of the RMS cooperate to construct a communication request. Section 6.6.3 explains how a RMS can send a communication request to a RMS that is located in a different WWICE system. Furthermore, this section explains how two Request handlers that are part of RMS-es that are located in different WWICE systems, can communicate with each other. Section 6.6.4 explains how a RMS deals with a communication request. Section 6.6.5 explains how a RMS starts real-time communication and section 6.6.6 explains how a RMS starts non real-time communication. Unfortunately, there was no time to work out the situation that the RMS must forward communication. Section 6.6.7 explains the functionality of the Activity Monitor. The WWICE system does not have an Activity Monitor. However, for this graduation project we assume that it does have such a subsystem. Therefore, section 6.6.7 indicates what functionality the Activity Monitor provides.

6.6.1 UI of the RMS

The UI of the RMS is similar to the UI of the Space Browser with respect to the allocation of resources for the UI. This means that when a device starts up, the corresponding UI Manager automatically allocates resources for the UI of the RMS at that device. Note that this does not mean that other applications cannot use these resources. For example, a GUI consists of multiple layers. When the UI Manager allocates some GUI-resources for the UI of the RMS, this means that it allocates some space in *one* (or several) layer(s) of the GUI. When a different application needs to use the same resource, this is possible since it can use a different layer than the RMS. This section deals with different scenarios that explain how the RMS takes care of the UI.

Scenario 1: create UI
Figure 40 shows how the different components of the RMS and the WWICE system collaborate to create a UI.

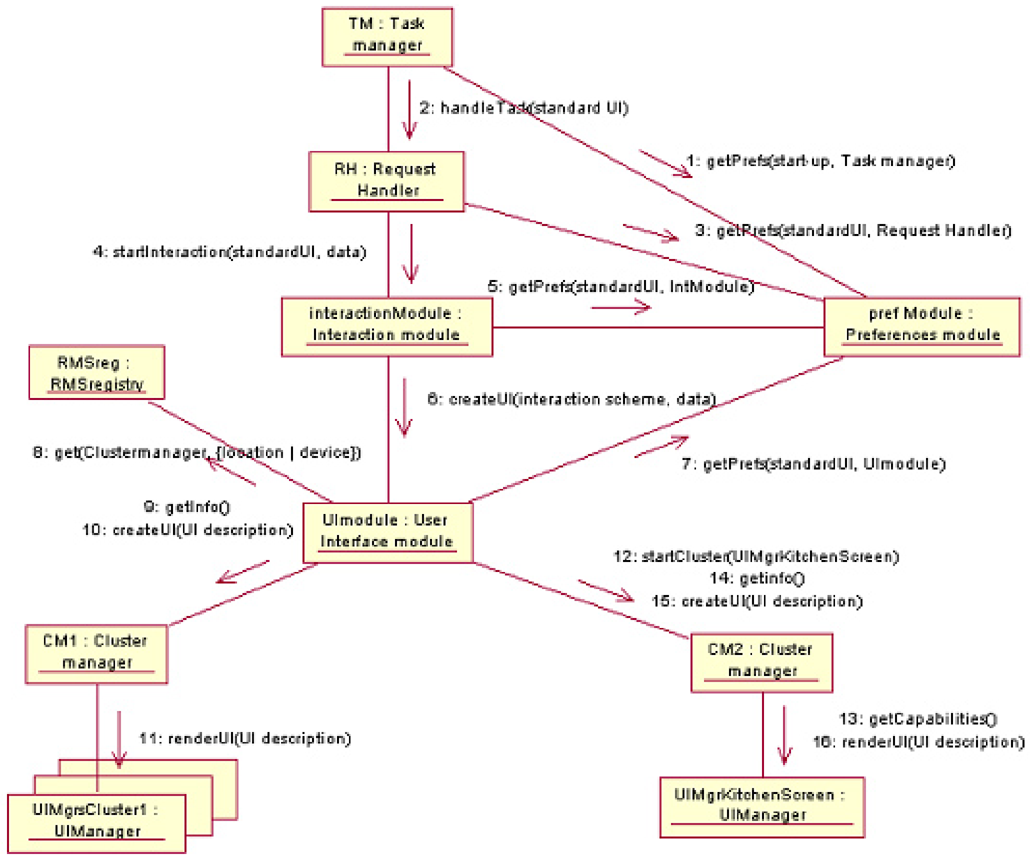

TM : Task manager
2: handleTask(standard UI)
1: getPrefs(start-up, Task manager)
RH : Request Handler
3: getPrefs(standardUI, Request Handler)
4: startInteraction(standardUI, data)
5: getPrefs(standardUI, IntModule)
interactionModule : Interaction module
pref Module : Preferences module
RMSreg : RMSregistry
6: createUI(interaction scheme, data)
8: get(Clustermanager, {location | device})
7: getPrefs(standardUI, UImodule)
9: getInfo()
10: createUI(UI description)
UImodule : User Interface module
12: startCluster(UIMgrKitchenScreen)
14: getInfo()
15: createUI(UI description)
CM1 : Cluster manager
CM2 : Cluster manager
11: renderUI(UI description)
13: getCapabilities()
16: renderUI(UI description)
UIMgrsCluster1 : UIManager
UIMgrKitchenScreen : UIManager

Figure 40 Create UI on multiple devices

This section explains how a RMS creates a UI, e.g. to notify a person of incoming communication, or to enable a person to start communication. In this scenario, a user of the RMS has indicated that the RMS must create a 'standard UI' on a particular screen in the kitchen and a particular screen in the living room. This standard UI enables persons to use to the basic functionality of the RMS, e.g. to start communication and to indicate preferences. For example, the RMS might show an 'address book'-icon that represents the RMS. If the user selects this icon, the RMS creates a pop-up menu that enables the user to start communication or to indicate preferences. Note that the RMS will usually show a standard UI such that people can start using the RMS. This means that the components that are depicted in Figure 40 are usually running continuously.

At start-up, the Task manager retrieves the preferences that indicate what tasks it should start (message 1). In this scenario, one of these tasks is to display a standard UI. The Task Manager starts a Request Handler to deal with this task (message 2). The Request Handler retrieves the preferences with respect to the standard UI from the Preferences module (message 3). In this scenario, these preferences indicate that the Request Handler must always show the standard UI, disregarding the status of the users of the RMS. Therefore, the Request Handler starts an Interaction module to take care of the interaction (message 4). The Interaction module retrieves any preferences with respect to the interaction from the Preferences module (message 5). An example of a preference is a preference that indicates whether a person is an 'administrator' or a 'user'. This preference influences the amount of options that a person can choose. For example, a 'user' can only choose the options 'start communication' and 'indicate preferences', while a 'administrator' can also choose the option 'manage users'. The Interaction module constructs the appropriate interaction scheme and starts a User Interface module (message 6).

The User Interface module retrieves the preferences that indicate how it should construct a standard UI, e.g. which device(s) it should use for the UI (message 7). In this scenario, it must use a screen in the kitchen and a screen in the living room. Next, it asks the internal registry of the RMS for Cluster managers that manage these devices (message 8). In this scenario, the Cluster manager 'CM1' manages a cluster that contains the screen in the living room. The User Interface module asks CM1 for any cluster specific information that it needs to create a UI description, e.g. the UI capabilities that are available at the cluster (message 9). Next, it creates a UI description and asks the Cluster Manager to create a UI (message 10). The Cluster manager combines this UI description with the UI descriptions that it receives from other User Interface modules (if any), and sends the WWICE-specific UI description to the appropriate UI Managers (message 11).

The User Interface module has determined that there is no Cluster manager for the cluster that contains the kitchen screen. Therefore, it starts a new Cluster manager (message 12). This Cluster manager determines which UI managers are in the cluster that contains the kitchen screen. In this scenario, the cluster contains only the kitchen screen. Next, it asks the UI Manager of the kitchen screen whether it has UI capabilities available for a RMS (message 13). Since the UI Manager allocates UI capabilities at start-up, a UI Manager will always have UI resources available if it determines that the device should be used for the UI of the RMS. If a UI Manager does not have any UI capabilities available, the User Interface module cannot use that device for the UI of the RMS. In this scenario, the UI Manager has the required UI capabilities available.

The User Interface module asks the Cluster manager for any cluster specific information that it needs to create a UI description for that cluster, e.g. the available UI capabilities and whether someone is using the cluster (message 14). Based upon this information, it creates a UI description and sends it to the Cluster manager (message 15). The Cluster manager creates a WWICE-specific UI description for the cluster, and sends that UI description to the appropriate UI Manager (message 16).

Scenario 2: Person starts to use cluster

When a User Interface module creates a UI in a particular cluster of devices, a person might start to use that cluster. This might influence the UI. The following scenario explains how User Interface modules deal with this situation. Suppose that a Cluster manager combines the UI descriptions of three User Interface modules. One User Interface module creates a UI for the notification of a telephone call for person C, one User Interface module creates a UI for the notification of new e-mail messages for person A, and one User Interface module creates a UI for the notification of new e-mail messages for person B. Person B logs in via the Space

Browser. Figure 41 shows how the different components of the RMS and the WWICE system collaborate to deal with this situation.

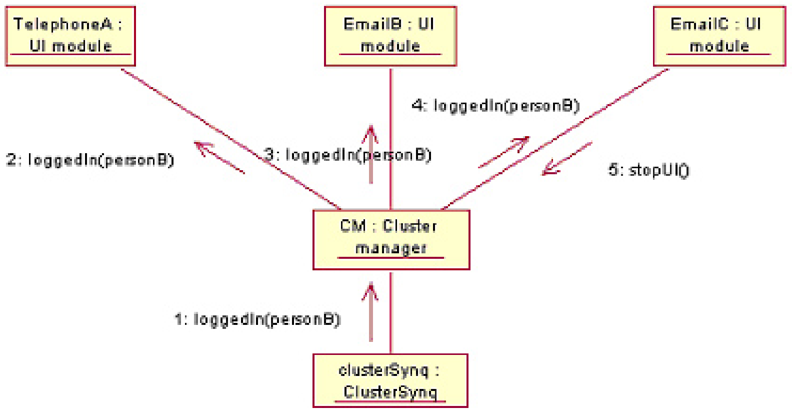


Figure 41 Person starts using device(s) in cluster

The Cluster manager has subscribed with the Clustersync to be informed when a user logs in. The Clustersync is a WWICE entity that keeps track of all status information of a particular cluster, e.g. which devices and UI Managers the cluster contains, and which persons are using the cluster. Therefore, the Clustersync notifies the Cluster manager that person B has logged in (message 1). The Cluster manager informs the User Interface modules of this event (message 2, 3, and 4). The User Interface modules know how to handle this situation. For example, the User Interface module that shows the notification of the telephone call knows that it should always create a UI for real-time communication (such as a telephone call), no matter who is using the cluster. The User Interface module that has creates the UI for the notification of new email for person B, knows that this information is only of interest to person B. Since person B logged in, it does not stop the UI. The User Interface module that shows the notification of new email for person C, knows that this information is not of interest to anyone but person C. Therefore, it stops the UI and waits until person B stops using the cluster. If person B stops using the cluster, the User Interface module sends its' UI description to the Cluster manager once again.

Scenario 2: Person starts to use the UI of the RMS

If a User Interface module creates a UI, a user might start to interact with that UI. Figure 42 shows how the different components of the RMS and the WWICE system collaborate to deal with this situation.

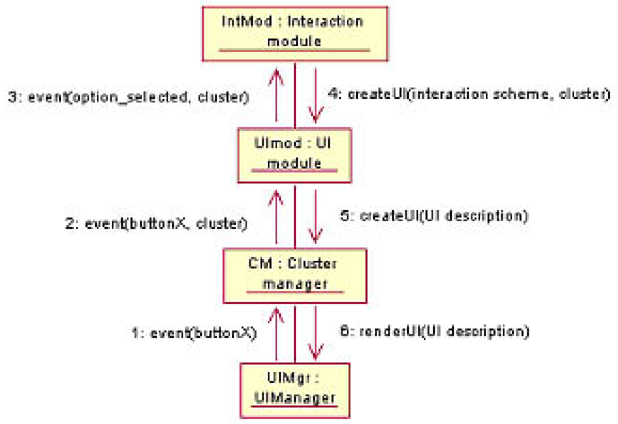

IntMod : Interaction module
3: event(option_selected, cluster)
4: createUI(interaction scheme, cluster)
UImod : UI module
2: event(buttonX, cluster)
5: createUI(UI description)
CM : Cluster manager
1: event(buttonX)
6: renderUI(UI description)
UIMgr : UIManager

Figure 42 Person starts using the UI of the RMS

A UI manager detects that the user has selected a particular button and sends this event to the Cluster manager (message 1). The Cluster manager forwards the event to the appropriate User Interface model (message 2). The User Interface module knows which option from the interaction scheme (that it received from the Interaction module) corresponds with the button-event. It informs the Interaction module (message 3). The Interaction module determines what to do. For example, if the user indicates that she want to start communication it forwards this request to the Request handler. However, in this scenario it determines that it must start to interact with the user. For example, if the UI shows a notification of new e-mail, the Interaction module must allow the user to start an application to read the e-mail message or to stop the notification. It constructs an interaction scheme and sends it to the User Interface module (message 4). The User Interface module creates a UI description and sends it to the appropriate Cluster manager (message 5). Note that the User Interface module might create a UI in multiple clusters. If a user starts to interact with a UI in a particular cluster, the User Interface module might change the UI that it has created in the other clusters. For example, if it has created a UI in multiple clusters for the notification of a new email message, it will stop the UI for the notification in these clusters, when a user starts to interact with the UI at a particular cluster.
The Cluster manager combines the UI description of the User Interface module with the UI descriptions that it receives from other User Interface modules (if any), and sends the WWICE-specific UI description to the appropriate UI Managers (message 6). Now, the user can select a new option.

Scenario 4: Cluster manager does not have enough UI capabilities available
When a User Interface module asks the Cluster manager whether it can create a UI on a particular cluster, the Cluster manager might indicate that this is not possible. The following scenario explains how the RMS deals with this situation. Suppose that there are two User Interface modules that create a UI on a particular cluster (UImodule1 and UImodule2). A third UI module also wants to create a UI on that cluster (UImodule3). However, the Cluster manager does not have enough UI capabilities available. Figure 43 shows how the different components of the RMS collaborate to deal with this situation.

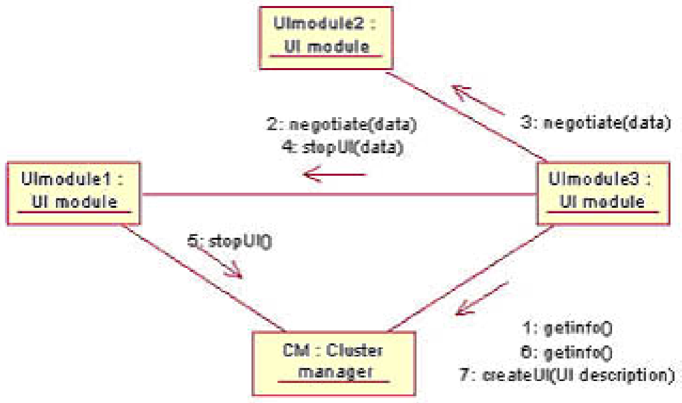


Figure 43 Cluster manager does not have enough UI capabilities available

UImodule3 asks the Cluster manager for the information that it needs to create a UI (message 1). The Cluster manager responds that it does not have enough UI capabilities available, but that there are two more User Interface modules that create a UI at that cluster. The Cluster manger does not know anything about the content of the UI. It just combines UI descriptions. Therefore, the User Interface module must deal with the situation that there are not enough UI capabilities available. UImodule2 starts to negotiate with the other User Interface modules (message 2 and 3). In this scenario, it determines that that there will be enough UI capabilities available if UImodule1 stops its' UI, and that its' own UI is more important than the UI that UImodule1 has created (e.g. because UImodule1 has created a UI for the notification of email communication and UImodule3 must create a UI for the notification of telephone communication). UImodule3 asks UImodule1 to stop its' UI (message 4). UImodule1 stops the

UI (message 5). Next, UImodule3 asks the Cluster manager once more for all information that it needs to create a UI (message 6). This time, there are enough UI capabilities available thus UImodele3 instructs the Cluster manager to create a UI (message 7).
If it is not possible to let another User Interface module stop its' UI, UImodule3 either waits until the required UI capabilities become available, or it searches for a different cluster that it can use for its' UI. The appropriate action depends on the user preferences that indicate what device(s) it must use for its' UI. If it cannot create a UI anywhere at all, it informs the Interaction module (if necessary).

Scenario 5: Cluster split
When a particular Cluster manager is managing a cluster, a cluster split might occur. This means that one (or more) device(s) leave the cluster. The device(s) that left the cluster join another cluster or start to constitute a new cluster. The Clustersync detects the cluster change (e.g. whether a device enters or leaves the cluster). The Clustersync is a WWICE entity that keeps track of all status information of a particular cluster, e.g. which devices and UI Managers the cluster contains, and which persons are using the cluster. The following scenario explains how the RMS deals with a cluster split. In this scenario, there are two User Interface modules that create a UI on a particular cluster. The Clustersync detects a cluster split. Figure 44 shows how the different components of the RMS collaborate to deal with this situation.

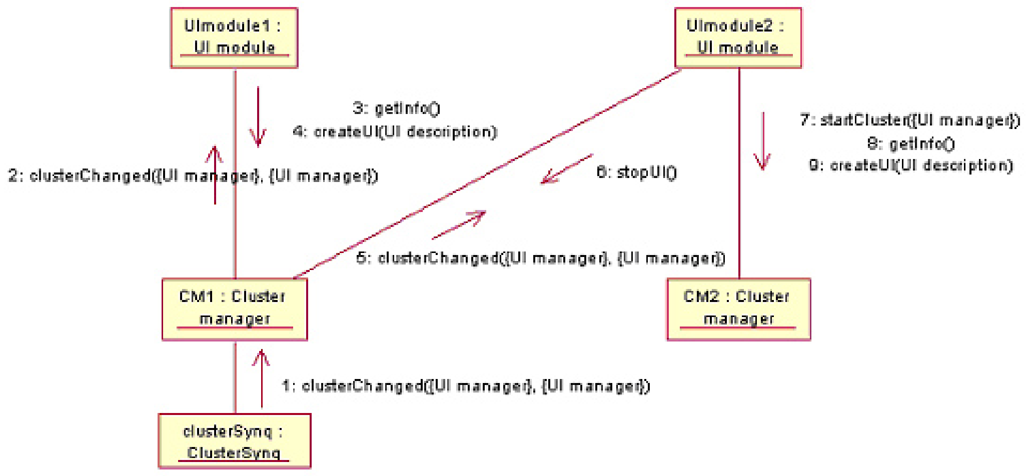

UImodule1 : UI module
UImodule2 : UI module
3: getInfo()
4: createUI(UI description)
7: startCluster([UI manager])
8: getInfo()
9: createUI(UI description)
2: clusterChanged([UI manager], [UI manager])
6: stopUI()
5: clusterChanged([UI manager], [UI manager])
CM1 : Cluster manager
CM2 : Cluster manager
1: clusterChanged([UI manager], [UI manager])
clusterSync : ClusterSync

Figure 44 Cluster split

The Clustersync informs the Cluster manager that the cluster is split in two clusters (message 1). It indicates how the UI managers are divided over the new clusters. The Cluster manager keeps on managing the cluster that it managed before. However, if that cluster does not contain any UI Managers anymore, it stops its' execution. For example, if the cluster contained only one device, and that device joined another cluster, this means that the UI Manager of that device is now part of that other cluster. In this situation, the cluster of that device does not contain any UI managers anymore thus the Cluster manager can stop its' execution.
The Cluster manager informs the User Interface modules which UI Managers are in which cluster (message 2 + 5). The User Interface modules determine whether they must still create a UI in the cluster that the Cluster manager manages. For example, if the cluster does not contain any UI Managers anymore, the User Interface modules will try to create a UI in the other cluster.
In this scenario, UImodule1 determines that it should still create a UI in the cluster that CM1 manages. It asks for the information that it needs to create a UI on that cluster, e.g. the available UI capabilities (message 3), and it instructs the Cluster manager to create a UI (message 4). UIModule2 determines that it should create its' UI in the new cluster, thus it instructs CM1 to stop its' UI (message 6). UIModule2 determines whether there is a Cluster manager for the other cluster. In this scenario there is no Cluster manager for that cluster, thus it starts a new one (message 7). It asks the new Cluster manager for the information that

it needs to create a UI on that cluster (message 8), and it instructs the Cluster manager to create a UI (message 9).

The situation that two clusters merge is similar to the situation of a cluster split. One cluster always joins the other cluster. If there was a Cluster manager for both former clusters, this means that one Cluster manager (CM1) now manages a cluster with all UI Managers, and the other Cluster manager (CM2) now manages a cluster with no UI Managers. The Cluster managers inform the User Interface modules that create a UI in their cluster, of the new UI capabilities. This means that the User Interface modules that created a UI using CM2, will start to create a UI using CM1. CM2 stops its' execution.

6.6.2 Start communication request

This section explains how a user can indicate that she wants to start communication with somebody. This means that the user is the initiator of the communication. She can use the UI of the RMS to indicate that she wants to start communication or she can use the Space Browser (the 'desktop application' of the WWICE system). The following scenarios explain these two options in detail.

Scenario 1: Person starts communication via UI of RMS
A person can indicate via the standard UI of the RMS, that she wants to start communication with somebody. For example, the standard UI displays an 'address book' icon that the user can select. If the user selects this icon, the RMS starts to display a pop-up menu that contains, among other things, the option to start communication. The first scenario of section 6.6.1 explains how the RMS creates a standard UI. Figure 45 shows how the components of the RMS and the WWICE system collaborate to deal with the situation that a person selects the option to start communication.

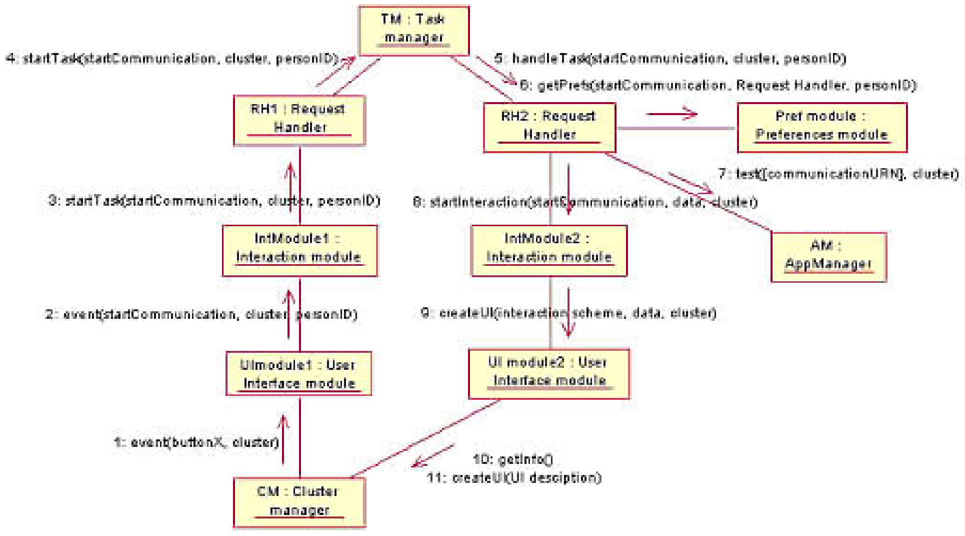

TM : Task manager
4: startTask(startCommunication, cluster, personID)
5: handleTask(startCommunication, cluster, personID)
6: getPrefs(startCommunication, Request Handler, personID)
RH1 : Request Handler
RH2 : Request Handler
Pref module : Preferences module
7: test([communicationURN], cluster)
3: startTask(startCommunication, cluster, personID)
8: startInteraction(startCommunication, data, cluster)
IntModule1 : Interaction module
IntModule2 : Interaction module
AM : AppManager
2: event(startCommunication, cluster, personID)
9: createUI(interaction scheme, data, cluster)
UImodule1 : User Interface module
UI module2 : User Interface module
1: event(buttonX, cluster)
10: getInfo()
11: createUI(UI desciption)
CM : Cluster manager

Figure 45 Start communication via UI of RMS

A UI Manager informs the Cluster manager that a user has selected a particular button. The Cluster manager forwards the event to the appropriate User Interface module (message 1). The User Interface module knows which option from the interaction scheme (that it received from the Interaction module) corresponds with the button-event. Furthermore, it knows which user is using the cluster (since the Cluster manager provides this information). In this scenario, the user has selected the option 'Start communication' that indicates that the user wants to communicate with someone. The User Interface module informs the Interaction module of the option that the user selected, of the cluster where the user started interaction with the UI of the RMS, and of the identity of the user (message 2). The Interaction module

determines what to do. In this situation, it determines that the Request Handler must deal with the task that the user has selected, thus it instructs the Request Handler to start that task (message 3). The Interaction module provides the Request Handler with all information that the Request Handler might need to deal with the task (the type of task, the cluster, and the identity of the person that instructs the RMS to start the task). The Request Handler determines that it should not deal with this task itself (since its' task is to create a standard UI). The Task manager is the component of the RMS that distributes tasks over Request Handlers. Therefore, the Request Handler forwards the task to the Task manager (message 4). The Task manager starts a new Request Handler to handle the task (message 5). It provides all information that the Request Handler can use to deal with its' task (e.g. the identity of the person who wants to start the communication and the cluster where the User Interface module should start the UI).
The new Request Handler asks the Preferences module for the preferences that it needs to deal with this task (message 6). Next, it asks the Application Manager to test which types of communication it can start at the cluster that the user is using (message 7). Meanwhile, the Request Handler starts an Interaction module to start interaction with the user, e.g. to find out what type of communication the user is willing to start and with whom she wants to start communication (message 8). The Interaction module retrieves the preferences that it needs to create an interaction scheme from the Preferences module (this is not depicted in Figure 45). Next, it starts a User Interface module (message 9).
The User Interface module retrieves the preferences that it needs to create a UI (this is not depicted in Figure 45) and asks the appropriate Cluster manager for all information that it can use to create a UI description, e.g. the UI capabilities of the cluster and the identity of the person that is using the cluster (message 10). Next, it creates the UI description and sends it to the Cluster Manager (message 11). The Cluster manager combines this UI description with the UI descriptions that it receives from other User Interface modules (if any), and sends the WWICE-specific UI description to the appropriate UI Managers. Now, the user can start communication.

Scenario 2: Person starts communication via Space Browser
A person can indicate via the Space Browser of the WWICE system that she wants to start communication. This scenario explains how the RMS deals with this situation. Figure 46 shows how the components of the RMS and the WWICE system collaborate.

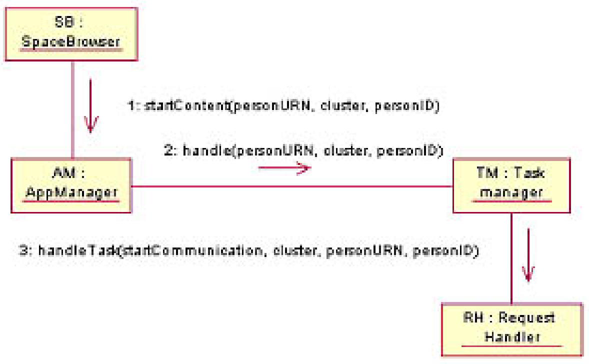

SB : SpaceBrowser
1: startContent(personURN, cluster, personID)
2: handle(personURN, cluster, personID)
AM : AppManager
TM : Task manager
3: handleTask(startCommunication, cluster, personURN, personID)
RH : Request Handler

Figure 46 Start communication via UI of Space Browser

When a person indicates via the Space Browser that she wants to start communication, this means that she selects a person icon that stands for the person with whom she wants to communicate. This person icon has a URN attached. The Space Browser asks the Application Manager to start, given this URN, the corresponding application at the correct device (message 1). The Application Manager determines what application should deal with the URN. There might be different applications that can deal with this URN. For example, there might be a 'Profile manager' that enables a user to manage some WWICE specific preferences. The Application Manager might show some options to the user via the Space Browser. In this scenario, the Application Manager determines that the RMS is the appropriate application that must deal with the URN. The RMS is running continuously, thus the Application Manager does not need to start the RMS. It sends the URN (with some other information that the RMS can use) to the RMS (message 2). The Task manager is the component of the RMS that is registered in the WWICE registry as a ‘RMS’, thus the Task manager receives the message from the Application Manager. The Task manager starts a Request Handler to handle this task (message 3). The first scenario of this section explains how the Request Handler deals with this task.

6.6.3 Interaction between remote RMS-es

This section explains how a Request Handler can send a communication request to a RMS that is located in a different WWICE system, and how components of remote RMS-es can communicate. Figure 47 shows how the components of the RMS-es collaborate.

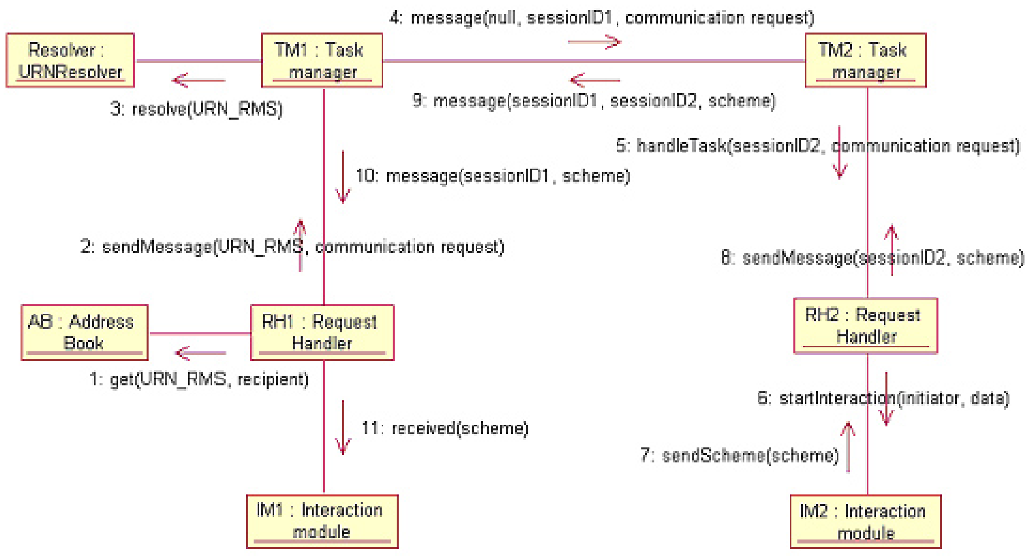

4: message(null, sessionID1, communication request)
Resolver : URNResolver
TM1 : Task manager
TM2 : Task manager
9: message(sessionID1, sessionID2, scheme)
3: resolve(URN_RMS)
5: handleTask(sessionID2, communication request)
10: message(sessionID1, scheme)
2: sendMessage(URN_RMS, communication request)
8: sendMessage(sessionID2, scheme)
AB : Address Book
RH1 : Request Handler
RH2 : Request Handler
1: get(URN_RMS, recipient)
6: startInteraction(initiator, data)
11: received(scheme)
7: sendScheme(scheme)
IM1 : Interaction module
IM2 : Interaction module

Figure 47 Start interaction with remote RMS

If an initiator is willing to start real-time communication with a person in a different WWICE system, the Request Handler of the RMS of the initiator (RH1) determines that it must send a communication request to the RMS of the recipient. It creates a communication request and retrieves the URN of the RMS of the recipient from the Address Book database (message 1). Next, it asks the Task manager of its’ RMS to send the communication request to the RMS of the recipient (message 2). Section 6.1 explains that the Task manager takes care of communication with other RMS-es. The Task manager creates a unique session identification number (sessionID1) that identifies the communication session between RH1 and the RH2 at the initiator’s side. Next, the Task manager asks a URN resolver to resolver the URN that stands for the RMS of the recipient (message 3). The Task manager receives a reference to the RMS of the recipient from the URN resolver, and sends the communication request (message 4). Note that the message contains the value ‘null’. This means that the Task manager does not (yet) have a sessionID that identifies the communication session at the recipient’s side.

When the Task Manager of the RMS of the recipient receives the communication request, it starts a Request Handler (RH2) to deal with the communication request (message 5). Furthermore, it creates a unique sessionID that identifies the communication session between RH1 and the RH2, at the recipient’s side (sessionID2). Now, RH1 can use sessionID1 to communicate with RH2 and RH2 can use sessionID2 to communicate with RH1.
RH2 deals with the communication request. In this scenario, it determines that it must start interaction with the initiator. It starts an Interaction module and instructs it to start interaction with the initiator (message 6). This Interaction module creates an interaction scheme and asks RH2 to send the scheme to the RMS of the initiator (message 7). RH2 asks TM2 to send

the message and indicates the sessionID (message 8). TM2 forwards the interaction scheme to TM1 and indicates that the message is part of the communication session that is identified at the initiator's side with 'sessionID1' (message 9). Furthermore, it indicates that its' own identification for the session is 'sessionID2'. This means that TM1 can use sessionID2 to respond. TM1 forwards the interaction scheme to RH1 (message 10). It indicates to RH1 that it can use 'sessionID1' to respond (since sessionID1 is the identification of the communication session at the initiator's side).

Now the communication session is set-up, and RH1 and RH2 can communicate via the local sessionIDs. The Interaction modules that are started by RH1 and RH2 communicate via these Request Handlers. Figure 48 shows how an Interaction module of one RMS can send an interaction scheme to the other RMS.

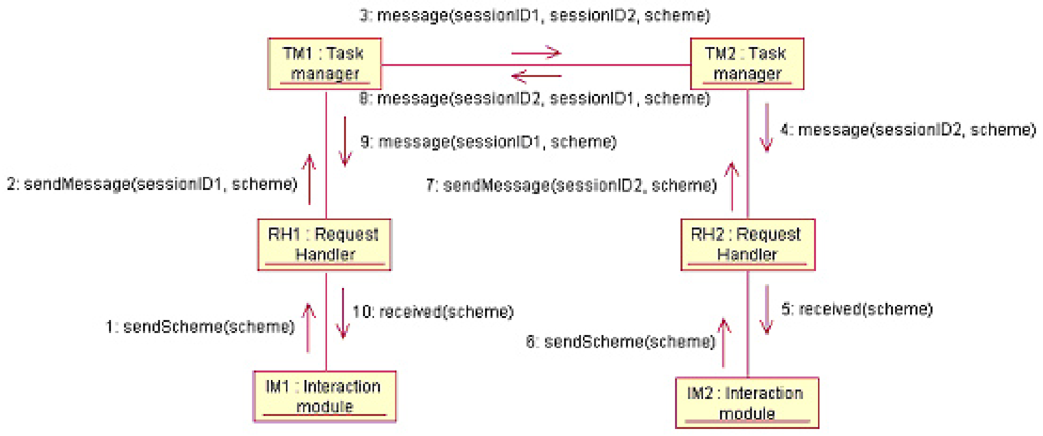

3: message(sessionID1, sessionID2, scheme)
TM1 : Task manager
TM2 : Task manager
8: message(sessionID2, sessionID1, scheme)
4: message(sessionID2, scheme)
9: message(sessionID1, scheme)
2: sendMessage(sessionID1, scheme)
7: sendMessage(sessionID2, scheme)
RH1 : Request Handler
RH2 : Request Handler
1: sendScheme(scheme)
10: received(scheme)
5: received(scheme)
6: sendScheme(scheme)
IM1 : Interaction module
IM2 : Interaction module

Figure 48 Interaction between remote Request Handlers

If the Interaction module of one RMS (IM1) must send an interaction scheme to the other RMS, it asks RH1 to send the interaction scheme (message 1). RH1 asks its' Task manager to send the scheme to the other RMS, and indicates the local sessionID (message 2). TM1 sends the message to TM2 and indicates that the message is destined for sessionID2, and that TM2 can use sessionID1 to respond (message 3). TM2 forwards the message to RH2 and indicates that it can use sessionID2 to respond (message 4). RH2 forwards the interaction scheme to IM2 (message 5). Figure 48 illustrates that IM2 replies with an interaction scheme in the same way (message 6 to 10).

6.6.4 Dealing with a communication request

This section explains how a Request Handler deals with a communication request. Note that if the communication request regards non real-time communication, the communication request does not originate from the RMS of the initiator. For example, suppose that an initiator sends an e-mail message. That e-mail message arrives at an e-mail server where a RMS-component is running. This component detects the arrival of new e-mail messages and constructs for every new message a communication request. It sends the communication request to the RMS. If the communication request regards real-time communication, the communication request originates from another Request Handler. This Request Handler is either part of a RMS that is located at a different WWICE system, or it is part of the same RMS. In the last situation, this means that a person wants to start communication with a person in the same home. The Task manager deals with all communication requests in the same way.

Figure 49 shows how the components of the RMS and the WWICE system collaborate to deal with a communication request.

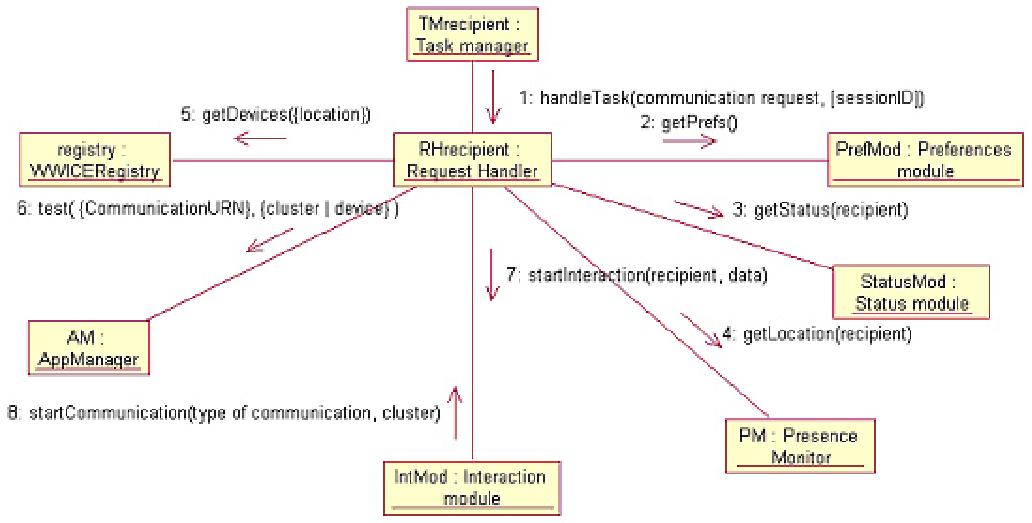

TMrecipient : Task manager
1: handleTask(communication request, [sessionID])
2: getPrefs()
5: getDevices([location])
registry : WWICERegistry
RHrecipient : Request Handler
PrefMod : Preferences module
6: test({CommunicationURN}, {cluster | device})
3: getStatus(recipient)
7: startInteraction(recipient, data)
StatusMod : Status module
AM : AppManager
4: getLocation(recipient)
8: startCommunication(type of communication, cluster)
PM : Presence Monitor
IntMod : Interaction module

Figure 49 Deal with communication request

The communication request arrives at the Task manager. If there is no Request Handler to deal with the communication request (e.g. the communication request regards real-time communication), the Task manager starts a new Request Handler. The Task manager sends the communication request to the Request Handler (message 1). If the communication request regards real-time communication, the Task manager provides a sessionID that the Request Handler can use for communication with the Request Handler in the RMS of the initiator. Otherwise, there is no communication between the RMS of the initiator and the RMS of the recipient (and thus no need for a sessionID).
The Request Handler first asks the Preferences module for the preferences that it needs to deal with the communication request (message 2). Since it is a time-consuming operation to test whether it is possible to start a particular type of communication, the Request Handler tries to minimize the types of communication that it must test. If the communication request regards real-time communication, the Request Handler of the initiator has provided the types of communication that it can start in the WWICE system of the initiator. Furthermore, this Request Handler might indicate which types of communication the initiator does *not* want to start. Based upon this information, the Request Handler of the RMS of the recipient determines which types of communication the recipient is willing to start. In order to determine whether the recipient is willing to start a particular type of communication, it needs the recipient’s status. It obtains this status from the Status module (message 3).
Next, it determines where the recipient is willing to start communication. The recipient might have indicated that she always wants to start communication with a device that is close her. Therefore, the Request Handler retrieves the current location of the recipient from the Presence Monitor (message 4). Furthermore, the recipient might have indicated that she likes to start particular types of communication at particular locations. Therefore, the Request Handler asks the registry of the WWICE system for the devices that are present at these locations (message 5).
Now the Request Handler knows which types of communication the recipient might be willing to start, and where she might want to start these types of communication. It asks the Application Manager to test whether it is possible to start these types of communication at these locations (message 6). If it is possible to start the communication that the recipient is willing to start, the Request Handler asks the Interaction module to notify the recipient. If the Request Handler determines that it should not notify the recipient, it starts another action. For example, it instructs the Interaction module to start interaction with the initiator. In this scenario, the Request Handler determines that it should notify the recipient, thus it starts an Interaction module to take care of this (message 7).

The Interaction module notifies the recipient and, if necessary, it starts interaction with the initiator. In this scenario, the result of the interaction is that the initiator and the recipient agree to start a particular type of communication. The Interaction module instructs the Request Handler to start this type of communication at the cluster where the recipient had indicated that she wants to start communication (message 8).

6.6.5 Start real-time communication

This section explains how RMS-es cooperate to start real-time communication. As an example, this section explains how the RMS-es cooperate to start video communication. The operation of the RMS-es is the same for other real-time communication such as telephone communication. Suppose that an initiator proposes to start video communication and the recipient accepts. The Request Handlers of the RMS-es must execute the following tasks:

1. Determine which RMS initiates the communication.
2. Inform AVComs.
3. Start communication.
4. Monitor communication set-up.

Determine which RMS initiates the communication
If both Request Handlers would ask an Application Manager to initiate communication, these Application Managers would set-up *two* video connections. Therefore, the Request Handlers must settle which Request Handler will initiate the communication. Since the initiator is the one who wants to start communication, the RMS of the initiator initiates the communication. The reason for this decision is that I assume that the initiator of the communication must pay for the communication. Obviously, it must be easy to change the operation of the RMS with respect to this point, since at this moment it is not clear how telephone operators (or internet providers) are going to bill people.

Inform AVCom
An Application Manager needs AVCom resources to set-up video communication. A Space Browser informs an AVCom if an AVCom can provide a particular type of communication to a particular person. Figure 50 summarizes how the WWICE system deals with real-time communication.

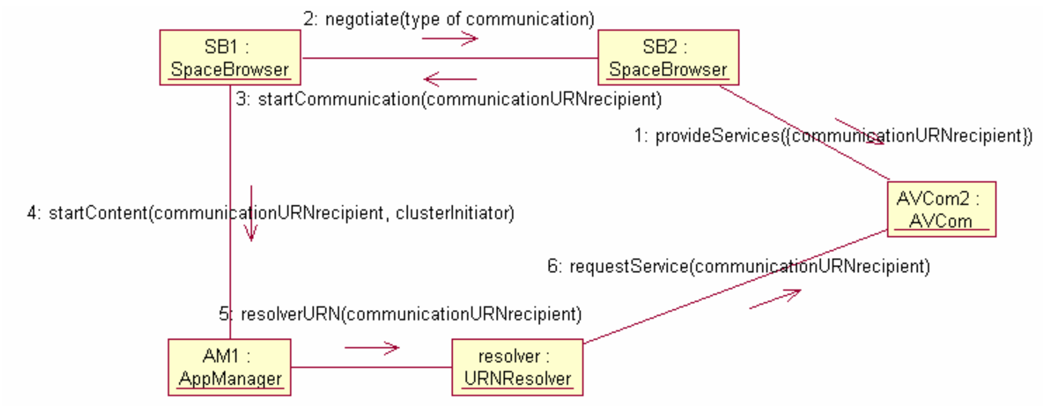

SB1 : SpaceBrowser
2: negotiate(type of communication)
SB2 : SpaceBrowser
3: startCommunication(communicationURNrecipient)
1: provideServices({communicationURNrecipient})
AVCom2 : AVCom
4: startContent(communicationURNrecipient, clusterInitiator)
6: requestService(communicationURNrecipient)
5: resolverURN(communicationURNrecipient)
AM1 : AppManager
resolver : URNResolver

Figure 50 Resolving a communication URN in WWICE

When a user logs in, the Space Browser informs the AVComs that are present in the cluster where the user has logged in, that they can provide communication to that user (message 1). More specifically, it informs the AVComs for which communication URNs they can provide communication. It retrieves these communication URNs from the person object of the user. If an initiator wants to start (video) communication with that user (this means that this user is the recipient), the Space Browser of the initiator (SB1) starts to negotiate with the Space Browser of the recipient (SB2) to determine whether the recipient is willing to start video communication (message 2). At this moment, this Space Browser of the recipient always indicates that the user is willing to start video communication. It provides the video communication URN of the recipient (message 3). The Space Browser of the initiator asks the Application Manager in the WWICE system of the initiator to start, given the video communication URN of the recipient, the corresponding application at the cluster that the initiator is using (message 4). In order to do this, the Application Manager needs the AVCom that can provide video communication with the recipient. It asks a URN resolver to resolver the video communication URN of the recipient (message 5). The URN resolver asks (via the URN resolver that is present in the WWICE system of the recipient) all resources that can provide communication in the WWICE system of the recipient, whether they can provide communica-tion services for the video communication URN of the recipient. The AVCom that can provide communication services for this URN knows that it can do this, since the Space Browser has informed it of this information. Therefore, the URN resolver can find this AVCom. Now, the Application Manager can start the video communication.

The Request Handler must take into account that the appropriate AVCom must know that it can provide communication to a person. If a person wants to start video communication on the device that she is using, the set-up of the video communication can proceed as depicted in Figure 51. However, if a person wants to start communication at a different device, this means that the Request Handler must inform the appropriate AVCom that it must provide video communication for that person. Furthermore, it must inform the AVCom of the device that the person is using at that moment, that it should not provide video communication. Now the set-up of the video communication can proceed as depicted in Figure 51.

Start communication
Figure 51 shows how the components of the RMS-es and the WWICE systems collaborate to start real-time communication. The only difference between the way WWICE starts real-time communication now, and the way RMS-es start real-time communication, is that the in the WWICE system the Space Browsers start the communication. The Space Browsers are now replaced by Request Handlers. If the Request Handler of the initiator initiates the communication, the Request Handler of the recipient must provide the communication URN of the recipient (as depicted by message 1 in Figure 51).

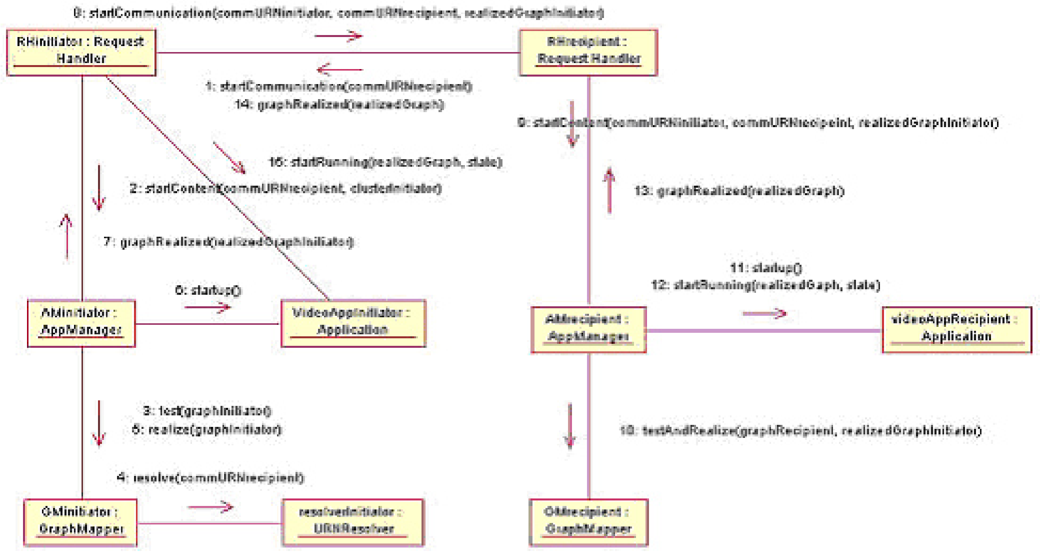

8: startCommunication(commURNinitiator, commURNrecipient, realizedGraphInitiator)
RHinitiator : Request Handler
RHrecipient : Request Handler
1: startCommunication(commURNrecipient)
14: graphRealized(realizedGraph)
9: startContent(commURNinitiator, commURNrecipeint, realizedGraphInitiator)
15: startRunning(realizedGraph, state)
2: startContent(commURNrecipient, clusterInitiator)
13: graphRealized(realizedGraph)
7: graphRealized(realizedGraphInitiator)
11: startup()
12: startRunning(realizedGaph, state)
6: startup()
AMinitiator : AppManager
VideoAppInitiator : Application
AMrecipient : AppManager
videoAppRecipient : Application
3: test(graphInitiator)
5: realize(graphInitiator)
10: testAndRealize(graphRecipient, realizedGraphInitiator)
4: resolve(commURNrecipient)
GMinitiator : GraphMapper
resolverInitiator : URNResolver
GMrecipient : GraphMapper

Figure 51 Starting video communication

Monitor communication set-up
The Request Handlers monitor the communication set-up. The communication set-up will probably succeed since the Request Handlers have already asked the Application Managers to test whether it is possible to start communication. However, other applications might have allocated the resources that are necessary to start video communication. If the communication set-up does not succeed, the Request Handlers instruct their Interaction modules to inform the recipient, and to propose other options to the recipient.

6.6.6 Start non real-time communication

In order to explain how a RMS starts non real-time communication, this section deals with two examples of non real-time communication: e-mail communication and video-mail.

Scenario 1: person starts e-mail communication
When an initiator decides to start e-mail communication, the Request Handler of the RMS of the initiator must have the URN that stands for e-mail address of the recipient. If the Request Handler does not have this URN, it instructs an Interaction module to start interaction with the initiator (to obtain the URN). The Request Handler might also try to retrieve the URN from the RMS of the recipient (if it has the URN that stands for the RMS of the recipient). In this situation, people only need to exchange the addresses of their RMS-es in order to be able to start communication. However, this option is not worked out in detail.
If the Request Handler does have the URN that stands for e-mail address of the recipient, it asks an Application Manager to start, given the URN, the appropriate application at the cluster where the initiator is located. The Request Handler monitors whether the application starts successfully. If the application starts successfully, its' task is fulfilled thus it stops its' execution. Otherwise, it instructs the Interaction module to propose a different type of communication to the initiator.

Scenario 2: person starts video mail
An initiator might decide to start video mail while she is negotiating with a recipient to start video communication. If there is a Transformation module at the WWICE system of the initiator that is able to record video-mail, the Request Handler of the RMS of the initiator will use that Transformation module (this way, there is no need to set-up a video connection between the WWICE system of the recipient and the WWICE system of the initiator). Otherwise, there must be a Transformation module at the WWICE system of the recipient that is able to record video-mail. In the first situation, the Transformation module records the video mail, and sends it to the RMS of the recipient. This section explains the last scenario in some more detail. Figure 52 illustrates how the components of the RMS-es collaborate.

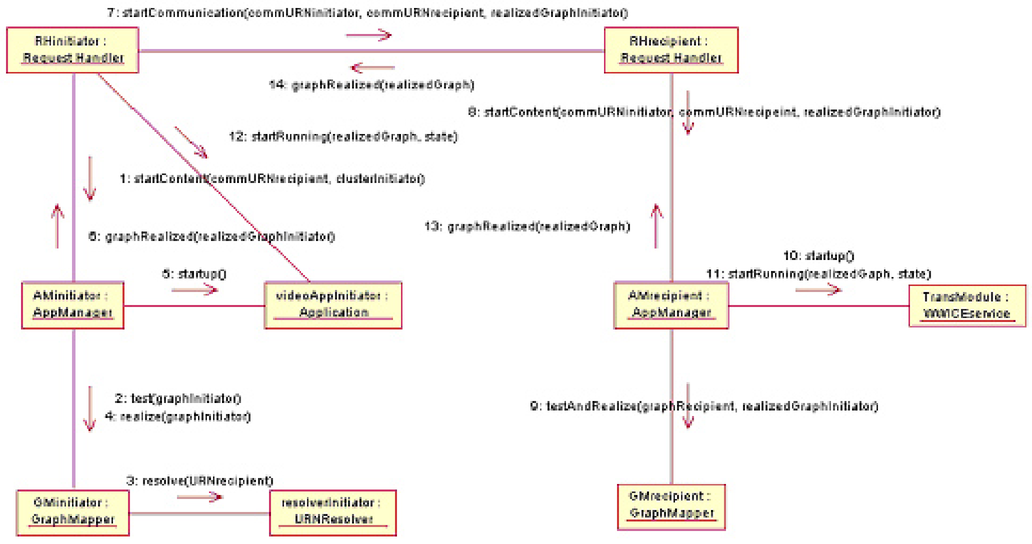

7: startCommunication(commURNinitiator, commURNrecipient, realizedGraphInitiator)
RHinitiator : Request Handler
RHrecipient : Request Handler
14: graphRealized(realizedGraph)
8: startContent(commURNinitiator, commURNrecipeint, realizedGraphInitiator)
12: startRunning(realizedGraph, state)
1: startContent(commURNrecipient, clusterInitiator)
13: graphRealized(realizedGraph)
6: graphRealized(realizedGraphInitiator)
10: startup()
11: startRunning(realizedGaph, state)
5: startup()
AMinitiator : AppManager
videoAppInitiator : Application
AMrecipient : AppManager
TransModule : WWICEservice
2: test(graphInitiator)
4: realize(graphInitiator)
9: testAndRealize(graphRecipient, realizedGraphInitiator)
3: resolve(URNrecipient)
GMinitiator : GraphMapper
resolverInitiator : URNResolver
GMrecipient : GraphMapper

Figure 52 Start recording a video mail

Note that Figure 52 is mainly the same as Figure 51. The only difference is that the Video communication application of the recipient is replaced with a Transformation module that is able to record video mail. If the Application Manager finds both a Video communication application as a Transformation module in the cluster that it should use for the set-up of the communication, it asks the Request Handler which one it should start. In this situation, the Request Handler indicates that the Application Manager must use the Transformation module for the communication. The Transformation records the video mail and stores it at a specific location (e.g. a local storage). If the video mail message is recorded successfully, the Request Handler of the RMS of the recipient constructs a communication request that indicates that a new video-mail message has arrived. If something goes wrong, the Request Handler of the RMS of the initiator instructs its' Interaction module to start interaction with the initiator (e.g. to propose to send an email message).

6.6.7 Activity Monitor

The WWICE system does not have an Activity Monitor. However, for this graduation project we assume that the WWICE system does have an Activity Monitor. The Activity Monitor keeps track of the current activities of the users of the WWICE system. The Status module and the User Interface module are the components of the RMS that might be interested in the information that the Activity Monitor provides. For example, the Status module might change the status of a person to 'Not available' when the Activity Monitor detects that the person is involved in the activity 'Having a shower'. The Activity Monitor is not a component of the RMS since several entities in the WWICE system might be interested in the information that the Activity Monitor provides. For example, the WWICE system contains a Ritual Player. A personal ritual is a series of activities that a person often performs. The Ritual Player 'automates' this series of activities, which means that when the system detects the start of a personal ritual, it automatically performs the actions that it anticipates. An example of a ritual is the 'wake-up'-ritual: when the morning alarm starts (this event indicates the start of the ritual), the light slowly turns on, a music starts to play and as soon as you enter the kitchen the morning news is displayed on the kitchen screen. The Ritual player is an example of a WWICE-entity that might use the information that the Activity Monitor provides, e.g. it might want to know when a person is involved in the activity 'Sleeping'.
Clients can subscribe with the Activity Monitor to be notified when an activity starts or stops, the location of the activity, and the identity of the person(s) who are involved in the activity. A client subscribes with a ({person}, {activity}, {location})-tupel. The set {person} is the set of persons of whom the client wants to know the activities. For example, a client might indicate that it wants to know the activities of the person 'Homer'. The set {activity} is the set of activities that the client is interested in. For example, a client might only be interested in the activity 'Sleeping'. The set {location} indicates where the activity should occur. For example, a client might only be interested in activities that occur in the living room. The Activity Monitor might not know with 100 % certainty whether an activity starts, who are involved in the activity, or where the activity occurs. It appoints a certainty factor to this information. For example, it might notify a client of the following information when it detects the activity 'Watching television':
Activity : Watching Television, certainty: 100%
Persons: Homer, certainty: 100%; Bart, certainty: 90%; Liza, certainty: 30 %
Location: Living room, certainty: 100%

The Activity Monitor subscribes with services that can provide information about the activities that it must monitor. For example, in order to detect the activity 'Having a shower', the Activity system subscribes with services (sensors) that can detect whether the shower is used. When one of these services has new information, it notifies the Activity Monitor. The Activity Monitor adds this observation as a fact to its knowledge database, called the Activity Fact Base. After each addition of a new fact, the Activity Monitor uses an inference mechanism to look at the total content of the Activity Fact Base to draw new conclusions. When it detects an activity, it sends a notification to its' clients (if necessary).
Further research must determine how the Activity Monitor should work exactly. The main research questions are what the abstraction level of an activity should be, and how the Activity Monitor can detect activities (e.g. what sensors are necessary to detect an activity).

7 CONCLUSIONS AND RECOMMENDATIONS

This thesis presents a typical example of Ambient Intelligence research. Ambient Intelligence research and related research areas such as Context Awareness [9] and Intelligent environments [10], are currently hot research topics. The ‘ambient’ part of this research project is that this project explores ways to use context information (e.g. the activities that a person is involved in), to give a person more control over her reachability. Furthermore, this project presents the architecture of a Reachability Management System (RMS) that can be incorporated into an ambient system. This chapter presents the conclusions and recommendations with respect to this thesis project.

7.1 CONCLUSIONS

From the work that is presented in this thesis, a number of conclusions can be drawn. These conclusions are presented below.

This thesis identifies aspects that influence one’s reachability. The choice for aspects is based upon a literature survey, discussions with supervisors, and interviews. Some important aspects are the type of communication, the priority of communication, and the activities that one is involved in. The aspects are used to create a realistic model that demonstrates how a user can control her reachability. The RMS uses this model to determine whether someone is available for communication.

Some important issues that played a part during the development of the model for Reachability Management, are that average consumers must be able to understand the model, and that the model must deal with contradictory information. For example, user preferences might indicate that the priority of communication must be both high (e.g. because the initiator is important) and low (e.g. because the subject of the communication is not interesting). The model that is presented in this thesis is easy to understand, and it indicates how the RMS should deal with contradictory information. Further research should refine the model based upon user tests.

This thesis identifies the basic functionality that a RMS must have with respect to Reachability Management in a home environment. It is possible to come up with more functionality, especially if one increases the scope to the office environment. However, the functionality that is presented in this thesis satisfies to demonstrate the use of a RMS.

This thesis presents the architecture of a RMS that introduces the components that are necessary to obtain the basic functionality of a RMS. This architecture defines the functionality of the components and indicates how this functionality can be implemented. Furthermore, the architecture shows how the components interact with each other, and how they interact with the entities of the WWICE system. This architecture satisfies to demonstrate the operation of a RMS and to demonstrate how the RMS can be implemented in the WWICE system.

The architecture that is presented in this thesis can easily be implemented in the WWICE system. Moreover, a substantial part of the architecture is not WWICE-specific. This means that the architecture can also be used to implement the RMS in another (home) system.

7.2 RECOMMENDATIONS

The work that is presented in this thesis can be used as a basis for further research. Further research could investigate the following issues:

Functionality of RMS and model for Reachability Management
In order to test the use of a RMS and the model for Reachability Management, one should implement the architecture of the RMS and conduct user tests that investigate (among other things) the following properties of the RMS:

- The benefit of the functionality of a RMS. In other words, does the RMS provide functionality that a user finds valuable?
- The clarity of the model for Reachability Management. In other words, is it clear to a user *how* the RMS determines whether she is available for communication?
- The benefit of the model for Reachability Management. In other words, does the model cover all preferences that a user wants to indicate with respect to her availability for communication, and does a user find the model easy to use?
- Privacy. What information should a RMS provide to people (e.g. should a RMS inform a caller that the callee is having a holiday), and what information is a RMS allowed to store (e.g. is the RMS allowed to store the number of times that a person is having a telephone conversation with someone)?
- Trust. How can the RMS make sure that people trust the RMS to manage their communication?

User Interface
From the interviews results that people are easily afraid that a RMS will block communication that they *do* want to receive. In important issue that increases the trust of people is the User Interface (UI) of the RMS. Further research should investigate how the UI can show in an easily understandable way, how the RMS manages the communication of a user. Furthermore, it must be very easy for users to indicate preferences and to indicate their current status via the UI of the RMS.

Activity Monitor
The architecture of the RMS takes the activities of the user into account. It retrieves these activities from an Activity Monitor that is assumed to be part of the WWICE system. However, the Activity system is not implemented in the WWICE system. Further research must determine how the Activity Monitor should work exactly. The main research questions are the abstraction level of an activity (e.g. whether the Activity Monitor most provide the information ‘user is sleeping’, or ‘user is not moving’), and how the Activity Monitor can detect activities (e.g. what types of information are necessary to detect an activity, and how can the Activity Monitor combine this information to determine that an activity occurs).

Exact operation of the components of the RMS
This thesis presents a thought-out architecture of a RMS. This architecture demonstrates the operation of a RMS and demonstrates how the RMS can be implemented in the WWICE system. However, further research is required to determine how the components must work in detail. For example, further research must investigate how the Learning module can determine new preferences (e.g. using neural networks or data mining techniques).

BIBLIOGRAPHY

[1] Aarts E.H.L. and Harwig H.A., Ambient intelligence: a new user experience. *Cebit 2000*, 2000.

[2] Aarts E.H.L., Ambient Intelligence: calming, enriching and empowering our lives. *Philips Research Password*, 8, 2001, http://www.research.philips.com/InformationCenter/Global/.

[3] Aarts E.H.L., Converting dreams into real experience. *World News*, 2001, http://www.research.philips.com/InformationCenter/Global/.

[4] Institute of the Future, "Sensors: The Next Wave of Infotech Innovation". *1997 Ten-Year Forecast*, 1997, http://www.iftf.org/html/iftflibrary/technology/sensors.pdf.

[5] Opleidingsdirecteur Technische Informatica, *Afstudeergids Technische Informatica*, 2001, http://academics.its.tudelft.nl/nl/TI/.

[6] Rannenberg K., "Multilateral Security A Concept and Examples for Balanced Security". ACM Press, Cork, Ireland, 2000, http://csrc.nist.gov/nissc/2000/proceedings/papers/202ra.pdf.

[7] Eggenhuizen H.H., *WWICE: Window on the World of Information, Communication and Entertainment*, Philips Research Password, 2000.

[8] Simons D. and Reuzel J., *WWICE Software architecture - only available within Philips Research*, CRE 1999, 1999.

[9] Dey A.K., Futakawa M., Salber D., and Abowd G.D. , "The Conference Assistant: Combining context-awareness with wearable computing". *Proceedings of the 3rd International Symposium on Wearable Computers (ISWC'99)*, 1999, http://www.cc.gatech.edu/fce/ctk/pubs/ISWC99.pdf.

[10] Meyers B., Brumitt B., Krumm J., Kern A., and Shafer S., "EasyLiving: Technologies for Intelligent Environments". *Handheld and Ubiquitous Computing*, 2000, http://research.microsoft.com/easyliving/publications.htm.

[11] Johannes S. and Marti W., *Active Messenger: Email Filtering and Mobile Delivery*. Thesis for the degree of Massachusetts Institute of Technology, 1999, http://citeseer.nj.nec.com/211552.html.

[12] Görg C., Farjami P., Bell F., Hagen L., Magedanz T., Vodslon M., Weckerle C., Vortisch W., Mauersberger J., Jaya S., Chandrasekaran V., Loryman M., Buckle P., Major B., Bretzke S., Hartmann J., Song W., Evensen R., Lülsdorf A., Kleier S., and Timphus F., *CAMELEON – Communication Agents for Mobility Enhancements in a Logical Environment of Open Networks*, 1998, http://www.comnets.rwth-aachen.de/project/cameleon/cameleon.html.

[13] Hartmann J., Gorg C., and Farjami P., "Agent Technology for the UMTS VHE Concept". 1998, www.jens-hartmann.de/papers/vhe.pdf.

[14] Raman B., Katz R.H. , and Joseph A.D., "Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network". Workshop on Mobile Computing Systems and Applications (WMSCA'00), 2000, http://www.cs.berkeley.edu/~bhaskar/research.html.

[15] Reichenbach M., Damker H., Federrath H., and Rannenberg K., "Individual Management of Personal Reachability in Mobile Communication". 13th International Conference on Information Security (SEC `97), 1997, http://citeseer.nj.nec.com/479072.html .

APPENDIX I LITERATURE SURVEY

I have identified four research projects that investigate (among other things) aspects that are of importance to Reachability Management. This appendix introduces each research project and explains for every research project the important aspects with respect to Reachability Management.

I.I ACTIVE MESSENGER

The Active Messenger project is a thesis project at MIT [11]. The primary design goal for the Active Messenger is to forward incoming text messages to available portable and stationary devices like text and voice pagers, cellular phones, etc. If necessary, the agent can send messages to several devices in turn, monitoring the reactions of the recipient and the success of the delivery. The Active Messenger project focuses on non real-time communication (e.g. email and SMS).

§ ***Description***

The Active Messenger is an agent that is capable of taking several steps over time to guarantee the delivery of a message, trying multiple channels and awaiting possible reactions of the recipient. Upon an incoming message, the Active Messenger executes the following primary tasks:

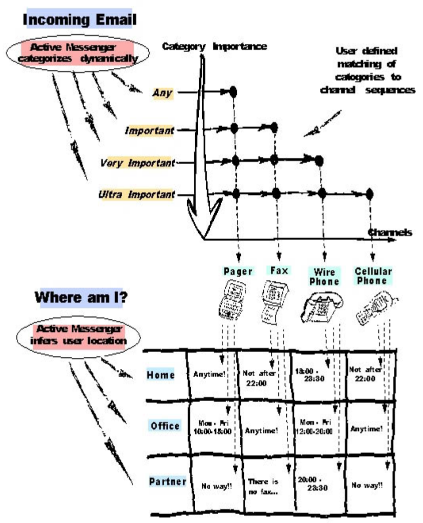
1. Determine which communication channels are active. In other words, which communication channels can be used for communication (e.g. is a computer on-line)?
2. Determine the importance of a message. Is it important enough to be forwarded to other channels?
3. If the message has to be forwarded, determine which is the appropriate communication channel.
4. Monitor the progress of the message delivery, and, if necessary, switch to alternative channels.

The current version of Active Messenger is implemented as a single PERL script. When Active Messenger starts up, it first loads the user preference file and then the stored email messages. After that, Active Messenger goes into an infinite loop that checks whether new messages arrive, and executes the four tasks that are mentioned above.

§ ***Important aspects with respect to Reachability Management***

The Active Messenger decides where to send a message, based on:

- The location of the recipient. The location is inferred by looking at the recipient's communication history and communication behavior.
- The importance of the message. The importance of the message is defined by rules that are specified by the recipient and by correlating the message with recent messages, the recipient's calendar, and her address book. Unfortunately, the Active Messenger project does not explain *how* the importance is calculated exactly.



The Active Messenger provides the following functionality:

- The system forwards a message, possibly to several devices in turn, and monitors the reactions of the recipient and the success of the delivery. For example, if a reply comes back shortly after a message is sent to a two-way capable device, the Active Messenger assumes that the user recipient read the message.

Figure 53 Determine the appropriate communication channel [11]

- The system may give feedback to the initiator about the status of the message delivery, the assumed location of the recipient, and the most recently used communication channels. This depends on the status of the initiator that is defined by the recipient.
- If a message is very important, the system will try at all costs to deliver the message, including calling up cellular phones and home phones with a text-to-speech module. This depends highly on the importance of a message that is previously determined by the recipient.

The Active Messenger is a nice example of a Reachability Management System. However, this project does not focus on the important aspects of one's reachability. The project merely focuses on the progress of message delivery. Unfortunately there are not many details available that explain *how* exactly the Active Messenger deals with the progress of message delivery.

I.II CAMELEON

The CAMELEON project[54] is an European project that investigates the application of Agent Technologies for the purpose of providing services anywhere in telecommunications networks [12], [13]. The goal of this project is to come up with the support and provision of new or enhanced services for the telecommunication market. An important part of the CAMELEON project is done at the University of Aachen and at the Ericsson Eurolab. The project deals with all types of communication (e.g. email, SMS, and telephone communication).

cameleon

Figure 54 Logo CAMELEON project

§ ***Description***

The use of Mobile Agents is a possible solution to the problem of service portability in heterogeneous networks. The Virtual Home Environment (VHE) is defined as a concept for personalised service portability across network boundaries and between terminals: users are consistently presented with the same personalised features, user interface and services, no matter what network or terminal (within the capabilities of the terminal) the user is using or where the user is located. To demonstrate this approach a set of services has been prototyped. The service that is of importance with respect to Reachability Management is the Adaptive Profile Manager. This service actively handles all incoming communication, e.g. it records or forwards calls.

§ ***Important aspects with respect to Reachability Management***

The CAMELEON project has identified some aspects that are of importance to the user profiles that define how incoming communication is handled. A user profile allows the recipient to set call handling options for incoming communication. The aspects that influence the user profile are:

- The recipient's state i.e. busy, idle, unavailable etc.
- The time of day.
- The identity of the initiator. A recipient may wish to specify a list of people for which she will be reachable and will expect all other calls to be rejected or forwarded.
- The recipient's location.

[54] For more information on the CAMELEON project, see http://www.comnets.rwth-aachen.de/project/cameleon/cameleon.html

- The type of service requested. The recipient may wish to dynamically specify how each type of incoming communication should be handled, e.g. reject particular fax messages, or forward particular e-mail messages to another person according to the sender or/and subject of each incoming e-mail.
- The class of terminal that the initiator is currently employing.
- The class of terminal that the recipient (if any) is currently employing.
- The type of network on which the recipient is currently registered.

Similarly for outgoing calls the handling of calls can be based upon but not limited to:

- The network on which the initiator is currently registered.
- The type of terminal that the initiator is currently employing.
- The class of terminal that the recipient (if any) is currently employing.
- The service requested.

Unfortunately, the literature that is available does not explain in detail *how* these aspects influence the user profile. The functionality of the Adaptive Profile Manager that is of importance to Reachability Management is:

- Handling of different end-systems with different capabilities for profile management.
- Provisioning of personalized call management capabilities for the recipient, i.e. the recipient can personalize the handling of calls by creating a set of rules (routing/filtering) as part of the user profile that the system uses to intelligently handle incoming communication requests. Examples of call management are:
  - Call (voice, fax, email) recording
  - Call rejection
  - Caller/application interactions via for example voice menu
  - Call forwarding/deflection
  - User registration/de-registration
  - Notification, e.g., via pager service, SMS, or email
  - Conversion of recorded voice- and fax messages for sending them as email
  - Forwarding of incoming email to a fax machine

The CAMELEON project focuses on UMTS and the system architecture that is needed to incorporate new or enhanced services into an UMTS network (e.g. agent architecture, billing, how services are offered to the user, etc.). However, the project has also identified some important aspects of one’s reachability and the Adaptive Profile Manager provides some interesting functionality that a Reachability Management System should deal with. Unfortunately there is not much information available that explains *how* the Adaptive Profile Manager deals with the aspects that influence the user profile or *how* the Adaptive Profile Manager implements it’s functionality.

I.III UNIVERSAL INBOX

The Universal Inbox is a project at Berkeley University that develops an architecture for the integration of communication services, based on the rapid growth of communication technology, characterized by new access networks (e.g. cellular, pager, wireless-IP) and end-devices (e.g. PDAs, two-way pagers, multi-model access devices) [14]. The project focuses on personal mobility and service mobility. Personal mobility is the ability to redirect communication across heterogeneous user devices, and service mobility provides access to services independent of the user’s end-point; i.e. the user sees the same set of services from all end-points. The Universal Inbox deals with all types of communication.

§ ***Description***

The three main goals of the Universal Inbox architecture are:

- Extensibility. It should be easy to add emerging communication services, and integrate them with all of the existing ones.
- Scalability. The system must scale to accommodate a large user community, and must be capable of operation on a global scale.

- Personalization. The user should be able to specify preferences for ubiquitous redirection of incoming communication.

In order to fulfill these goals, the architecture has three key functional capabilities:

- Any-to-any data transformation for accommodating diverse devices with different data formats.
- Storage and processing of user preferences for ubiquitous redirection of incoming communication.
- Device name translation and mapping to handle the heterogeneous name spaces like cell-phone numbers, pager numbers, IP-addresses, etc.

Figure 55 shows an example of the operation of the Universal Inbox.

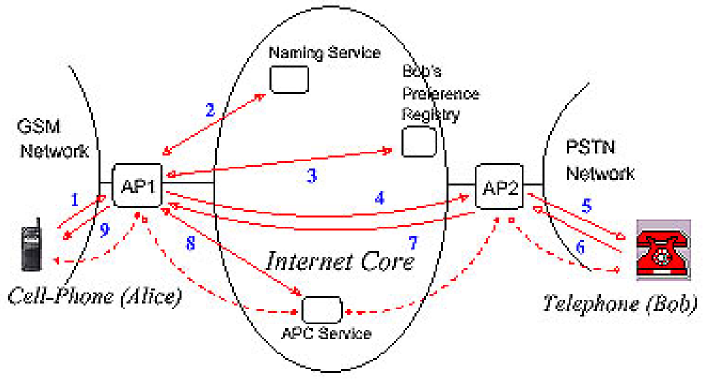


Figure 55 An example of the architecture of Universal Inbox [14]

When the initiator starts communication (1), the device of this person contacts the nearest Access Point (AP1). An Access Point provides gateway functionality for an access network: it exports a generic session setup interface to the Internet core. This Access Point retrieves the location of the recipient and the location of the preference registry of the recipient from the Naming Service (2). Next it contacts the preference registry to determine whether the recipient desires communication (3). Then it contacts the Access Point (AP2) near the recipient (4). This Access Point tries to setup the communication with the device that the recipient is using (5). Possibly, the APC service is used in order to transform one type of communication into another, e.g. to transform GSM into Voice over IP (8).

§ ***Important aspects with respect to reachability Management***

The Universal inbox project has not investigated the aspects of reachability. The functionality that the Universal Inbox has with respect to Reachability Management is the following:

- The initiator and the recipient can use every device for communication (when this is possible with respect to the capabilities of the device). The system takes care of the transformation from one type of communication into another.
- There is a redirection agent that contains user preferences that specify how a particular incoming communication should be handled. At this moment these preferences are implemented as a simple script, e.g.

```
IF (9AM $<$ hour $<$ 5PM) // At Office
THEN Preferred-End-Point = Office-Phone;
IF (5PM $<$ hour $<$ 11PM) // At Home
THEN Preferred-End-Point = Home-Phone;
IF (11PM $<$ hour $<$ 9AM) // Sleeping
THEN Preferred-End-Point = Voice-Mail;
```

The Universal Inbox project focuses on the implementation aspects of the architecture: network and device independence, data transformation and a naming service for handling heterogeneous devices and services that have different name spaces (e.g. UMTS's UPT number and an IP address). The project does not provide new insights with respect to Reachability Management.

I.IV REACHABILITY MANAGEMENT SYSTEM

Microsoft research and the university of Freiburg have developed a personal reachability management prototype called the Reachability Management System[55] (RMS^{MS}) [15], [6]. The RMS^{MS} is a concept for controlling personal reachability while maintaining a high degree of privacy and data protection. The prototype implementation serves to demonstrate the concepts of multilateral security and to examine their relation to the users needs. The term multilateral security is used to describe an approach aiming at a balance between the different security requirements of different parties [6]. The project mainly focuses on real-time communication. However the aspects of reachability that are identified by this research and the functionality of the RMS^{MS} are also applicable to non real-time communication.

§ *Description*

There are two strategies to obtain multilateral security: avoidance of data (data that does not exist or is not transmitted does not need to be protected) and careful allocation (give the storage and the processing of data into the control of those who require the confidentiality). The central idea enabling multilateral security in a call situation is the careful modeling of the communication context. The communication context contains all information that is of importance with respect to a call. The RMS^{MS} works as follows: recipients are able to control their personal reachability through their personal RMS^{MS}. During the signaling phase of a call the initiator transmits information concerning the nature and content of his communication request. The RMS^{MS} of the initiator and the RMS^{MS} of the recipient start negotiating. The connection with the recipient will only be established if the negotiated communication context has fulfilled certain conditions. If not, the RMS^{MS} is capable of offering a variety of reactions, for example storing a message, or diverting a call to another person. The main design aspects of the RMS^{MS} are the communication context and the representation of urgency, as well as the secure data processing and storage.

§ *Important aspects with respect to Reachability Management*

The communication context, the urgency, and the rules that are used to evaluate the communication context are of importance with respect to reachability. Unfortunately the evaluation rules are not explained in detail. A communication context contains the following information:

- How the initiator and the recipient are acquainted with each other. For example, anonymous, by a pseudonym, with their real identity.
- The intention of the communication request. In other words, the reason the initiator wants to start communication.
- The urgency of the communication request. In other words, the importance of the communication request.
- The manner of communication. For example telephone communication.
- The existing security requirements. This issue has to do with secure communication and is thus not within the scope of this graduation project.
- The mechanisms used to ensure the actual communication. This issue has to do with secure communication and is thus not within the scope of this graduation project.

One can indicate the urgency of a communication request as follows:

- The assertion of urgency. The initiator can indicate a certain degree of urgency.
- The specification of a function. The initiator can give details about the reason for his call, about his position, or even his qualification. For example, she may call as a member of a particular project or company.
- The specification of a subject. This specification can be evaluated by the RMS^{MS} when a prearranged list of possible topics exists.
- The provision of a reference. The initiator mentions the recommendation of a third person.
- The presentation of a voucher: The voucher differs from the reference in that the recipient has issued it herself. It may increase the chance of a return call.

[55] The RMS of Microsoft research and the university of Freiburg is denoted as RMS^{MS} to distinguish it from the RMS that is developed as part of this graduation project

- Offering a surety. The initiator may remit to the recipient an (possibly negotiated) amount of money as a surety. If the recipient does not agree with the initiator's evaluation of the urgency of his call, she has the potential to withhold this amount.

The rules of the evaluation come from three different areas:

- Status. The status describes the recipient's current situation (e.g. private, at work, meeting). The recipient indicates the status. This information changes frequently and determines which part of the rules will be applied.
- User specific reachability rules. The recipient can indicate individual evaluation rules that define how the reachability manager should react to incoming communication requests.
- Common reachability rules. The recipient cannot change these evaluation rules.

The following aspects are interesting with respect to the functionality of the RMS^{MS}:

- The RMS^{MS} negotiates both with the initiator as the recipient of a communication request.
- When the RMS^{MS} determines that communication is not desired, the RMS^{MS} sends appropriate messages to the initiator and might even propose new actions. Two actions are mentioned: storing a message and diverting a call to another person.
- The RMS^{MS} takes care of security-issues. Two strategies are possible: avoidance of data and careful allocation.

Microsoft research did some user tests. 31 “expert test persons” from different healthcare organizations participated. The most important results are that users happily accepted the opportunity for controlling their own security even though this introduced extra complexity. Some users never changed the pre-configured situation rule sets (“connect every call”, “no calls”, and “meeting”). Many participants created some new situations or changed rules in existing situations. Some users created a large number of situations in advance (e.g. “visiting a patient”, “office work”, or “stand-by”) but reduced this number later after having gained more experience. The system should not become too complex since complexity decreases the perceived usefulness of the system dramatically.
The RMS^{MS} research at Microsoft research and the university of Freiburg has a strong relationship with my graduation project. However, the focus of this research is on the security aspects of communication, while the focus of my graduation project is on the important aspects of reachability and Reachability Management in general.

APPENDIX II INTERVIEWS

As part of this graduation project, eleven people have been interviewed: the secretary of the Media Interaction group, four PhD students, four MsC students, and two other people. These people filled in a questionnaire. Based upon this questionnaire, these people have been interviewed to in order to obtain their opinion with respect to Reachability Management. The interviews have helped a lot to determine when people are available for a particular type of communication, and to determine what functionality people expect from a Reachability Management System. Some interesting results are the following:

- People want to have control over their reachability. This means that people want to be able to indicate preferences for everything that influences their reachability. The RMS should not automatically adjust preferences.
- People do not want to indicate many preferences. For example, they want to have mostly the same preferences for all real-time and for all non real-time communication.
- The priority^r of communication depends mainly on the priority^i and the identity of the initiator. Not many people want the RMS to take into account the subject, since this would require the user to indicate too many preferences.
- Whether or not someone is available for communication depends mainly on the priority^r of the communication, the status, and the type of communication.
- The system should be easy to use. People want to indicate their reachability with 'one push on a button'.
- People want to be able to indicate that they are always available for some persons.
- An initiator must always be able to 'override' the system such that the RMS does notify the recipient. In other words, people always want to be available for emergencies.
- The system should never (accidentally) block any communication that the user would like to receive. If this happens, even if it happens only a few times, people will not use the system.
- Privacy is very important to people. People do not want the system to provide any personal information to initiators. For example, the system should only inform an initiator that the recipient is 'not available'. It should not provide more information, e.g. what the recipient is doing. Furthermore, the system should not keep track of personal information, such as with whom people are calling.
- For real-time communication, the RMS should use a device that is close to the user for the UI of the notification. For non real-time communication, the RMS should use either a device close to the user, or a specific group of devices.
- If the recipient is not available for real-time communication, the RMS should propose other types of communication to the initiator (e.g. voice mail, email, etc.).
- People want to receive a notification of non real-time communication, even if it is not possible to start non real-time communication at that moment.

The next page shows the questionnaire that is used as a basis for the interviews.

Questionnaire “Reachability Management System”

My name is Johan Ensing and I’m working on my graduation project at Philips Research. I am developing a software system that helps you to control your reachability with respect to communication such as telephone calls and email messages. You could see this system as your own digital secretary. For example, the system enables you to indicate that you do not want to be disturbed by communication for a while (e.g. because you are going to sleep). On the other hand, the system can make sure that you are always reachable for communication by redirecting communication to devices that are nearby you. This system will be part of the ‘home-of-the-future’ of Philips Research.

By means of this questionnaire I’m trying to determine the functionality that the system should have and how the system can determine whether or not you want to start communication. Please pay some attention to the open spaces that can be used to explain your answer in some more detail. Often you can choose multiple answers. These questions are marked with an *. I would like to use your answers to discuss the topics that are dealt with by the questions.

Thanks in advance,

Johan Ensing

Question 1 When new communication arrives at the system (e.g. a telephone call or an e-mail message), the system must determine how important the communication is according to you. I call this the priority of the communication. How would you like to define priority?

☐ Classification, for example ‘Low’, ‘Normal’, ‘High’, ‘Emergency’.
☐ Percentage (0-100%). This percentage corresponds with a scale from ‘not important‘ to ‘very important‘.
☐ ....

Explanation: ....
....
....

Question 2* How should the system determine the priority of the incoming communication? The priority depends on:

☐ The importance of the communication according to the person who wants to start the communication. For example, this person might indicate that it is an emergency.
☐ The identity of the person who wants to start the communication. For example, your girlfriend might always have a high priority.
☐ The subject of the message. For example, when the subject is ‘meeting’, the communication might have a high priority.
☐ ....
☐ ....

Explanation: ....
....
....

Question 3* Suppose that there is new communication and the system determines that it should notify you of this communication. The system must determine how soon you want to know about the communication (e.g. immediately or some time in the future). How soon you want to know about the communication depends on:

- ☐ The type of communication. For example, you might immediately want to know about an incoming telephone call, but you might not immediately need to know about a new e-mail message.
- ☐ The importance of the communication according to you. For example, you might immediately want to know when a high priority message arrives.
- ☐ The time that the system has received the communication. For example, when the system has received an email message and you have not read the message within a day, the system might make sure that you read the message within an hour.
- ☐ ....
- ☐ ....

Explanation: ....
....
....

Question 4* The system might use a lot of different devices in your home to notify you that new communication has arrived. For example, it might use the device that you are currently using (if any), or a specific device (e.g. a screen in the kitchen). The system might adapt which devices it uses, based upon how soon you want to know about the communication. For example, it might use the device that you are currently using if you want to know about the communication immediately. Which devices should the system use for the notification? Please use the line below your answer(s) to explain when the system must use these devices.

☐ The system must use a device that is close to me.

....

☐ The system must use one device per room.

....

☐ The system must always use the same set of devices. For example, it must use a screen in the kitchen and a screen in the living room.

....

☐ ....

Question 5* Sometimes people do not desire to start communication when they are involved in an activity. During which activities would you not want to be disturbed by communication?

- ☐ When you are sleeping.
- ☐ When you are having a shower.
- ☐ When you are talking to somebody.
- ☐ When you are tired.
- ☐ ....

Explanation: ....
....
....

Question 6 When new communication arrives, the system must determine to what extent you want to start communication. I call this your status. How would you define your status?

- ☐ Classification, for example ‘Available’, ‘Busy’, ‘Working’, and ‘Not available’.
- ☐ Percentage (0-100%). This percentage corresponds with a scale from ‘not available‘ to ‘available‘.
- ☐ ....

Explanation: ....
....
....

Question 7* How should the system determine your status at a certain point in time?

- ☐ You could indicate you status yourself, at a certain point in time. For example, you could select your current status using a GUI.
- ☐ You could indicate your status during specific periods, e.g. your status is 'not available' between 0:00h and 7:00h.
- ☐ You could indicate the activities you are involved in. For example, you could indicate that you will be sleeping for the next 8 hours. The system must use this information to determine your status.
- ☐ The system must detect activities and use this information to determine your status.
- ☐ ……………………………………………………………………………………

Explanation: ……………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

Question 8* When new communication arrives at the system, it must determine whether it should notify you. In other words, it must determine whether you are reachable for this communication. How should the system determine this?

- ☐ It should take into account my status (question 7)
- ☐ It should take into account the priority of the communication (question 2)
- ☐ It should take into account within what period of time I want to know about the communication (question 3)
- ☐ It should take into account who wants to start communication. For example, when your status is 'Working', you might not want to be disturbed by your mother-in-law, but you do want to communicate with colleagues.
- ☐ ……………………………………………………………………………………
- ☐ ……………………………………………………………………………………

Explanation: ……………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

Question 9* Suppose that new communication arrives (e.g. a telephone call) and the system determines that you do not want to be disturbed. What should the system do? Please explain when the system should choose which action.

- ☐ The system should inform the one who is calling that you are not available.
- ☐ The system must explain why you are not available
- ☐ The system must propose to 'overrule' the decision of the system, e.g. by increasing the priority of the call.
- ☐ The system must propose to forward the communication.
- ☐ The system must propose different types of communication, e.g. e-mail, voice-mail, or SMS.
- ☐ The system must inform the one who is calling when you will be available.
- ☐ ……………………………………………………

Explanation: ……………………………………………………………………………
……………………………………………………………………………………………
……………………………………………………………………………………………

Question 10* What do you expect that a reachability management system does for you? What functionality should this system have?

- ☐ Reject incoming communication when it determines that you do not desire the communication.
- ☐ Forward communication if the system cannot reach you.

☐ Record communication if you do not desire to start direct communication (e.g. record a voicemail message)
☐ ………………………………………………………………………………………………
☐ ………………………………………………………………………………………………
☐ ………………………………………………………………………………………………

Question 11* What problems should the system solve?

☐ The system must make sure that I am not disturbed by communication when I do not desire communication.
☐ The system must make sure that I am aware of new communication in time (e.g. that I know about important messages in time)
☐ The system must make sure that people can send a message when I do not desire direct communication (e.g. telephone communication).
☐ ………………………………………………………………………………………………
☐ ………………………………………………………………………………………………

Question 12* What should the system definitely not do?
……………………………………………………………………………………………………
……………………………………………………………………………………………………
……………………………………………………………………………………………………
……………………………………………………………………………………………………

Thanks for filling out this questionnaire!

APPENDIX III DETAILS ARCHITECTURE OF RMS

This appendix explains the operation of some components of the RMS in more detail. Section III.I deals with the interaction schemes that the Interaction module constructs. Section III.II states two examples of the interface of a Transformation module, and finally, section III.III shows two examples of adaptive behavior of the Learning module.

III.I INTERACTION SCHEMES

This section explains the interaction schemes that the Interaction module constructs. Section III.I.I explains the schemes that the Interaction module construct when it deals with incoming communication, and section III.I.II explains the schemes that the interaction module constructs when it deals with starting communication. An interaction scheme consists of the following parts:

- The information that must be transferred, e.g. the Interaction module can inform the recipient of a call of the identity of the initiator and the priority[r] of the communication.
- The options that a person can choose from, e.g. a recipient can choose to start communication or to propose a different type of communication to the initiator.

The Interaction module asks the Preferences module for the preferences with respect to the interaction. It uses these preferences to construct an interaction scheme. For example, the preferences might indicate that the Interaction module must propose to some initiators the option “forward communication to mobile phone”, and to other initiators the option ”Record voice-mail”. The preferences also influence the information that the Interaction module provides. For example, the Interaction module might inform some initiators when the recipient will be available (e.g. the recipient is having a holiday until the 5th of December). The interaction schemes always contain the following options:

- Start using a different device. A person that interacts with the UI of the RMS, might want to start using a different device. For example, because she wants to start communication at a different device. If the user chooses this option, the Interaction module asks the User Interface module to start a notification on every device that the person can use (for the communication that she wants to start). The Interaction module stays in its’ current state.
- Stop RMS. If a person chooses this option, this means that the she does not want the RMS to deal with the communication request anymore. For example, she does not want it to show a notification of a communication request regarding non real-time communication anymore, or she rejects a communication request regarding real-time communication[56]. If the initiator and the recipient were negotiating, the Interaction module constructs an interaction scheme to inform the other party of the cancellation. It sends this scheme to the RMS of the other party. This scheme does not contain any options. Next, it informs the Request Handler that there are no further actions necessary, and stops the interaction.
- Change status. If a *recipient* chooses this action this means that she does not want (notification of) the communication. The Interaction module of the RMS of the recipient sends the new status to the Status module. If the communication request regards real-time communication, the Interaction module constructs an interaction scheme to inform the other party of the cancellation and sends this scheme to the RMS of the other party. Next, it informs the Request Handler that no further actions are necessary, and stops the interaction. If an *initiator* chooses this action, this does not change her reachability for the current communication request (since the initiator wants to start communication). The Interaction module of the initiator sends the new status to the Status module, and stays in its current state.

[56] User tests must determine whether one rejects only *real-time* or *all* communication when one stops the RMS. I think that it is best to reject only real-time communication since one can take notice of non real-time communication whenever one likes, thus this communication is not really disturbing.

III.I.I Incoming communication

This section explains the interaction schemes that the Interaction module of the RMS of the *recipient* constructs.

Scheme 1. Start notification
If the Interaction module of the recipient receives this event, this means that it must notify the recipient (the user of the RMS) of new communication. There are two possibilities: the communication is real-time or it is not. If the communication is not real-time, the Interaction module constructs an interaction scheme, sends it to the User Interface module and goes into the ‘Interacting with recipient’ state. This interaction scheme contains the following information[57]:

- The identity of the recipient. For whom is the communication?
- The type of communication or an indication that the communication is not real-time.
- The priority[r]. Is the communication important according to the recipient? When there are multiple messages, the Interaction module should show the highest priority[r].
- The number of new messages
- Whether the recipient can start the corresponding communication application.
- Current status of the recipient

The interaction scheme contains the following actions[58]:

- Start communication application(s). If the recipient chooses this action, the Interaction module asks the Request Handler to start the appropriate communication application. Next, it goes into the ‘Waiting for Request Handler’ state.

If the communication is real-time, the Interaction module constructs an interaction scheme based upon the scheme that it received from the RMS of the initiator (Scheme 5). If the initiator does not have a RMS, the Interaction module constructs this interaction scheme itself. It sends the scheme to the User Interface module and goes into the ‘Interacting with recipient’ state.

Scheme 2. Propose communication to initiator
The recipient does not want to start the communication that the initiator proposes, but she proposes a different type of communication to the initiator. The Interaction module creates an interaction scheme, sends it to the RMS of the initiator, and goes into the ‘Interacting with initiator’ state to wait for the response of the initiator. The interaction scheme contains the following options (based upon the preferences of the recipient):

- Accept proposed communication. If the initiator chooses this action, the Interaction module of the RMS of the recipient asks the Request Handler to start the proposed type of communication and goes into the ‘Waiting for Request Handler’ state.
- Propose different type of real-time communication[59,60]. If the initiator chooses this action, the Interaction module of the RMS of the recipient forwards the interaction scheme that it receives from the RMS of the initiator to the User Interface module (Scheme 5), and goes into the ‘Interacting with recipient’ state to inform the recipient of the proposed type of communication.
- Start non real-time communication[60,61]. If the initiator chooses this action, the Interaction module of the RMS of the recipient receives an interaction scheme that

[57] If this information is available, e.g. there is an Information module that can extract this information from the message.
[58] Note that the introduction of this section states some options that an interaction scheme always contains, e.g. to stop the RMS.
[59] When the recipient and the initiator are negotiating to start communication, the initiator can select all types of real-time communication that the Request Handlers of both RMS-es can start, and that are not explicitly rejected by the recipient.
[60] The initiator can choose to start non real-time communication that the recipient does *not* want to start, if the Address book database contains the communication address of the recipient for that type of communication (e.g. the e-mail address).
[61] The initiator can choose from the types of non real-time communication that are supported both by the (WWICE) system of the recipient as the (WWICE) system of the initiator (e.g. both systems might support e-mail communication).

informs the recipient that the initiator started non real-time communication. If forwards this scheme to the User Interface module and goes into the 'Interacting with recipient' state to wait for the response of the recipient.

- Forward communication. For example, forward telephone communication to the mobile phone of the recipient or forward email communication to the email address at work. If the initiator chooses this action, the Interaction module of the recipient asks the Request Handler to forward the communication and goes into the 'Waiting for Request Handler' state.
- Increase priority[i] [62]. If the initiator chooses this action, the Interaction module of the recipient forwards the interaction scheme (Scheme 5) that it receives from the RMS of the initiator to the User Interface module. Next, it goes into the 'Interacting with recipient' state to wait for the response of the recipient.
- Change subject[62]. If the initiator chooses this action, the Interaction module of the recipient forwards the interaction scheme (Scheme 5) that it receives from the RMS of the initiator, and sends it to the User Interface module. Next, it goes into the 'Interacting with recipient' state to wait for the response of the recipient.

Scheme 3. Start interaction with initiator

If the Interaction module receives this event, this means that the Request Handler of the recipient has determined that the recipient is not reachable and that it should inform the initiator. The Interaction module of the recipient constructs an interaction scheme and sends it to the RMS of the initiator. The interaction scheme contains the following information:

- The fact that that the recipient is not available for the communication request.
- Possibly, the Interaction module might inform the initiator that the RMS automatically rejected the request and why (status). However, due to privacy reasons the Interaction module might not give this information.

The interaction scheme contains the following actions:

- Accept proposed communication. Explained in Scheme 2.
- Propose different type of real-time communication. Explained in Scheme 2.
- Start non real-time communication. Explained in Scheme 2.
- Forward communication. Explained in Scheme 2.
- Increase priority[i]. For every type of communication, the Interaction module might indicate what minimum priority[i] the communication needs such that the recipient is available for that type of communication (if any). The rest is explained in Scheme 2.
- Change subject. For every type of communication, the Interaction module might indicate what subject the communication needs such that the recipient is available for that type of communication (if any). The rest is explained in Scheme 2.

[62] The recipient can reject some type of communication and indicate what minimum priority[i] or subject that communication needs, such that she is available for that communication. The initiator can change respectively the priority[i] or the subject to propose this communication again. Of course, the initiator can also change the priority[i] or the subject and choose a different type of communication.

III.I.II Starting communication

This section explains the interaction schemes that the Interaction module of the *initiator* constructs.

Scheme 4. Gather information

The Request Handler of the initiator asks the Interaction module to interact with the initiator in order to obtain the information that is necessary to start communication (e.g. the type of communication and the identity of the recipient). The Interaction module constructs an interaction scheme and sends it to the User Interface module and goes into the 'Interacting with recipient' state. The scheme does not contain information. It only contains the following actions:

- Indicate the identity of the initiator (if the Interaction module does not know this yet).
- Indicate the identity of the recipient.
- Indicate the proposed type of communication.
- Indicate communication address[63].
- Indicate the priority[i].
- Indicate the subject of the communication.
- Start communication[64]. The initiator can choose this action as soon as all information that is necessary to send a communication request is available (e.g. the identity of the recipient and the type of communication). This causes the following event to happen: 'Propose communication to recipient' (Scheme 5).

Scheme 5. Propose communication to recipient

If the Interaction module of the RMS of the initiator receives this event, this means that the initiator proposes to a recipient to start real-time communication. If this is the first proposal, the Interaction module sends the interaction scheme to the Request Handler such that it can send a communication request. If the proposal is part of a negotiation between the initiator and (the RMS of) the recipient, the Interaction module sends the interaction scheme to the RMS of the recipient. In both situations, it goes into the 'Interacting with recipient' state. The scheme contains the following information[65]:

- The identity of the recipient. For whom is the communication?
- The identity of the initiator. From whom is the communication request?
- The type of communication[66]. What type of communication is proposed?
- The priority[r]. Is the communication important according to the recipient?
- The subject of the communication. What is the communication about?
- A list of the types of *real-time* communication that the recipient may propose [67].
- A list of the types of *non real-time* communication that the recipient may propose[68].
- Current status of the recipient

The interaction scheme contains the following actions:

- Accept communication. If the recipient chooses this action, the Interaction module of the RMS of the initiator asks the Request Handler to start the appropriate communication application, and goes into the 'Waiting for Request Handler' state.

[63] If the Interaction module cannot obtain the communication address (for the RMS of the recipient or for the type of communication) from the Address book database, it asks the initiator to indicate this.
[64] Obviously, the initiator *must* indicate some information before it can start communication.
[65] If this information is available, e.g. the initiator might not indicate the subject of the communication.
[66] If the initiator and the recipient are already negotiating, the interaction scheme only needs to contain the type of information that the initiator proposes.
[67] The items in this list are the types of real-time communication that the Request Handler of the *initiator* can start at that moment in the home system of the *initiator*. The Request Handler of the RMS of the *recipient* uses this list to construct the list of the types of communication that the Request Handler of the RMS of the *recipient* can start in the home system of the *recipient*.
[68] This list contains the type of non real-time communication that the home system of the initiator supports. The Interaction module of the recipient uses this list to construct the list of communication that recipient can propose to start.

- Propose different type(s) of communication [69]. If the recipient chooses this action, this means that the recipient rejects the proposed type of communication. She indicates what types of communication she *does* want to start, and possibly which types of communication she *does not* want to start. For example, she proposes telephone communication instead of video communication. The Interaction module of the RMS of the initiator receives an interaction scheme from the RMS of the recipient (Scheme 2). It forwards the scheme to the User Interface module, and goes into the 'Interacting with initiator' state.
- Indicate minimum priority[i]. The recipient can indicate what minimum priority[i] communication needs to have, such that she is available for that type of communication. If the recipient chooses this action, the Interaction module of the RMS of the initiator receives an interaction scheme from the RMS of the recipient (Scheme 2). It forwards this scheme to the User Interface module and goes into the 'Interacting with initiator' state.
- Indicate subject. The recipient can indicate for some type of communication what subject that communication needs, such that she is available for that type of communication. If the recipient chooses this action, the Interaction module of the RMS of the initiator receives an interaction scheme from the RMS of the recipient (Scheme 2). It forwards this scheme to the User Interface module and goes into the 'Interacting with initiator' state.

Scheme 6. Time-out (start interaction initiator)

If the Interaction module receives this event, this means that the RMS of the recipient is not responding and thus there is a system error at the recipient's side. The Request Handler determines that it should start interaction with the initiator. The Interaction module constructs an interaction scheme to inform the initiator that the RMS of the recipient does not respond. This interaction scheme contains the following actions:

- Start communication. The initiator can decide to start other communication. The Address book database must have the communication address for that communication (since the RMS of the recipient does not respond). If the initiator chooses this action, the Interaction module asks the Request Handler to start the type of communication that the initiator wants to start, and goes into the 'Waiting for Request Handler' state.
- Indicate communication address. If the initiator indicates the communication address of the recipient, the Interaction module stores the communication address at the Address book database. It asks the Request Handler to start the type of communication that the initiator wants to start, using the communication address that the initiator provided. Next, it goes into the 'Waiting for Request Handler' state.

[69] The recipient can only choose communication that is in the lists that are explained in the 'information'-part of the interaction scheme.

III.II TRANSFORMATION MODULE

Section 4.5 explains the Transformation module. This section explains some examples of the interface that a Transformation module provides to the other components of the RMS. This interface consists of transformation- and input-parameters. The transformation-parameters indicate what type of transformation the Transformation module performs. Other components of the RMS can use the transformation parameters to find the Transformation module that provides the service that they need. The input-parameters indicate what information the Transformation module needs to be able to execute the transformation. This section explains two examples of a service: the 'Text-to-voicemail' service and the 'video-to-videomail' service. Of course, the actual interface will be different (for example, it might also need an input-parameter that indicates at what location it should store the result). However, these examples give a good idea how the interface could look like.

§ ***Text-to-voicemail***

This services transforms email into voicemail. The Transformation module that offers this service provides the following interface:
Transformation: From <format: [text]> to <type of communication: voicemail>, <format: [mp3]>
Input: <data>, <type of communication>, <Identity of initiator>

The first transformation parameter indicates that the Transformation can only transform text-data. The next parameters indicate that this module transforms the data into voicemail communication that is encoded in mp3. The input parameters indicate that the Transformation module needs respectively the data that must be transformed, the type of communication (e.g. email), and the identity of the initiator (e.g. Bill). When the Transformation module starts the transformation, it might add a line to beginning of the text, such as "You received an email message from Bill. The message is…" Next, it transforms the text into audio, encodes it into mp3, and stores the result in a file: the output is a voicemail message that is encoded in mp3.

§ ***Video-to-videomail***

This services transforms video into videomail. The Transformation module that offers this service provides the following interface:
Transformation: From <format: [H323], [Mpeg4]> to <type of communication: videomail>, <format: [Mpeg4]>
Input: <type of format: [H323], [Mpeg4]>, <address of video channel>

The first transformation-parameter indicates that the Transformation module can transform both H323- and Mpeg4 streams. The next transformation-parameters indicate that this module transforms an incoming stream into a video mail message that is encoded in Mpeg4. The first input-parameter states that one must indicate the format of the incoming stream (e.g. whether it is Mpeg4 or H323). The next input parameter indicates that the Transformation module needs the address of the video channel that will provide the video stream. When the Transformation module starts the transformation, it transforms the incoming video stream in Mpeg4 (if necessary) and stores the result in a file. Now the transformation is done: the output is a videomail message that is encoded in Mpeg4.

III.III LEARNING MODULE

Section 4.9 explains the Learning module. The Learning module is the component of the RMS that makes the system 'intelligent'. The Learning module tries to determine new user preferences that make the system comply with the behavior of the user even better. This section explains two examples of preferences that the Learning module might try to adjust. These preferences are the preference 'Devices preferred for video communication' and the preference 'Status when watching television'.

§ ***Devices preferred for video communication***

A user of the RMS can indicate that she prefers to use some particular devices for particular types of communication in a particular room. For example, she can indicate that in the kitchen, she prefers to use the main screen for video communication. If an incoming communication request for video communication arrives, the Request Handler searches in every room for a device that it can use for communication and notification. If the user has indicated preferences that determine which device(s) it should investigate, it first investigates these devices. If the user has not indicated any preferences, the Request Handler arbitrarily investigates devices.

The Learning module can learn from the user behavior to determine which device(s) the user prefers to use for a particular type of communication in a particular room. It is important that the Request Handler has a good search order to search for devices, since this it takes time to investigate many devices and time is very important if the Request Handler must deal with a communication request regarding real-time communication. In order to determine the search order, the Learning module subscribes with the User Interface module for the following information under the following conditions:

Condition: Every time a user chooses the action 'Start communication' (when some other party sent a communication request).
Information:
- The identity of the user.
- The ID and location of the device that shows the UI that the user uses at that moment.
- The type of communication the user wants to start.

Condition: Every time a user starts interaction with a UI to start communication (the user starts a communication request).
Information:
- The identity of the user.
- The ID and location of the device that the user is using.
- The type of communication the user wants to start.

The Learning module stores the information that it retrieves from the User Interface module. For every room, it keeps track of how often a particular user uses a particular device for a particular type of communication. The more often a user uses a particular device to start a particular type of communication, the higher that device is ordered in search order for that room. When the Learning module determines that a particular device should be higher in a preferences list, it automatically adjusts this preference. In other words, it sends the new preference to the Preferences module without consulting the user. It does not consult the user since this type of preference does not change the reachability of the user: the Request Handler still searches for a device in every room. Only if it determines that a particular device must be higher in the list than a device that is put in the list by the user, it proposes the new preference to the user herself.

§ ***Status when watching television***

The user can indicate that she wants a particular status to be active during some activities. However, it might be too much trouble to indicate this for every activity. This section explains how the Learning module might determine the status of the user when the user is watching television. For example, a user might want to have the status 'Busy' when she is watching her favorite television programs. The Learning module can try to determine the favorite television programs of the user. This is only possible if there is a 'Television application' in the home system of the user that can indicate what television program it is showing, and at which device is showing this program. The Learning module subscribes with the User Interface module for the following information under the following conditions:

Condition: Every time a user changes her status to a status that is *not* ‘Available’.
Information:

- The identity of the user
- The new status of the user
- The location of the user

Condition: Every time a user rejects communication
Information:

- The identity of the user
- The location of the user

Every time the User Interface module notifies the Learning module, the Learning module subscribes with the Television application for the following information under the following conditions:

Condition:

- The Television application is showing a program at that moment.
- The location of the device that the application is using to show the program is the same as the location of the user (the Learning module got this location from the User Interface module).

Information:

- The ID of the program
- The identity of the person(s) that watch the program (if it can provide this information)

If it turns out that a particular user always changes her status or rejects real-time communication when she is watching a particular program, the Learning module asks the Preferences module to propose the following preference to the user: change status to ‘Busy’ when the Television application is showing that particular program and the system detects that user is watching that program (e.g. the user is in the same room as the device that is displaying that program). The algorithm that is described above is not flawless (e.g. the Learning module must not determine that the status of the user is ‘Busy’ when the Television application is broadcasting advertisements). However, this algorithm does give a good idea how the Learning module might determine a new preference.

[1] Rannenberg K., "Multilateral Security A Concept and Examples for Balanced Security". ACM Press, Cork, Ireland, 2000, http://csrc.nist.gov/nissc/2000/proceedings/papers/202ra.pdf.

APPENDIX IV DESIGN DECISIONS DEPLOYMENT

This appendix explains the deployment of the components of the RMS in detail. The deployment of the components is based upon the criteria for deployment that are stated in section 6.3. Every section of this appendix deals with the deployment of a component with respect to the component(s) that use the functionality of the component. For example, the Request Handler is the only component that uses the functionality of the Interaction module. Therefore, the section that deals with the deployment of the Interaction module, explains the deployment of the Interaction module with respect to the Request Handler. This way, one can determine the deployment of all components with respect to each other.
A user does not like to install many software components. The more components are part of one software application (the ‘RMS’), the easier it is for the user to install the RMS. This means that preferably, all components of the RMS are part of one application that a user can install at one device. Only when it is not possible to install a component as part of the ‘RMS’, or when external entities (entities that are not a component of the RMS) must use the component, then that component has a distributed deployment with respect to the RMS. This means that one installs that component as an independent WWICE-service.

§ ***Request Handler***
A Request Handler executes a task for the Task manager, e.g. it handles a communication request. The Task manager is the only component that uses the functionality of the Request Handler. Therefore, this section explains the deployment of the Request Handler with respect to the Task manager. The Request Handler has the following properties:

- The Request Handler is very important to the Task manager, since it executes the tasks that the Task manager must deal with.
- The Request Handler does not need a lot of resources (e.g. CPU, memory, and storage), since it only has to do simple calculations, such as calculating the priority[r] of communication.
- The Task Manager takes care of the communication between the Request Handler and other Request Handlers (that are part of a different RMS). The size of these messages is approximately between 1 Kb and 20 Kb. Therefore, there is not a high information flow between the Request Handler and the Task manager.
- The Task Manager takes care of the communication between the Request Handler and Request Handlers that are part of a different RMS. If the Request Handler must deal with a communication request regarding real-time communication, it must send a response to the Request Handler that is part of the RMS of the initiator as fast as possible, since an initiator does not like to wait very long for a response. Therefore, the communication delay between the Request Handler and the Task manager must be low.

The Request Handler must have a local deployment with respect to the Task manager. The main reasons for this decision are that Request Handler does not need a lot of resources and there is a need for a low communication delay.

§ ***Interaction module***
The Interaction module determines how it must interact with a particular person (e.g. via a User Interface module or via the RMS of that person), and it determines the interaction scheme. The Request Handler is the only component that uses the functionality of the Interaction module. Therefore, this section explains the deployment of the Interaction module with respect to the Request Handler. The interaction module has the following properties:

- A Request Handler that deals with a communication request really needs the Interaction module for the execution of its’ task. If it cannot start an Interaction module, it cannot start interaction with either the recipient or the initiator, thus it cannot execute its’ task.
- The Interaction module does not use a lot of resources (e.g. CPU, memory, and storage), since it mainly constructs interaction schemes. This is a simple task.

- There is primarily communication between the Interaction module and the Request Handler, if the Interaction module must send an interaction scheme to another Interaction module. It sends this interaction scheme via the Request Handler. The size of an interaction scheme is between approximately 1 Kb and 20 Kb. Therefore, there is no high information flow between the Request Handler and the Interaction module.
- If the Request Handler must deal with a communication request regarding real-time communication, it must send a response to the RMS of the initiator as soon as possible. The Interaction module is the module that takes care of this response (since it determines the interaction scheme). Therefore, the communication delay between the Request Handler and the Interaction module must be low.

Based upon the properties of the Interaction module and the criteria for deployment, the Interaction module must have a local deployment with respect to the Request Handler. The main reasons for this decision are that the Interaction module does not need a lot of resources and there is a need for a low communication delay.

User Interface module
The User Interface modules take care of the UI of the RMS. The Interaction module is the only component that uses the User Interface modules. Therefore, this section explains the deployment of User Interface modules with respect to the Interaction module. It is important to distinguish between the type of User Interface module(s) that a RMS *must* have (the internal User Interface module) and the User Interface modules that provide some additional functionality (the external User Interface modules). This section deals with these two types of User Interface modules separately.

Internal User Interface module
The RMS *must* have a User Interface module that is able to use the UI managers of the WWICE system to create a UI (e.g. a GUI). This User Interface module creates UI descriptions and sends these UI descriptions to the Cluster managers that manage the cluster where it wants to create a UI. This type of User Interface module has the following properties:

- This User Interface module is very important to the Interaction module, since it provides the basic UI-functionality that the Interaction module needs to start interaction with a user of the RMS (e.g. such that a user can start communication).
- This User Interface module does not use a lot of resources (e.g. CPU, memory, and storage), since it only needs to construct UI descriptions.
- There is mainly communication between the User Interface module and the Interaction module, when a person starts to use the UI of the User Interface module. In this situation, the User Interface module informs the Interaction module which option from the interaction scheme the user has selected. The Interaction module constructs a response (a new interaction scheme) and sends it to the User Interface module. The size of an interaction scheme is between approximately 1 Kb and 20 Kb. Therefore, there is not a high information flow between the User Interface module and the Interaction module.
- When a user starts to interact with the UI, she expects to see a response from the RMS within a second. Therefore, there is a need for a low communication delay between the User Interface module and the Interaction module.

Based upon the properties of the Interaction module and the criteria for deployment, the internal User Interface module must have a local deployment with respect to the Interaction module. The main reasons for this decision are that this type of User Interface module does not need a lot of resources and there is a need for a low communication delay.

External User Interface module
This section deals with the deployment of the type of User Interface module that provides additional functionality. Again, the Request Handler is the only component of the RMS that uses this type of User Interface module. However, other WWICE entities might also use the functionality that this module provides. An example of this type of User Interface module is a User Interface module that is able to interact with a person using a telephone connection.

This type of User Interface module has the following properties:

- The Interaction module can function without this type of User Interface module. However, the actions that it can undertake depend upon the available User Interface modules. For example, if there is no User Interface module available to start interaction with an initiator using a telephone connection, the Interaction module cannot start interaction with an initiator who is trying to make an ordinary phone call with an ordinary phone.
- This type of User Interface module might use a lot of resources (e.g. CPU, memory, and storage). For example, a User Interface module that takes care of video interaction needs a lot of CPU and memory.
- There might be a high information flow between this type of User Interface module and an external entity. For example, a User Interface module that is able to start video communication with a person out-side the WWICE system, needs a lot of bandwidth between itself and the Gateway.
- When a user starts to interact with the UI, she expects to see a response from the RMS within a second. Therefore, there is a need for a low communication delay between the User Interface module and the Interaction module.

Based upon the properties of the Interaction module and the criteria for deployment, this type of User Interface module must have a distributed deployment. The main reason for this decision is that this type of module may use a lot of resources (e.g. CPU, memory), and it might have a high information flow between itself and an external entity (e.g. the Gateway). This graduation project does not deal with this type of User Interface module in more detail, since this type of User Interface module is not essential for the functioning of the RMS. As from this point forward, a 'User Interface module' refers to the first type of User Interface module (the User Interface module that creates UI descriptions and that sends them, via Cluster managers, to the UI managers of the WWICE system).

§ ***Cluster manager***

The Cluster manager determines the specific properties of a cluster (e.g. the available UI resources, and whether someone is using the cluster). Furthermore, it combines the UI descriptions of different User Interface modules in order to create one WWICE-specific UI description for a cluster[70]. The User Interface module is the only component that uses the Cluster manager. Therefore, this section explains the deployment of the Cluster manager with respect to the User Interface module. The Cluster manager has the following properties:

- The Cluster manager is very important to the User Interface module since the User Interface module cannot create a UI without a Cluster manager.
- The Cluster manager does not use a lot of resources (e.g. CPU, memory, and storage), since it only combines UI descriptions and it gathers some cluster-specific information (e.g. whether someone is using the cluster).
- Every time a user interacts with the UI (e.g. she selects a button), the Cluster manager informs the User Interface module. The User Interface module informs the Interaction module and receives a new interaction scheme. Based upon this scheme, it constructs a new UI description and sends it to the Cluster manager. The size of a UI description is between approximately 1 Kb and 20 Kb. Therefore, there is no high information flow between the Cluster manager and the User Interface module.
- There is need for a low communication delay between the Cluster manager and the User Interface module, since the user expects a response within a second.

Based upon the properties of the Cluster manager and the criteria for deployment, the Cluster manager must have a local deployment. The main reasons for the local deployment are that the Cluster manager does not use a lot of resources and there is a need for a low communication delay.

[70] The operation of the Cluster manager is analogue to the operation of the UI manager: both entities combine UI descriptions.

§ ***Status module***

All components of the RMS can use the Status module to obtain status information. The Request Handler uses the status information to determine whether a person is willing to start communication. Since this information is very important to deal with a communication request, this section explains the deployment of the Status module with respect to the Request Handler. The Status module has the following properties:

- The Status module provides information that is very important to the functioning of the Request Handler. Without the status of a person, the Request Handler cannot determine whether a person is willing to start communication[71].
- The Status module does not use a lot of resources (e.g. CPU, memory, and storage), since it only calculates the status of persons. This is a simple calculation. The Status module only computes the status when it receives an event that influences the status (e.g. a person has started a new activity).
- There is not a high information flow between the Request Handler and the Status module. The Request Handler subscribes for the status of a particular person (e.g. the status of the person 'Homer'). The Status module only notifies the Request Handler when it has determined a new status for the particular person.
- If the Request Handler deals with real-time communication, it must deal with the request as fast as possible. The Request Handler needs the status to deal with the communication request, thus there is need for a low communication delay between the Request Handler and the Status module.

Based upon the properties of the Status module and the criteria for deployment, the Status module must have a local deployment. The main reason for this decision is that the Status module does not use a lot of resources. External entities might need the information of the Status module. For example, MS ® Messenger might need status information. However, I think that calculating status information is a specific functionality of a RMS. The RMS might provide an interface that enables other applications to retrieve status information from the Status module.

§ ***Information module***

An Information module provides a service that extracts information from available data, e.g. a 'Extract subject from text'-service or a 'Extract identity from video stream'-service. The Request Handler is the only component of the RMS that uses Information modules. However, other WWICE entities might also use the functionality that an Information module provides. For example, the Information module that is able to extract the identity of a person from a video-stream, might be used by other applications than the RMS, e.g. by a burglar alarm system that uses video camera's to protect the house. This section explains the deployment of the Information module with respect to the Request Handler. An Information module has the following properties:

- A Request Handler can function without Information modules. However, the way it determines whether a person is willing to start communication depends to some extent upon the available Information modules. For example, if there is no Information module that is able to extract the subject from a text, the Request Handler cannot take the subject into account when it determines whether a person is willing to start communication.
- Some Information modules might need a lot of resources (e.g. CPU, memory, and storage). For example, an Information module that extracts the identity of a person from a video stream needs a lot of CPU and memory.
- There is not a high information flow between the Request Handler and an Information module. A Request Handler asks the Information module to extract a particular piece of information, e.g. the subject of a message, and the Information module returns that information. However, some Information modules might need a lot of bandwidth between itself and a particular device of the WWICE system. For example, an Information module that extracts the identity of a person from a video stream that originates from out-side the WWICE system, needs a lot of bandwidth between itself and the Gateway.

[71] If the Request Handler cannot retrieve the status, it uses the default status 'Available', such that the recipient does not miss important communication.

- If a Request Handler deals with real-time communication, there is a need for a low communication delay between the Request Handler and a Information module, since the Request Handler must deal with its' task as fast as possible.

Based upon the properties of the Information module and the criteria for deployment, the Information modules must have a distributed deployment with respect to a Request Handler. The main reasons for this decision are that Information modules might need a lot of resources (CPU, storage), and that external entities might use the functionality of the Information module.

§ ***Transformation module***

A Transformation module provides a service that can transform one type of communication into another, e.g. a Transform-video-to-videomail service. The Request Handler is the only component that uses the Transformation module. However, external entities might also use the functionality that this module provides. For example, the Transformation module that transforms mpeg-2 into mpeg-4 is merely a Transcoder-resource. A Graphmapper might use a Transcoder-resource to enhance a graph. This section explains the deployment of the Transformation module with respect to the Request Handler. A Transformation module has the following properties:

- A Request Handler can function without any Transformation modules. However, the actions that it can undertake, and the options that it can propose to an initiator, depend upon the available Transformation modules. For example, if there is no Transformation module available to record a voice-mail, the Request Handler cannot propose this option to an initiator.
- Some Transformation modules might use a lot of resources, e.g. a Transformation module that transforms a video stream into a video mail message.
- The Request Handler asks a Transformation module to take care of a particular task, and the Transformation module returns the result (e.g. a reference to a file that contains a video-mail). Therefore, there is not a high information flow between a Request Handler and a Transformation module. However, some Transformation modules might need a lot of bandwidth between itself and a particular device of the WWICE system. For example, a Transformation module that transforms a video stream that originates from out-side the WWICE system into a video mail message, needs a lot of bandwidth between itself and the Gateway.
- There is no need for a low communication delay between the Request Handler and the Transformation module. For example, a Transformation module might need to record a video mail message. The user of the RMS will understand that it might take while until the system is ready to start recording.

Based upon the properties of the Transformation modules and the criteria for deployment, the Transformation modules must have a distributed deployment. The main reasons are that Transformation modules might need a lot of resources (CPU), and that external entities might use the functionality of a Transformation module.

§ ***Learning module***

The Learning module is the component of the RMS that tries to extract new preferences. It might use information from any component of the RMS. Therefore, this section explains the deployment of the Learning module with respect to all other components of the RMS. The Learning module has the following properties:

- Components of the RMS can function without the Learning module. If the Learning module is not available, this only means that the RMS cannot learn from any user behavior at that moment.
- The Learning module keeps track of a lot of data. The database(s) that store these data can become quite large. Furthermore, the Learning module might use complex algorithms to extract new preferences (e.g. neural networks). Therefore, it might need a lot of CPU resources.
- There is not a high information flow between the Learning module and a component. A Learning module subscribes with a component for some information. For example, the Learning module might subscribe with the Request Handler for the identity of all

users that start communication, and the location where they start communication. A component notifies the Learning module when it has new information.
- The Learning module does not need to determine new preferences real-time. Therefore, there is no need for a low communication delay between a component and the Learning module.

Based upon the properties of the Learning module and the criteria for deployment, the Learning module must have a distributed deployment. The main reason for this decision is that the Learning module might need a lot of storage and CPU.

One instance of a Learning module per WWICE system is sufficient, since it is a waste of storage to store all the data that the Learning module gathers at multiple locations. Furthermore, multiple instances of a Learning module would require synchronization of the data. Since components can have information for the Learning module at any time, it runs continuously. The WWICE system starts the Learning module when the device where the Learning module is installed, is booting. This document does not deal with the Learning module in more detail, since the Learning module is not essential for the functioning of the RMS.

§ ***Preferences module***

The main functionality of the Preferences module is to enable the components of the RMS to store and retrieve preferences. It stores these preferences in the Preferences database. Several components of the RMS use preferences. Therefore, this section explains the deployment of the Preferences module with respect to all other components of the RMS. The Preferences module has the following properties:
- The preferences that the Preferences module provides, are of utmost importance to the functioning of the components of the RMS. For example, a Request Handler needs preferences to determine the reachability of a recipient[72].
- The Preferences module needs storage to store the preferences. The size of the preferences that a component needs to execute its' task, is approximately between 1 Kb and 20 Kb. Therefore, the Preferences module needs at most a few Mb to store all preferences.
- Only when a component needs to store or retrieves preferences, there is an information flow between that component and the Preferences module. Since the size of these preferences is approximately between 1 Kb and 20 Kb, there is not a high information flow between a component and the Preferences module.
- If the RMS must deal with a request for real-time communication, there is a need for a low communication delay between the Preferences module and the components that have to deal with this communication request (e.g. a Request Handler and a Interaction module). There is a need for a low communication delay since an initiator must receive a response from the RMS of the recipient as fast as possible. The components need preferences to determine the response.

If a component of the RMS is installed at only one device, and it does not need to share any preferences, it can store its' preferences locally. Such a component does not need a Preferences module (if there is enough storage on the device where it is installed). This goes for the Transformation modules, the Information modules, and the Learning module.

A user can install multiple RMS-es per WWICE system[73]. The components of these RMS-es need to use the same preferences. Therefore, the preferences must be stored at a central storage that is continuously available, such that all components can retrieve the latest preferences from that storage. User-specific preferences are stored at the Person services that are part of the WWICE system, since these services are able to store preferences and are running continuously. Therefore, these services are very suitable to store all user-specific preferences. However, not all preferences are user-specific. An example of a preference that applies to all users of the RMS, is the preference that determines the search order that a

[72] If a component cannot retrieve the preferences it needs, it uses default preferences.
[73] Section 6.5 explains the deployment of RMS-es in the WWICE system.

Request Handler uses to search for devices that it can use for a particular type of communication. The Learning module might optimize this search order over time. This kind of preference is stored at a Preferences database that has a local deployment with respect to the software component that is registered in the WWICE system as a ‘RMS‘.
Since the preferences are of utmost importance to the operation of the components of the RMS, the Preferences module must have a local deployment with respect to the ‘RMS’. This means that there is one Preferences module per RMS. Figure 56 shows the deployment of the Preferences module and the Preferences database(s).

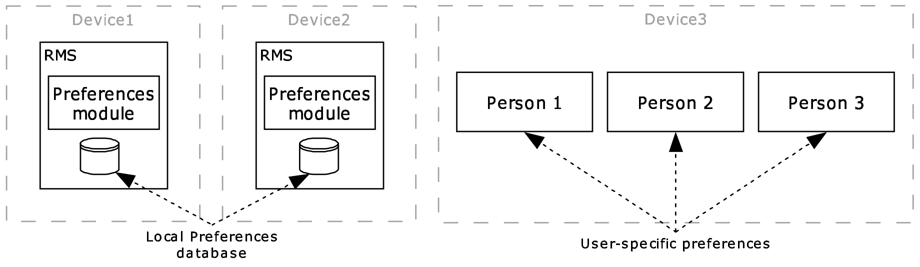


Figure 56 Preferences module: local deployment
Preferences database: local deployment + distributed deployment

At start-up, the RMS starts a Preferences module. This Preferences module is running as long as the RMS is running. The Preferences module takes care of the synchronization of the local Preferences databases. If a Preferences module must change one of the local preferences, it sends this update to the Preferences modules that are part of other RMS-es that are running at that moment. However, a Preferences module that is part of a RMS that is not running at that moment, will not receive this update. Therefore, a Preferences module asks for updates when it detects a new RMS (e.g. when a RMS is started). A problem occurs when two Preferences modules change the same preference while they cannot synchronize that preference, e.g. because of a network split between the two RMS-es. In this situation, these modules do not know which preference-update is the correct preference, when they detect each other. Therefore, one of these Preferences modules contacts the user to solve the problem (if necessary). The Preferences module takes care that two users do not change the same preference at the same time. If two users want to change the same preference at the same time, the Preferences module detects a conflict and notifies the users. Section 4.7 explains that the Preferences module retrieves the most important preferences at start-up, e.g. the preferences that components of the RMS need to deal with a communication request regarding real-time communication. Therefore, a Preferences module subscribes with the State monitor for changes in the user preferences. If a user preference changes (e.g. because the Learning module changes a preference), the State monitor notifies the Preferences module such that it can retrieve the new preference.