Active Messenger:
Email Filtering and Mobile Delivery

by

Stefan Johannes Walter Marti

M.S. Special Psychology, Philosophy, and Computer Science
University of Bern, Switzerland, 1993

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

Massachusetts Institute of Technology

September 1999

Signature of Author: ____
Program in Media Arts and Sciences
August 12, 1999

Certified by: ____
Christopher M. Schmandt
Principal Research Scientist, MIT Media Laboratory
Thesis Supervisor

Accepted by: ____
Stephen A. Benton
Chairman, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Active Messenger:
Email Filtering and Mobile Delivery

by

Stefan Johannes Walter Marti

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
on August 13th, 1999 in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

This thesis is about delivering electronic messages. Active Messenger, a software system, makes it easier and more efficient.

Today, there are a multitude of communication devices and channels available to exchange messages. However, each of these devices and channels has different capabilities. Additionally, the location of a user is often unknown. Current systems that decide where to send a message in a heterogeneous communication environment are complicated to customize and inefficient. The Active Messenger improves this situation.

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 user reactions. It infers the location of the user by looking at her communication history and communication behavior. If a message arrives in the user's inbox, the Active Messenger decides if it is important by looking at user-specified rules, as well as correlating it with recent messages, the user's calendar, and her address book. Depending on the importance of the message and the inferred location of the user, the Active Messenger decides where to send the message, possibly to several devices in turn, monitoring the reactions of the user 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 has read the message. If the primary communication device used provides no back-channel information, the Active Messenger infers whether the message has been read by monitoring other channels shortly thereafter. Depending on the status of the sender defined by the user, the Active Messenger may also give feedback to the sender about the user's location and communication behavior. The goal is to make the Active Messenger easily configurable. This thesis shows a first step towards an integration of the available communication channels.

Thesis Supervisor: Christopher M. Schmandt
Title: Principal Research Scientist

Thesis Committee

Thesis Supervisor: ____

Christopher M. Schmandt
Principal Research Scientist
MIT Media Laboratory

Thesis Reader: ____

Walter Bender
Senior Research Scientist
MIT Media Arts and Sciences

Thesis Reader: ____

Allen E. Milewski
Technology Consultant
AT&T Labs Research

Acknowledgements

You never do something alone in an academic environment, but I was overwhelmed by the support I got from the MIT community. Although I indeed expected to meet there the brightest people in the world, I didn't expect that they were so supportive and nice as well! I would like to thank a lot MIT and the Media Lab, but very special thanks go to:

- Chris Schmandt, my advisor during the last two years. He hired me and therefore gave me the opportunity to study at MIT—although I had to write two applications! But that's ok. He is heading a group that focuses on how people really use speech and communication technologies, and examines how these technologies can affect everyday life. That's cool! Now you just have to trust me and let me build the speech controlled micro helicopter, and we both will get very famous for that! :-)
- Members of the Speech Interface Group, Keith, Natalia, Nitin, and Sean. Keith was always up for discussing strange but sensational ideas like *ethical advisor systems* (I will get the Nobel prize for that, I promise!), playing racquetball, movies, sailing, rollerblading, biking, and food, of course. Always, always. He was even my roommate during the last year—I guess you understood and didn't mind that I spent the last few months with my Love. Nitin, my office-mate, provided useful advice and guidance as the only senior member of the group—although I am much older than you are, kid! ;-) With Natalia, I have talked about many things, especially about how to deal with certain people... And Sean, the most recent geek-boy, is a very open and energetic influence for our group. Your treasure hunt was a hit! All these people helped me with my project, especially in the design phase—and some are actually using my code!
- Allen Milewski, who served as a reader for my thesis. Perhaps because he is not from the Lab, he had some very interesting comments. Thanks a lot!
- Walter Bender, director of the News in the Future consortium and also a reader for my thesis. Walter made a lot of things easier, because Pascal is his student. And hey, are you aware that you brought me to the idea of my autonomous video helicopter?
- The sponsors of the MIT Media Lab: AT&T for making me a fellow, Motorola for a generous equipment grant, and particularly Sheila Griffin, Liz Altman, and Kim Goldinger.
- Pascal Chesnais and Joshua Randall. They let us use their pagers in the first place, thanks a lot! No, I don't mind that Canard had to be rebooted very often, and I know that it's not your bug...
- Gert-Jan Zwart †. He started out with me at the Lab, his chair about three meters from mine. Then, after only a few months, he had to leave. How far away are you right now? I don't know—can one measure this distance anyways? I am sure you would have an answer, you always had one. I miss you, and you make me think about to be or not to be. Hey, at the place where you are right now, do they have this antigravity stuff we were talking about? I guess yes.
- My parents: Helene and Johannes, my sister Christine, and my brother Lukas. I guess I was never able to express my thanks for their love and support throughout my whole life, but I really thank you for being there for me. Only when you are far away, you realize what you don't have anymore.
- Ralph Diemer, Jasmin Schmid, Markus Spielmann, and Sandra Gallauer, for being my best friends.
- And last but no least, Kimiko, my Love. What would I do without you? E ha di gärn!!

Stefan Marti, August 1999.

Preface

This is my second Master's thesis. I wrote the first one at the University of Bern, 1993, in the field of Special Psychology [18]. It was about the impact of telecommunication technologies on the user. Although many things have happened since then, I have realized just recently that both my theses were actually about the same issue: people using modern telecommunication technologies.

But my two theses are focusing on completely different perspectives of this problem. When my psychology thesis tried to explain why users tele-communicate the way they do, why they choose one medium over another, it still was just a description of the status quo. Now I am at the Media Lab, and more than trying to explain "the world," we rather shape it ourselves. This time, it is about how to combine existing communication infrastructure into something useful.

Why would I want to do that? The reason is that none of our communication technologies seem to be perfect. However, we have learned to live with them. Sometimes, we even like to have a choice between different communication channels, because certain channels are more appropriate than others for a given situation, message content, and sender-recipient relation. But sometimes, we would like to have just *one* channel, *one* device, which adapts to where we are, what we want, and how we want it. A good example for such a device is the Star Trek Next Generation communicator, also called Comm Badge:

Figure 1: Star Trek comm badge, ca. 2364

> Communicator: Personnel subspace communication device, originally hand-held, later in the 24th century integrated in the Starfleet badge, the latter is also referred to as comm badge. The communicator serves to establish a voice contact to another person or computer and provides lock-on contact for the transporter. The comm badge is usually programmed with a crewmember's individual bioelectric data, which is verified through a dermal sensor. The communicator will fail if used by an unauthorized person. [31]
>
> Communicators (...) are considered fast FTL radios. With support from a ship in orbit they have a range of about 50,000 km (...) Transporters can lock onto this signal in order to increase reliability of transport. Comm badges can adhere to almost any surface using a magnatomic adhesion area. They are powered by a rechargeable Saurium Krellide crystal that provides continuous usage for two weeks. [8]

But we are not there yet. Even more importantly, we may never have such a device, not only because it is technologically very challenging. However, in the mean time, we have to deal with poorly designed user interfaces, limited range, lacking content transcoding, etc. My thesis is only about text messages, but my system could also be used for other communication modes. The most important thing I have learned anyways is that we don't know well enough what the user actually wants. Or rather, we think we know it in theory, but in reality, everything is a bit different.

When I started coding my Active Messenger agent software, I thought I would finish it within a few weeks. Indeed, the basics were programmed within a week—but to fine-tune the agent, it took me several months! It was "only" about details, but these details made a big difference for the users. Because my goal was to make the life of the user easier, this fine tuning stage was getting more and more important. Therefore, I will describe the Active Messenger in one chapter, and the iterative design and user evaluation in another one.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

1.1 What is the problem?

Because the number of portable hand held communication devices like cellular phones and pagers will probably increase[1], the number of channels through which a person can be reached will also grow (see Figure 2: **A multitude of communication channels**). This leads to the problem that a person has several phone numbers and addresses that are difficult to keep track of. Therefore, users have started to give out only one address where they wish to receive all their messages[2]. However, there is no satisfying solution available for the problem of how the messages are forwarded to the several available devices. The most sophisticated systems today filter text messages and try to prioritize them by looking at the user's recent communication history, calendar, and address book entries. However, they are not effective in deciding where to send a message depending on its importance and on the available communication channels.

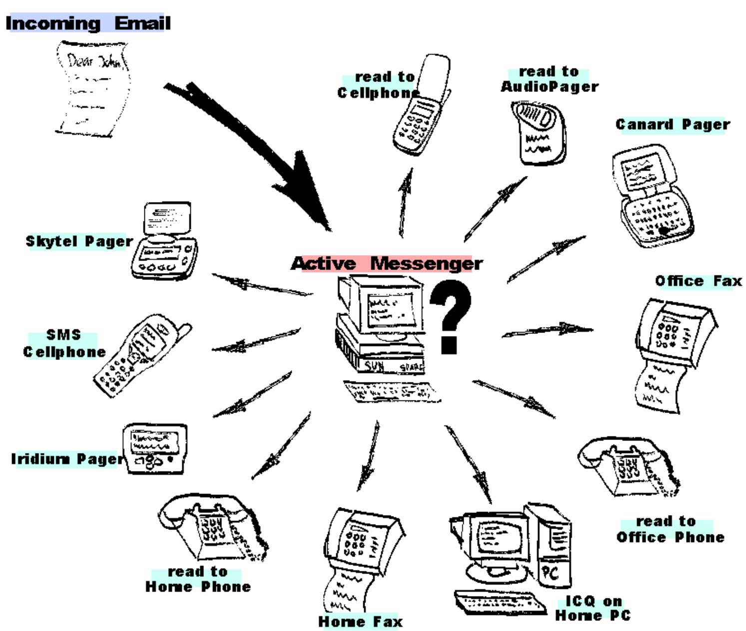


Figure 2: A multitude of communication channels

1 Most probably because they are getting cheaper, due to a miniaturization of the electronic components, and due to more and more wireless networks.

2 The basic idea is the *universal inbox*, "a single place where faxes, E-mail, and voice messages are collected and presented to the user. Apple™ first implemented such a capability in System 7 Pro, and Microsoft™ is building one into Chicago." [23]. See also http://www.cs.berkeley.edu/~bhaskar/univ-inbox/.

Active Messenger solves this problem. It is a software system that integrates mobile delivery systems like pagers, with conventional mail reading systems. The Active Messenger filters incoming text messages by using location specific filtering rules. It can forward messages to the available portable and stationary devices like text and voice pagers, cellular phones (alphanumeric and text-to-voice), etc. The agent does not simply send a message to all channels at the same time, but sends it to the most appropriate device, and then waits for possible user reactions. If the user does not read the message, the agent uses the next channel, and waits again. Therefore, forwarding messages includes not only routing a message to the appropriate channels, but also using several channels sequentially over time, avoiding redundant messages and reducing information overload. This makes the forwarding more a process over time than just a static routing. If necessary, email messages are transformed to fax messages or read to the user over the phone. The Active Messenger is aware of which devices are available for each subscriber, which devices were used recently, and whether the message was received and read by the user, or not. Because most of the currently available communications channels do not provide such information directly, the Active Messenger exploits back-channel information and infers from the users communication behavior over time if a message was read, or if a channel is active. Depending on the status of the sender, it is able to give information back to the sender about where the user may be and which channels she used recently.

In summary, the Active Messenger

- **Integrates mobile delivery devices like pagers, with conventional email reading systems.**
- **Makes filtering a process rather than a routing problem.**
- **Associates filtering rules to user locations.**
- **Converts email messages to faxes and speech.**

1.2 Other approaches and related work

1.2.1 Procmail

Active Messenger replaces an email filtering and forwarding system that is usually based on the shareware program *Procmail* (e.g., [32]). With Procmail, the user can specify rules that define which email messages are forwarded to another account, stored in a file, or trigger the execution of a program. A simple example for such a file is shown in Figure 3: **Sample .procmailrc file**.

```
PATH=/bin:/usr/bin:/usr/local/bin
MAILDIR=$HOME/Mail      #you'd better make sure it exists
DEFAULT=$MAILDIR/mbox   #completely optional
LOGFILE=$MAILDIR/from   #recommended

:0:
* ^From.*berg
from_me

:0
* ^Subject:.*Flame
/dev/null
```

Figure 3: Sample .procmailrc file

Procmail uses static rules and therefore does not filter dynamically. The rules are similar to the Boolean logic rules used by, e.g., Eudora [7]. Furthermore, Eudora as well as Procmail rules are hard-coded to devices. To make the email forwarding more flexible, some users have several sets of rule files and switch between them by sending email commands, or commands over the phone. However, this "semi-dynamic" filtering is not satisfying because users may forget to switch the rules back. Active Messenger solves this problem because its filtering is a process, dynamic, location dependent, and takes in account the status of the user's communication infrastructure.

1.2.2 Clues

Clues is a filtering and prioritization system developed by Matt Marx at the Media Lab [19]. It detects the timely nature of messages by finding correlation between a user's calendar, rolodex, to-do list, and record of outgoing messages and phone calls. It is used to prioritize email messages, but it could also handle voice messages based on caller identification, as well as fax messages based on the fax header information.

Clues is a useful extension for a Procmail based email forwarding system, but it has neither awareness of which communication channels are available, nor the possibility to take several steps over time. Active Messenger compensates for that and allows the user to take advantage of Clues by specifying messages that are timely. First, the user has to run a Cron job[3] to generate new rules regularly. Once the rules are created and stored, the Clues filter program can give back the importance level of a message. This importance is expressed in the form of a category. The user specifies part of the possible categories manually with static procmail rules. Other categories, like "timely," are generated dynamically by Clues. It infers message timeliness by considering calendar appointments, outgoing messages and phone calls, and by correlating these "clues" via a personal rolodex.

User studies have shown [19] that Clues is especially useful for subscribers with high message traffic who often access their messages on mobile devices or on the phone. Nevertheless, the user still has to decide what to do with a message of a certain category. Active Messenger makes this process automatic on behalf of the user by modifying the filtering rules according to the importance of the message.

1.2.3 Canard community messaging

Canard [5][6] is a Media Lab project that uses two-way pagers, a touch-tone based telephone interface with synthesized speech, a WWW interface, and electronic whiteboards. These communications devices and protocols are converted into a uniform message representation. Using personal databases, the relevancy of a message is evaluated and appropriate delivery channels selected (see Figure 4: **Canard overview**). The user only needs to know the person she is communicating with, and not the method of transport. By using filters similar to Procmail, Canard selects the most economic channel for the message as a function of the sender/recipient relationship and message urgency.

Although Canard attempts to solve the problem of different communication channels, it does not have the ability to wait and assess the success of message delivery, or the ability to try different channels in turn.

In his most recent work [6], Chesnais focused on the social implications of community-centered messaging like Canard. He proposed a framework that "must account for the varying skills of the users, the amount of support the network provider is willing to invest, and the effort needed to use tools." Living in a heterogeneous communication environment with many channels available requires specific knowledge and additional external support to manage and use these communication channels efficiently (P. R. Chesnais, personal communication, December 21, 1998).

There have been several attempts to solve the routing problem of multiple communication devices. In the following sections, I will describe some of them: Alter Egos™, PersonaLink™, OneMail™, iPulse™, OnTheMove, and The Mobile People Architecture.

---

[3] Cron is a Unix command for scheduling jobs to be executed sometime in the future. A cron is normally used to schedule a job that is executed periodically, e.g., to send out a notice every morning. It is also a daemon process, meaning that it runs continuously, waiting for specific events to occur.

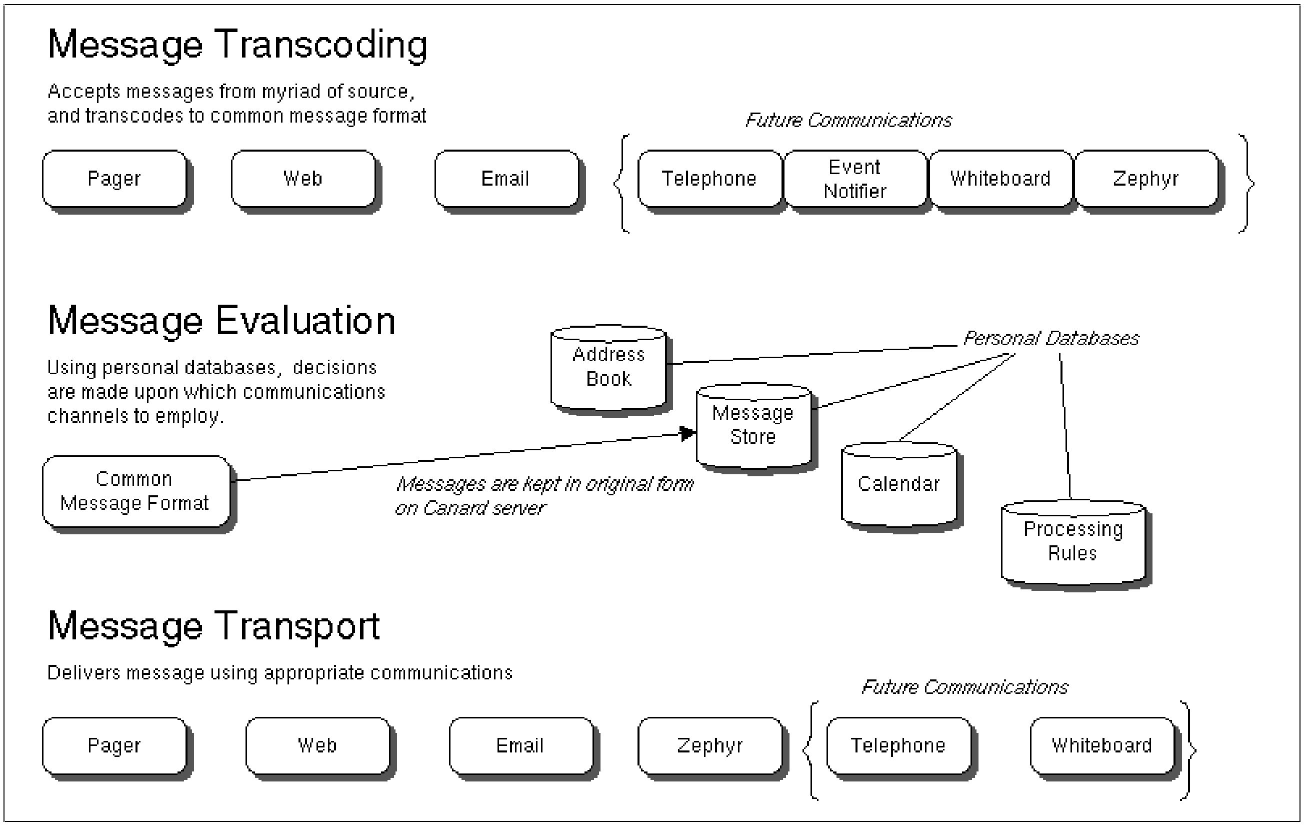


Figure 4: Canard overview

1.2.4 Alter Egos by IBM

IBM's Intelligent Communications Service end-user agents, called *Alter Egos*, keep track of user preferences and their location as well as the interfaces and protocols that are needed [9]. E.g., when a voice message arrives while the subscriber is travelling abroad, *Alter Egos* translate it into text and deliver it via email to the user's mailbox, so she can retrieve it on her laptop computer. On the other hand, *Alter Egos* may "speak" email messages directly into a voice mailbox, which the user can listen to from a pay phone. The agents have more than the ability to track down a user in order to send a barrage of messages. They deliver important messages, sorting them by what they know about the user as well as by what the user's explicit instructions are. The user may specify, for example, that all calls from family get top priority.

The idea behind *Alter Egos* is interesting. A typical scenario in IBM's Intelligent Communications network would involve "one user querying another user's *Alter Egos* to find out how to best route a message of a given type to the recipient at a specific moment. The response provided by the *Alter Egos* may be a user pager number or an email address. Subscriber proxies insulate people who are sending messages from having to know the intimate details of a recipient's routing path. Proxies also insulate users from details regarding phones, land-line modems, and other devices." [24] Knothole, which will be described in chapter 1.3, is based on similar ideas.

Today, there is no commercial system available that does what IBM has promised three years ago. It seems as if IBM has lost its interest in this specific domain. IBM's newest product, the MemoryAgent™, is described as "a smart assistant that learns what people like, and anticipates what they may need; it also learns complex processes and can make improvements" [11]. Although it may be able to do all the above-described tasks, a lot of customization may be required first.

1.2.5 PersonaLink™ and OneMail™ by AT&T

Another, somewhat comparable effort was AT&T's proprietary PersonaLink™. It was based on General Magic's Telescript™ technology. Telescript™ proposed that agents are active entities travelling across the network. PersonaLink™ was an electronic community for email, information retrieval, and online shopping. The network consisted of centralized servers, accessed by phone or wirelessly through the Ardis™ packet radio network. A problem was that PersonaLink™ required Telescript™-enabled devices and "telescripted" software. There were only two devices available, the Motorola Envoy™, and the Sony Magic Link™. Both were PDAs using General Magic's Magic Cap™ operating system, which had built in Telescript™. The system was introduced in September 1994 and shut down two years later, as "a continuation of AT&T's strategy to shift from proprietary network-based services to the Internet" [3].

There are other products and manufacturers that address the problem of delivering messages in a heterogeneous environment.

AT&T's OneMail™ provides the user with an email mailbox that will give her access through the telephone or web browser to email and voice phonemail in one place on the Internet.

Portico™ is a second-generation virtual assistant service. Using an intelligent voice interface called magicTalk™, Portico™ lets users access their email, voice mail, calendar, address book, news, and stock quote information via any telephone or standard web browser.

However, both systems rely on rules that have to be specified by the user manually.

1.2.6 iPulse™ by Ericsson and Oz.com™

A more recent effort is iPulse™[4] by Ericsson and Oz.com™. This system mediates between two subscribers by finding a way to get a text message or audio stream through, according to the preferences of the receiver. The manufacturer says that it can instantly and easily connect users to each other by computer, phone, pager or mobile phone through a simple point-and-click contact menu (see Figure 5: **iPulse™ screen shot**). It also allows users to customize their communications by setting up individual profiles that indicate when, by whom and how they want to be reached. It is supposed to alleviate the contacting person from the burden of finding the right channel. It supports paging, voice chat, text chat, web conference (text chat with the control of a common web browser), and IP telephony. The user also can keep a contact list and set her online status very similar to ICQ™ [12] (see also chapter 2.3.6).

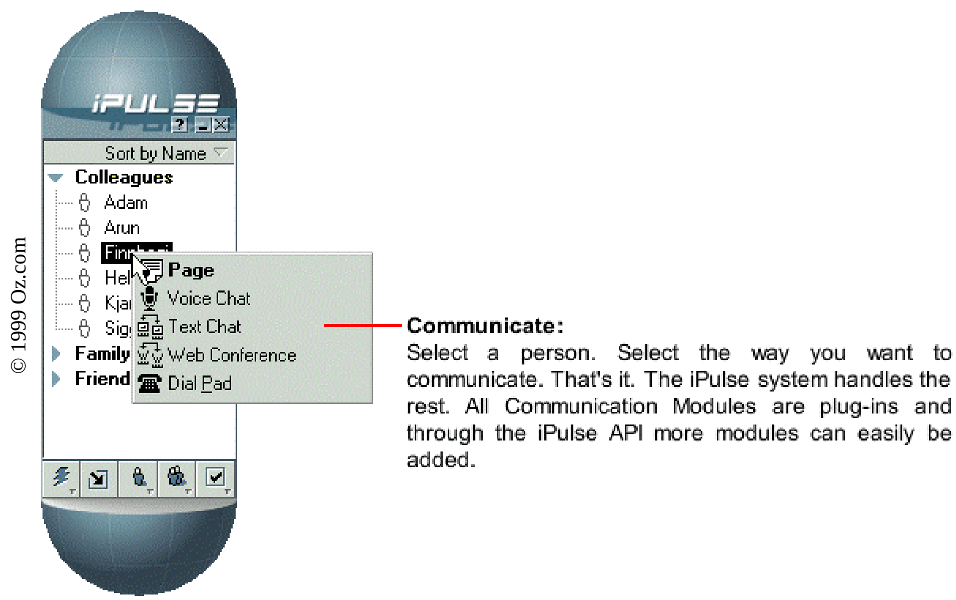


Figure 5: iPulse™ screen shot

The iPulse™ framework consists of a client application and a back-end server system. The main function of the framework is to provide users with a simple and secure way of establishing communication sessions with other users or services, running either on IP or other networks like PSTN. Basically, iPulse™ acts as a mediator. It mediates communication services between two or more people and regulates access to value added services.

[4] http://www.oz.com/ipulse/.

Active Messenger addresses the same problems as iPulse™. However, iPulse™ is a proprietary system that is directed towards service providers. Although the manufacturer writes that new services can be implemented easily, legacy services such as stand-alone paging systems or fax may not be integrated.

1.2.7 OnTheMove

OnTheMove [21], a three-year project that ended 1997, was initiated by the European Commission for the ACTS program[5]. It focused on how to deliver multimedia content to mobile devices. The initial incentive for the project may have been that the demand for access to multimedia content with mobile devices is likely to increase with the launch of Third Generation Mobile Systems such as the *Universal Mobile Telecommunications System* (UMTS) around 2002 (e.g., [15]).

Beside having done a lot of conceptual work, the researchers have implemented a middleware prototype called *Mobile Application Support Environment* (MASE) that is located between the wireless networks, e.g., GSM, DECT, UMTS, and the applications, e.g., video conferencing, personal newspaper, etc. MASE stores user preferences, detects the location of the user, and adapts to the status of the wireless networks and the available bandwidth. The developers say that it "hides the complexity of networks from applications, making different wireless networks appear as a seamless and homogeneous communication medium. **Multimedia conversion** allows content to be delivered to a mobile device in a format that is appropriate to its capabilities and also the characteristics of the network in use. The **location manager** provides a means of determining geographical position. Automatic **bearer switching** between GSM, DECT, and wireless LAN enables the use of multiple bearer networks. A **session manager** provides resilience to unplanned disconnection and a **replica manager** shows how file synchronization can be achieved. (...) Personalization is achieved through the use of profiles. (...) The MASE was implemented as a distributed system with functionality located both on mobile devices (Windows 95 and Windows CE) and on a mobility gateway (Windows NT4) connected to both fixed and mobile networks. The project fully embraced the WWW technology which emerged at the start of the project and much of the MASE was developed using Java." [10]

Although this project addresses the same problems as Active Messenger, no information is available about if or how MASE filters and prioritizes communication. It is also not clear how the user specifies her preferences for certain channels depending on the importance of an event. Active Messenger fits well in the general framework set by the *OnTheMove* project.

---

[5] Ericsson, IBM, Siemens, Sony, British Telecom, Deutsche Telecom, and others also sponsored the project.

1.2.8 The Mobile People Architecture

The *Mobile People Architecture* (MPA) [2][30] is an advanced and promising framework for connecting people instead of their devices. The design is strongly related to the ideas behind Active Messenger.

The researchers focus on *routing between people*. The key challenge today is to find people and communicate with them personally, as opposed to communicating only with their possibly inaccessible machines like cellular phones and pagers that are turned off. They define the **Personal Proxy** (see Figure 6: **The Mobile People Architecture personal proxy design**), which has a dual role. As a **Tracking Agent**, the proxy maintains the list of devices or applications through which a person is currently accessible. As a **Dispatcher**, the proxy directs communications and uses **Application Drivers** to convert the message into a format that the recipient can see immediately.

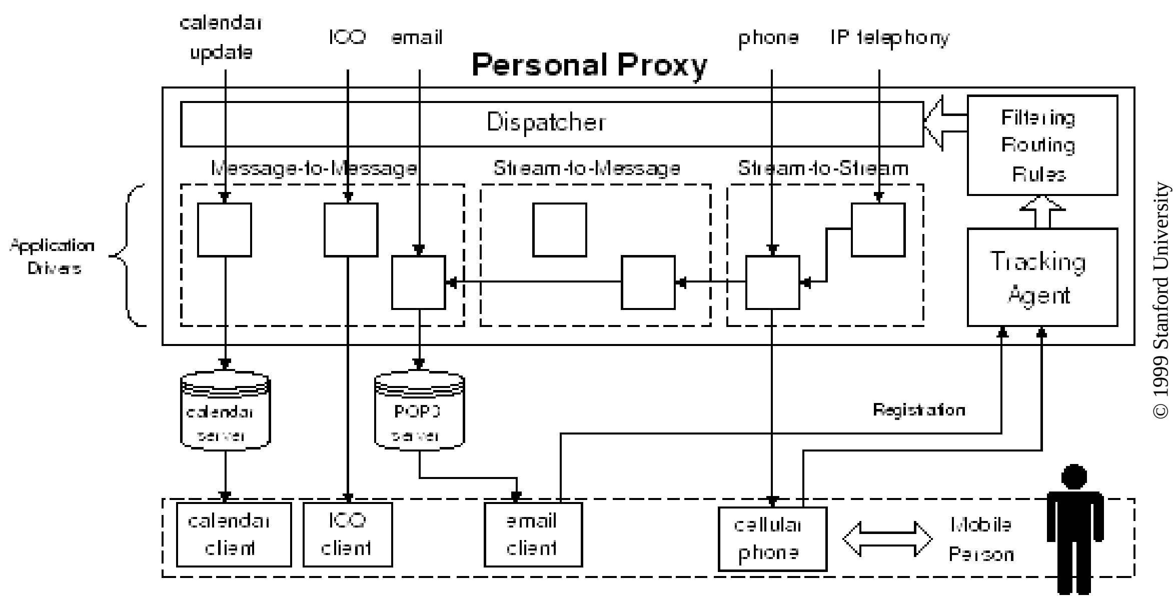


Figure 6: The Mobile People Architecture personal proxy design

Because no one wants to be receiving messages constantly, an important function of the Personal Proxy is to protect the user's privacy by blocking unwanted messages and hiding the true location of the user.

Unlike other approaches, the MPA is not a proprietary system and supports legacy applications.

The framework of the MPA is more general than the one of Active Messenger, because it includes also Stream-to-Message conversion. For example, if the user receives a phone call and is currently reachable through email only, the Personal Proxy converts the voice mail to an email and sends it to the user's computer. Currently, this feature is not implemented completely: the voice is not transcribed to text with voice recognition. However, the researchers describe in their most recent paper that they look at the pitch of an incoming voice message to decide where the message has to be routed.

A significant difference between Active Messenger and the Mobile People Architecture is that the MPA does not take several steps over time to guarantee the delivery of a message, trying multiple channels and awaiting possible user reactions.

Although the MPA framework is promising, the main disadvantage is that it isn't implemented yet completely. Currently, the prototype interoperates with telephony, email, and ICQ™. It will be very interesting to see a system that has all the announced features.

1.3 Predecessor system Knothole

Through creating the Knothole system, the author learned how to process email messages and how to forward and route them to different pager systems. Several features of Active Messenger are derived directly from Knothole modules.

1.3.1 Overview

Knothole is a system that uses two-way pagers to provide a bi-directional link to subscriber's normal email, with additional access to personal information on the subscriber's computer and external information via the Web and other sources. It's a combination of public domain programs, PERL scripts, various other scripts, some C programs, and a number of local Media Lab PIMs.

Knothole is relevant to Active Messenger because it enhances and encourages two-way paging services. E.g., all messages to and from a pager are routed through the desktop computer. Therefore, by scanning the incoming messages, the Active Messenger is able to draw conclusions about the location of the user, the status of the messages, and the status of the communication channels.

The main characteristics of Knothole are:

- **Information services.** The user can access a variety of information services, like weather and traffic reports, stock quotes, definitions, news articles, etc. The system also encourages email sending, which generates location information for other services like Active Messenger.

- **Dynamic email filtering.** For prioritization of incoming email, Clues [19] takes in account the user's calendar, rolodex, and communication history.
- **Proxy: the user's desktop computer as the only passages from and to the mobile devices.** Because all email passes through it, it can keep track of the user's activities.
- **Unified email appearance.** All email from the user has the same address, whether it was sent from a pager, cellular phone, or the desktop.
- **Hiding the two-way pager address.** No one else but the user's agent should decide which email messages have to be forwarded to the mobile devices.

Knothole enhances two-way paging services in the following three areas: email handling, mail control, and information access.

1.3.2 Email handling

A subscriber's email is filtered for selection of which messages get sent to the pager, and then those messages are processed before transmission. Procmail [32] does the first level of filtering. It allows messages to be indicated by matching regular expressions based on lines in the header or the message body. Rules are stored in the .procmailrc file (see also Figure 3: **Sample .procmailrc file**). The normal Unix delivery mechanism gets the message into procmail via the subscriber's .forward file.

Once Procmail has selected a message to be forwarded, its text is processed to fit into a reasonable sized page. The header lines and "included" text parts are removed, and the message is assigned a sequence number. The message is sent as email to the pager, and a copy is saved in a special directory, indexed by the sequence number.

With this forwarding mechanism, messages sent to the pager appear to come from the subscriber. This allows the user to reply so that the original sender does not see the pager address. The subscriber also sets up her .procmailrc file to detect messages from the pager and run them into a large PERL script for processing.

The most generic message is a reply, generated using the pager's "reply" mechanism. Back on the subscriber's computer (or rather, that computer which receives the message), the sequence number is detected in the pager's reply. Furthermore, the subscriber's message is re-packaged with a return address that is the subscriber's normal email address, a "Re: " is added to the subject line, and the original message is attached.

Some paging systems allow only short outgoing messages. Therefore, if the last character of a reply message is "+", the page contents are appended to any outgoing message in progress, and text is accumulated until a reply without a "+" termination arrives. This allows longer messages to be written, e.g., from a Tango™ device [28] or from a SMS cellular phone that can send only 160 characters per message.

1.3.3 Mail control

Any reply message can also be a command addressed to the system, not to the original sender. These messages are characterized by a keyword, the first word of a message.

E.g., **M <name> <text>** sends the <text> as email to <name>. <name> is resolved by first checking the user's normal mail aliases that are stored in the.mailrc file. If not found there, the name is looked up in the user's rolodex and translated to an email address. This feature depends on the Media Lab rolodex and other utilities. These messages can also be terminated with "+" to accumulate a longer message. Other mail control features include a summary. The user can request a summary of the last *n* messages, which can include sender, arrival date, and message number. Additionally, the user can make Knothole resend one or several messages**.**

1.3.4 Information access

Several functions can be triggered by keywords. The following is a list of the current functions.

- **Weather:** The user can request weather reports for a specific city. The forecast text is heavily filtered and abbreviated before being sent to the pager. The information is equivalent to web pages of, e.g., The Weather Underground[6]. In addition to standard reports, Knothole can provide short weather forecasts, extended weather forecasts, the current weather conditions, and even a list of all cities within a state. Furthermore, the system can give back sunrise and sunset times for a specific area.
- **Personal calendar:** The user can look up information in her UNIX rolodex file. She can send in a name and gets back various fields as a page, including home and work addresses and telephone numbers.

[6] http://www.wunderground.com/

- **Traffic:** Knothole can look up the current traffic conditions for several highways and thruways in Boston. This feature depends on access to the SmarTraveler[7] traffic conditions web site for Boston, and is also filtered from the HTML tags.
- **Dictionary:** The user can ask for the definition, or the etymology and pronunciation of any word. This information is taken from an online Merriam-Webster dictionary[8].
- **Financial:** STOCKS returns current values and percent change of the DJIA, S&P 500, and NASDAQ composite. This feature depends on access to the Bloomberg[9] world equity index page.
- **Operating system:** There is an extension mechanism for the UNIX operating system. Pages beginning with "+" or "-" are allow any executable or script to be accessed. A "+" causes the command to be executed with the arguments given, and the output is sent back as a page. A "-" opens a pipe and feeds the content of the page in there. The results are sent back to the pager as well.
- **News:** The command NEWS plus argument gives back the news story with the argument in the headline. The news stories are retrieved from the MIT Media Lab Fishwrap project. Similarly, HEAD plus argument gives back a list of headlines containing the word in the argument, and the user can request specific stories by using the multiple-choice reply feature of the PageWriter2000™.
- **Personal Calendar:** The user can add calendar entries to her UNIX calendar file, as well as request the entries of any day or week.

1.3.5 Summary

Knothole is a single PERL script, but each of the information functions depends on calling PERL scripts and executables of other authors to perform the requested task. For example, the STOCKS function causes the main PERL script to call a *csh* script that fetches a URL using text mode via a call to the CERN http library and then filters the output using *grep* and *awk*.

Knothole enhances the possibilities of a two-way pager if the messages from and to the pager are routed through the user's desktop and filtered for keywords. Many features of Knothole are vital for Active Messenger. E.g., Active Messenger reads an activity log file that Knothole writes. However, Knothole does not take in account the possibilities of several two-way paging systems and other communication channels working in parallel. This shortcoming is one of the reasons why Active Messenger was built.

---

7 http://www.smartraveler.com/scripts/bosmap.asp?city=bos&cityname=Boston
8 http://www.m-w.com/dictionary.htm.
9 http://www.bloomberg.com/markets/wei.html.

1.4 Overview of Active Messenger

The primary design goal for the Active Messenger is to forward incoming text messages to the 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 user and the success of the delivery.

When a message arrives in the user's inbox, the Active Messenger decides if the message is important by looking at the user's recent communication history, user-specified rules, and other resources. Depending on the importance of the message and the inferred location of the user, the Active Messenger decides where to send the message. 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 user reactions. It infers the location of the user by looking at her communication history and communication behavior. 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 has read the message. If the primary communication device used provides no back-channel information, the Active Messenger tries to infer whether the message has been read by monitoring other channels shortly thereafter. Depending on the user-defined status of a sender, the Active Messenger may also give feedback to the sender of a message about the location and communication behavior of the user.

Active Messenger is a software agent that acts autonomously on behalf of the user. Because the agent is autonomous, no direct interaction between the user and her agent is required. However, the user has to specify initially how to treat incoming email messages and what her infrastructure is. This includes telephone numbers, email addresses, and the names of office and home computers. This information is stored in the user preference file, a simple ASCII text file (see Figure 7: **Sample user preference file**).

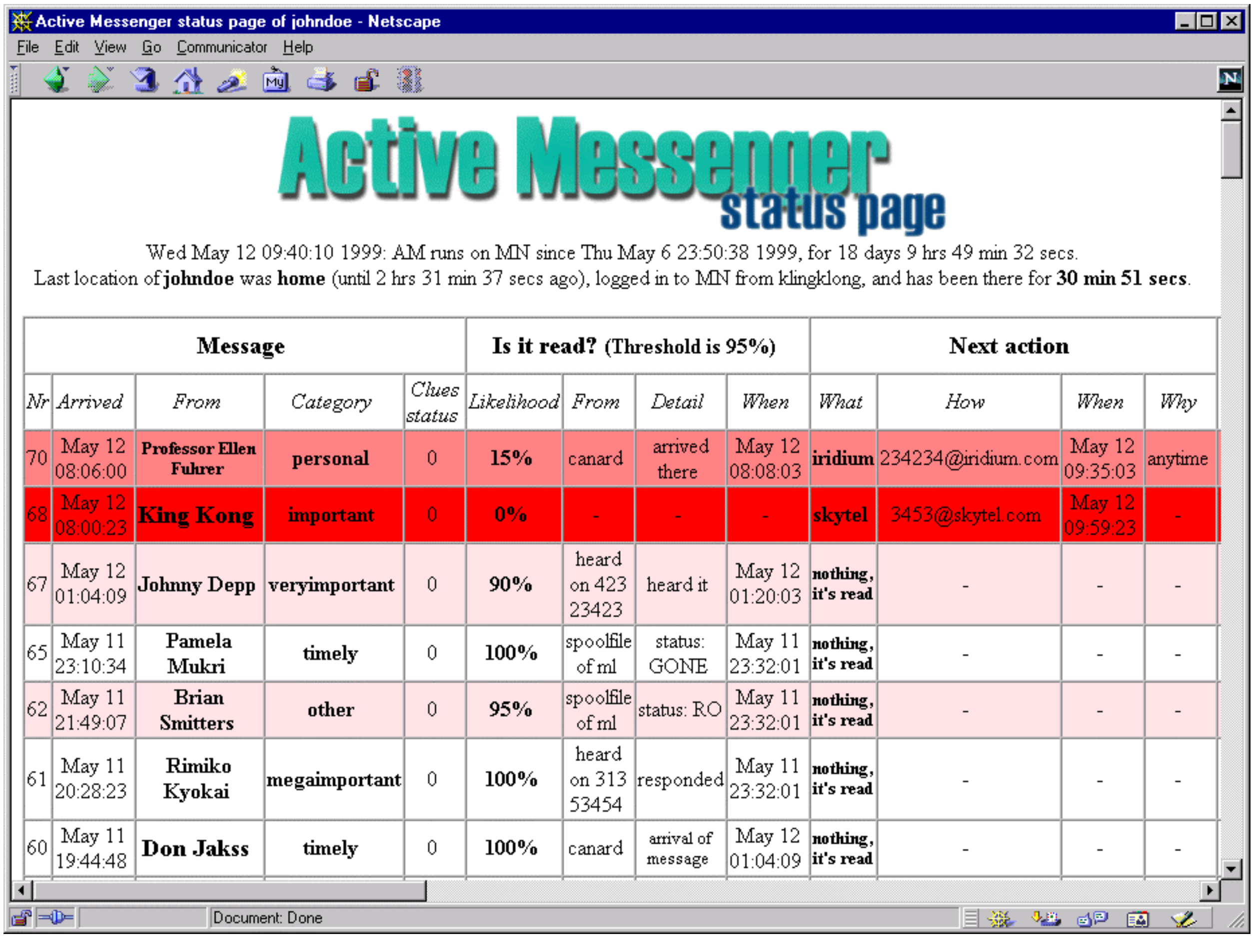
```
-------------
Mapping

personal       = canard
timely         = canard, skytel
important      = canard, skytel, fax
veryimportant  = canard, skytel, sms, iridium
ultraimportant = canard, sms, phone, cellphone, iridium
--------------
home

phone          = 568-5031, not 22-9
fax            = 396-8499 (no cover), not 23-8
computer       = dialup
--------------
office

phone          = 553-5386, anytime
fax            = 495-5244, M-F 10-23
computer       = magama, klingklong
--------------
Devices

canard         = johndoe@canard.media.mit.edu
skytel         = 4963278@skytel.com
sms            = 16178183421@omnipoint.net
iridium        = 881536082914@iridium.com
```

Figure 7: Sample user preference file

This file is the only information the agent needs from the user.

On the other hand, the user may want to know what the current status of the system is and what happened in the past. For this purpose, the Active Messenger writes its status to a web accessible HTML file (see Figure 8: **Sample status monitor page**) and to a log file.

The current version of Active Messenger is implemented as a single PERL script. The main data structure, the message list, holds all necessary information in program memory. For each message, several parameters are stored, e.g., who sent the message, when did it arrived, how was it read (from which channel or device), what and when will be the next action to take, what actions were already taken for this message. Other data structures hold additional information, e.g., a list of all errors that have occurred, all user preferences, and other device-specific information.



Message					**Is it read?** (Threshold is 95%)				**Next action**			
Nr	*Arrived*	*From*	*Category*	*Clues status*	*Likelihood*	*From*	*Detail*	*When*	*What*	*How*	*When*	*Why*
70	May 12 08:06:00	**Professor Ellen Fuhrer**	**personal**	0	**15%**	canard	arrived there	May 12 08:08:03	**iridium**	234234@iridium.com	May 12 09:35:03	anytime
68	May 12 08:00:23	**King Kong**	**important**	0	**0%**	-	-	-	**skytel**	3453@skytel.com	May 12 09:59:23	-
67	May 12 01:04:09	**Johnny Depp**	**veryimportant**	0	**90%**	heard on 423 23423	heard it	May 12 01:20:03	**nothing, it's read**	-	-	-
65	May 11 23:10:34	**Pamela Mukri**	**timely**	0	**100%**	spoolfile of ml	status: GONE	May 11 23:32:01	**nothing, it's read**	-	-	-
62	May 11 21:49:07	**Brian Smitters**	**other**	0	**95%**	spoolfile of ml	status: RO	May 11 23:32:01	**nothing, it's read**	-	-	-
61	May 11 20:28:23	**Rimiko Kyokai**	**megaimportant**	0	**100%**	heard on 313 53454	responded	May 11 23:32:01	**nothing, it's read**	-	-	-
60	May 11 19:44:48	**Don Jakss**	**timely**	0	**100%**	canard	arrival of message	May 12 01:04:09	**nothing, it's read**	-	-	-

Figure 8: Sample status monitor page

All incoming messages are stored in a separate directory by a program that is called by the user's Procmail. 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, where the following modules are called sequentially:

- **Load messages** (chapter 2.6.1)**.** If new messages have arrived, load them into the message list and schedule the initial event, e.g., where and when to send them.
- **Find user location** (chapter 2.6.2)**.** Where may the user be?
- **Check message read status** (chapter 2.6.3)**.** Go through all messages: What is the likelihood that the message is read? Checking the mail spool file, Canard, SkyTel™, and other resources.
- **Check status of paging systems** (chapter 2.6.4)**.** Did the Canard or SkyTel™ handsets come back in range? Is it necessary to resend older messages?

- **Send messages** (chapter 2.6.6)**.** Go through all messages: If a scheduled event is due, send the message to the specified device.
- **Schedule next events** (chapter 2.6.5)**.** After sending a message, schedule the next event.
- **Write web file** (chapter 2.6.7)**.** Store the current status in a file that is accessible on the web.
- If user preference file has changed, **reload user preferences**. Like that, changes can be made to the behavior of the agent without necessarily restarting the program. By editing the user preference file, the user can change most of the internal variables.

The following chapter describes the system: first, the background and basic concept of Active Messenger (chapter 2.1), then the current implementation. For that purpose, the devices and communication channels are described that the agent can interact with (chapter 2.3). What follows is a description of the architecture of Active Messenger: the program structure (chapter 2.4), the data structure (chapter 2.5), the main modules (chapter 2.6), and finally the user interface (chapter 2.7).

2. Description of the system

2.1 Background and concept of Active Messenger

The Active Messenger focuses explicitly on the specific communication needs of the members of the MIT Media Laboratory Speech Interface Group. They have a multitude of communication channels and devices available. First, there are networked PC's and workstations in the offices, and all users have PC's with dial-up or even permanent network connections at home. Email can be delivered to alphanumeric pagers with different ranges (*Canard* [5][6] is MIT Campus wide, *SkyTel*™ [27] USA nation wide in major metropolitan areas, and *Iridium*™ [13] worldwide), as well as to text-capable cellular telephones (Short Messaging Service based on the GSM standard). Email can also be transformed to fax messages and sent to the most likely location of the user, be it at home or in the office. Messages can be sent to *ICQ*™ by Mirabilis [12] or the *Instant Messenger*™ by AOL [1]. Both are popular commercial instant or even real-time messaging channels, based on the "buddy list" idea.[10] Furthermore, the Active Messenger can call up wired or cellular phones and read the email message using a text-to-speech module. Over the phone, it is not restricted to synchronous telephone connections, but can also leave messages on answering machines and voice mail systems. Additionally, a user can read email and listen to voice mail from Phoneshell [26][11], the Speech Group's phone interface to email and voice mail.

Upon an incoming email message, the Active Messenger has to solve the following four primary problems:

1. Which communication channels are active?
2. Determining the importance of a message. Is it important enough to be forwarded to other channels?
3. If the message has to be forwarded, which is the appropriate primary communication channel?
4. Monitoring the progress of the message delivery, and, if necessary, switching to alternative channels.

---

[10] "ICQ™ ('I Seek You') is a program that lets you find your friends and associates online in real time. You can create a Contact List containing only people you want to have there, you can send them messages, chat with them, send files, configure ICQ to work with external applications and more." [12]

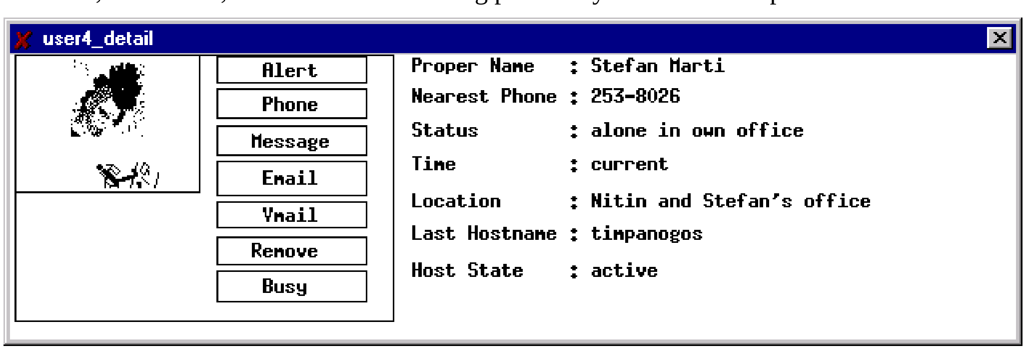
[11] Phoneshell is a telephone-based application providing remote voice access to personal desktop databases such as voice mail, email, calendar, and rolodex. Several forms of information can also be faxed on demand. Phoneshell offers its users numerous opportunities to record voice entries into its underlying databases. This utility for stored voice as a data type necessitates multimedia support for the traditional graphical user interfaces to these same databases.

Problem 1 is non-trivial, because most channels don't provide information about their status and the status of sent messages (see Table 1: **Characteristics of some communication channels**). However, looking at the usage of the devices (log files) can provide hints to the availability of devices. E.g., if a user sends a message back from a pager, one can assume that another message sent to this device shortly before was transmitted successfully and read, because sending a message implies that the device is online, messages can be received, and the user is probably using her device.

Problem 2 is solved by using the prioritization system *Clues* [19]. In addition, Active Messenger looks at log files of received phone calls (caller ID), sent and received pager messages, sent and received cellphone text messages (SMS), and Activity Server [16] information that is based on *finger*[12] and is used also for *Locate*[13] and *Watcher*[14].

Problems 3 and 4 are recursive. Depending on the importance of a message, different strategies are selected by the system. In general, the device most likely to be available to the user is selected. E.g., if the Activity Server reports that a person is logged in to a terminal and typing, the Active Messenger does not send the message anywhere immediately, because the user will probably read it from the standard UNIX mail spool file. However, because it is possible that the user has just left the terminal, the Active Messenger waits to determine if the message was indeed read and may take more steps. 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. In this case, the user who has previously agreed on that she can receive such important messages from a specific sender, has to acknowledge actively the message by sending an email reply or calling the system back, e.g., via Phoneshell [26]. The "obtrusiveness" of the Active Messenger's behavior depends highly on the previously determined importance of a message.

[12] *Finger* is a common UNIX server that typically gives back the last login time at a specific workstation.
[13] *Locate* is a command line program similar to *finger*, but gives back not only the last login time, but also the machine. The answers are plain text like "John was last seen about 4 and a half hours ago on computer Santa."
[14] *Watcher* is an X windows application developed 1990 by Steve Tufty at the Media Lab. It not only gives back the last login time and location of a user, but also the nearest phone number, last host and current host state, as well as fast access to email, voice mail, and an efficient alerting possibility. Here is a sample Watcher window:



Upon request of the ratified senders, the system can also send back information about the status of the message delivery, the assumed location of the user, and the most recently used communication channels.

Figure 9: **Matching priorities to channels, depending on the user's location** shows that depending on the importance of an incoming message, Active Messenger selects several channels sequentially, awaiting the user's reactions. Depending on the user's location, different phone or fax numbers are used. The user can limit the availability of the device for each location and channel.

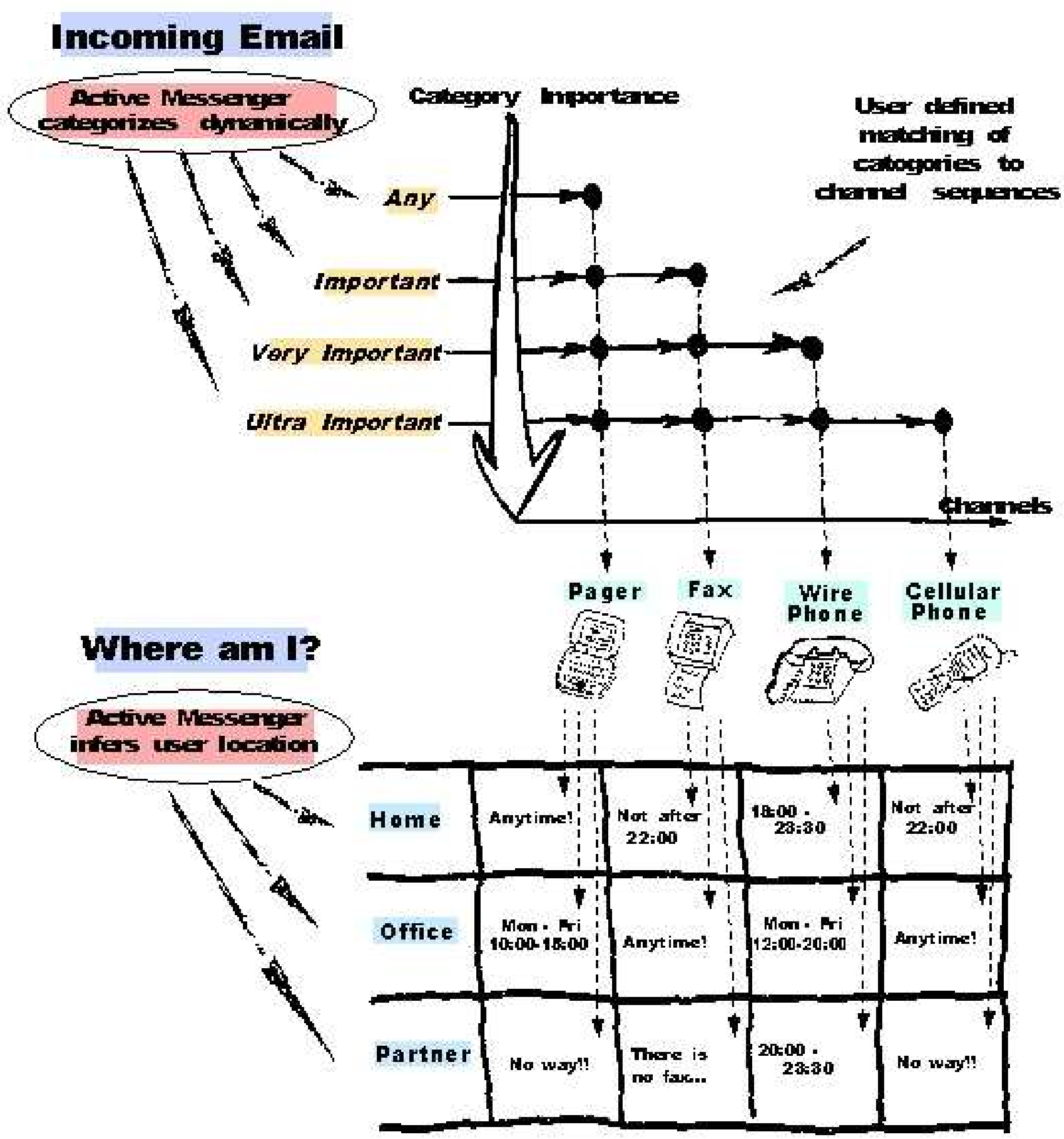


Figure 9: Matching priorities to channels, depending on the user's location

To keep the configuration of the Active Messenger simple, there is just one preference file (see Figure 7: **Sample user preference file**). It contains all relevant information about the user, such as her current communication infrastructure and preferences about what actions are appropriate at what time of day or week, and which aren't.

2.2 Current implementation of Active Messenger

- **Software:** Active Messenger is a single PERL script, currently about 5000 lines long and consisting of about a quarter of a million characters. It uses several helper scripts, mainly written by other authors. The user can run Active Messenger by executing a local copy of the script file, or the original file over the network.
- **Hardware:** Active Messenger is explicitly designed to run on several different machine types and under different operation systems. Currently, it is running on a SUN SPARCstation under SunOS 4.1.4, and on a DEC Alpha.
- **Other communication infrastructure:** Because the purpose of Active Messenger is to interact with other communication systems such as pagers, fax machines, wired and cellular phones, it doesn't make sense to run it completely without any such subsystem. However, no specific communication system is required. Active Messenger adapts to the specific communication environment of the user. The only requirement is that the user has a standard UNIX mail spool file.
- **User interaction:** The user interacts with the Active Messenger only by editing her user preference file (see Figure 7: **Sample user preference file**). The current status of Active Messenger can be viewed on the user web page (see Figure 8: **Sample status monitor page**). Additionally, Active Messenger writes log files.

2.3 Systems that Active Messenger can interact with

In order to understand the requirements for and the functionality of the Active Messenger, the systems will be described that it is able to interact with, focusing on the characteristics that are relevant for the Active Messenger. These are the systems it interacts with at the moment:

- *Canard* [5][6]: MIT campus wide two-way paging system.
- *SkyTel*™ [27]: USA nation wide two-way paging system.
- *Iridium*™ [13]: world wide one-way paging system.
- SMS Short Messaging Service: two-way paging system built into all GSM cellular phones in Europe and Asia.
- Fax machines and fax modems.
- UNIX mail spool file: where all incoming email messages get stored usually.
- Voice pager [20]: one-way paging system that plays voice mail messages or synthesized email.
- Wired and cellular phones.

These are the system that will be integrated, but it is not done yet:

- *ICQ*™ [12], Internet tool that informs the user who is currently online.

2.3.1 Heterogeneous communication environment

The communication devices that Active Messenger has to interact with have different characteristics, see Table 1: **Characteristics of some communication channels**.

E.g., some of them are one-way, and others are two-way. The former means that the information flow goes only in one direction: the user receives information. Two-way, however, means that a user can receive as well as send messages from this channel. A telephone is clearly two-way, a generic paging system is only one-way.

Another important feature is buffering. A device is buffered if a message will reach the user eventually, even if the device is temporarily out of range or switched off. Buffering means that the service provider tries to deliver the message again later, possibly several times, until the message has arrived. Some channels are buffered, some aren't. If they aren't, Active Messenger tries to compensate for that.

However, the main problem Active Messenger has to solve is that not all communication channels provide the agent with information about if a device is in range, if a message arrived to the device, or if it was read by the user. This is important for an agent that watches the user over time, awaiting her reactions to sending messages to a channel, and based on these reaction deciding which channel to use next.

Active Messenger tries different strategies to compensate for this missing information. The agent watches the user and tries to infer from her behavior if a device may be able to receive messages, if a single message arrived and was read by the user.

Table 1: Characteristics of some communication channels

	Is device two-way?	*Is device buffered?*	*Info about device in range?*	*Info about message received?*	*Info about message read?*
Canard pager [5][6]	Yes	Only for 10 minutes	Only *back in range*	Yes	Possible
SkyTel™ pager [27]	Yes	Yes	No	Yes	No
Short Messaging Service (GSM cellular phone)	Yes	Yes	No	No	No
Iridium™ pager [13]	No	Yes	No	No	No
Fax machine	Yes	Yes	Yes	Yes	No
Playing message to voice pager [20]	No	Yes	No	No	No
Playing message to phone	Yes	Yes (answering machine)	Yes	Yes	No
Playing message to cellular phone	Yes	Yes (voice mail)	Yes	Yes	Yes
Phoneshell [26] (phone interface to email and voicemail)	Yes	Yes	Yes	Yes	Yes
Mirabilis ICQ [12], AOL Instant Messenger [1]	Yes	Yes	Yes	No	No
UNIX mail spool file	Yes	Yes	Yes	Yes	Yes

2.3.2 Canard paging system

Part of the *Canard Community Messaging* [5][6] project at the MIT Media Lab is the two-way paging system. The range is limited to 2 - 3 miles, which covers MIT campus. 70 - 80 undergraduate and graduate students use this system to receive and send messages around the Media Lab and MIT.

Table 2: Characteristics of the Canard paging system

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
Canard pager [5]	Yes	Only for 10 minutes	Only *back in range*	Yes	Possible

The Canard paging system is based on the commercially available PageWriter™ 2000 [22] handset by Motorola (see Figure 10: **PageWriter™ 2000 by Motorola**). The backlit graphical display of this hand-held device can display nine lines of text, each 28 characters long. Therefore, it is big enough for a user to read email messages comfortably. The 47-key QWERTY keyboard (size 3½" x 2") allows to compose reply messages and originate new messages. Although touch-typing with 10 fingers is not possible, many users are comfortably typing with two fingers.

© 1999 Motorola

Figure 10: PageWriter™ 2000 by Motorola

Canard is an in-house research project at no costs for the users, and therefore the two-way text communication channel that is explored the best. The Canard user can log in to a secure web page that shows which messages were sent to Canard, and if they were received by the pager or not. Especially for Active Messenger, new features were implemented. E.g., to identify the messages on the web page, more information had to be displayed. It was also possible to suggest new features; e.g., we also know when the pager comes back in range. In this case, the Canard system sends the user a short email message, known as a “Quack” message, indicating that the user can receive messages again on his pager.

The Canard system is buffered, which means that it resends messages four times that were not received by the pager. However, resending happens only during the first ten minutes upon arrival of a message at the Canard server. After this time, if the pager is still not in range or switched off, the message is lost and will never arrive at the pager. Active Messenger was built to compensate for that shortcoming.

Theoretically, the hardware of the PageWriter™ 2000 allows checking if the user has read a message, a feature that would be unique among our communication channels that use mobile devices. Unfortunately, the manufacturer never implemented it. However, Active Messenger tries to infer from the user’s behavior to compensate for that as well.

Several other shortcomings of the Canard paging system are fixed by Knothole [14]. E.g., the rather cryptic reply format of Canard is modified by the Knothole service[15].

2.3.3 SkyWriter™ paging system by SkyTel™

SkyTel™ [27] has a two-way paging service called *SkyWriter*™. The coverage area is USA nation wide, in major metropolitan areas.

Table 3: Characteristics of the SkyTel™ paging system

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
SkyTel™ pager [27]	Yes	Yes	No	Yes	No

Handsets like the PageWriter™ 2000 by Motorola can be used as well as pagers of other brands. Currently, we are using an enhanced *Tango*™ device [28] (see Figure 11: **Tango™ by Motorola**). This is the world’s first two-way pager. Although the functionality is almost identical to the one of the PageWriter™ 2000, the significant difference is the user interface. There is neither a complete keyboard,

[15] For Knothole users, all messages to and from Canard are routed through Knothole. Therefore, it can reformat replies so that they seem to come from the normal email address of the user, and not from the pager address. Knothole complements the Canard system in several ways, but can be used independently from any specific two-way paging hardware, e.g., it works also with SkyWriter™ and SMS capable devices.

nor a graphical screen. Only six buttons and a three-line display are available. Although it is fully two-way, the simple user interface make it more awkward to read and write email messages. Knothole compensates for some of these shortcomings.

Figure 11: Tango™ by Motorola

It is possible to query the status of a message sent to SkyTel™ if—and if only—one sends the message through SkyTel™'s web interface. Like that, the sending party gets a "receipt" which consists of a message status number. With this number and the PIN code of the pager, the sender can request a web page that shows if this message was delivered successfully to the pager. Obviously, checking the status of a message is not possible when it is sent directly to the pager's email address.

Although SkyTel™ claims that SkyWriter™ service is fully buffered, which means that every message should get delivered eventually, we have other experiences. Over three months of testing the service with Active Messenger, several times outstanding messages were not delivered even after the pager came back in range. Because we couldn't rely on the assumption that every message sent to SkyTel™ will arrive eventually, special subroutines were designed for Active Messenger to make sure a SkyTel™ message gets delivered. E.g., after two messages don't arrive, Active Messenger stops sending messages and postpones them. As soon as the pager can receive email again, it resends these messages, if they are not yet read otherwise. However, sometimes a SkyTel™ device seems to be able to receive message after having been out of range, but the SkyTel™ service provider is not aware of that and therefore doesn't try to deliver outstanding messages. Active Messenger is able to send a "wake up" message in regular intervals. If it arrives at the pager successfully and the page acknowledges the arrival, SkyTel™ resends automatically the outstanding messages.

There is not information available about if the pager is in range, just came in range, or just went out of range. Active Messenger tries to infer these events and states by comparing the messages that were sent with the information available on the web page, as well as by looking at incoming messages. E.g., if a message arrives from the user's SkyTel™ pager, Active Messenger correctly considers the pager as receiving again, even if the SkyTel™ service provider for some reason does not consider the handset as in range.

2.3.4 Iridium™ paging system

Iridium™ became the world's first global satellite phone and paging system on November 1, 1998. Its network of 66-low earth orbiting satellites, combined with existing terrestrial cellular systems, enables customers to communicate around the globe. The paging system is a one-way. The subscriber gets the messages directly from the satellites (see Figure 12: **Iridium™ overview**).

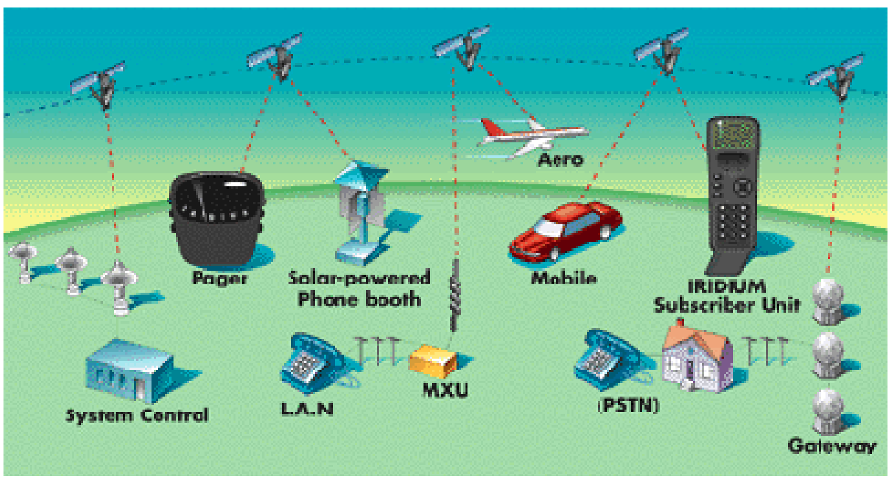


Figure 12: Iridium™ overview

Table 4: Characteristics of the Iridium™ paging system

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
Iridium™ pager [13]	No	Yes	No	No	No

Although it is not two-way, it is the ideal complement to the local Canard and nation wide SkyWriter™ service. If the user is out of range of these systems, she still may be able to receive messages through Iridium™. If the user misses some messages because the handset was turned off or was not able to receive for some other reasons, she gets notified that she has missed some pages. Unfortunately, the sending party has absolutely no information about the status of a message: neither if it has arrived at the Iridium™ server, nor if the handset has received it. Therefore, there is not enough back-channel information so that an external agent like Active Messenger could resend lost messages. But because Iridium™ is buffered, this shouldn't be necessary anyways.

There are two handsets available right now, one of them is shown in Figure 13: **Satellite Series 9501 Pager by Motorola**.

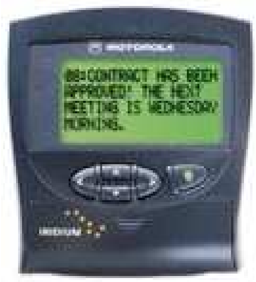


Figure 13: Satellite Series 9501 Pager by Motorola

© 1999 Iridium LLC

2.3.5 Short Messaging Service (SMS)

SMS is very popular[16] all over the world outside the United States because it is built into all GSM cellular phones[17]. SMS capable devices can send and receive messages of a length of up to 160 characters. Although SMS is two-way, sending messages from a cellular phone is awkward, because most cellular phones have no full keyboard[18]. On the other hand, receiving SMS messages is automatic and even possible if one is talking on the cellular phone.

Table 5: Characteristics of SMS capable devices

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
Short Messaging Service (GSM cellular phone)	Yes	Yes	No	No	No

[16] GSM holds 60% of the world's total cellular market. In April 1999, over 1 billion SMS message were sent.
[17] Only recently a GSM network was launched in the United States. Unfortunately, this GSM network is not compatible to the rest of the world. In Europe and Asia, GSM sends on 900 MHz and 1800 MHz respectively, in USA on 1900 MHz. Dual band devices are available, triple band is still rare.
[18] Handsets like the Nokia 9000 are still an exception.

SMS is buffered, so every message sent to an SMS cellular phone will arrive. However, there is no information available about if the handset is in range, or if a specific message has arrived.

Figure 14: SMS message on a Nokia 6190

2.3.6 ICQ™

ICQ™ by Mirabilis [12] is an example for an Internet tool that informs the user who is currently online. Based on the idea of a "buddy list," it simplifies the search for friends or associates on the net. ICQ™ alerts the user when the people on her buddy list log on.

Table 6: Characteristics of ICQ™

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
Mirabilis ICQ [12]	Yes	Yes	Yes	No	No

ICQ™ supports different modes of communication, e.g., email, URL and file transfer, chat, voice, message board, data conferencing. It supports also a variety of popular Internet applications and serves as a universal platform from which one can launch peer-to-peer applications such as Microsoft™ NetMeeting™ or Netscape™ CoolTalk™. It can also be used in a multiple-user mode for conferencing. The program runs in the background of a PC or a Mac, so while the user works with other applications, ICQ™ alerts her when friends and associates log in.

Active Messenger uses the email interface to ICQ™. Because the user has a specific email address at ICQ™, it is possible to send her a message directly and she will get alerted immediately on her buddy list (see Figure 15: **Sample ICQ™ window**).

Figure 15: Sample ICQ™ window

Relevant for Active Messenger is that ICQ™ provides additional information about if a user is online or offline. Unlike the earlier described Activity Server [16] that is based on *finger* and is used for *Locate* and *Watcher*, the ICQ™ user can set her availability manually. She can select from a list of different online and offline states (see Figure 16: **ICQ™ availability possibilities**) what other people will see if they try to contact her.

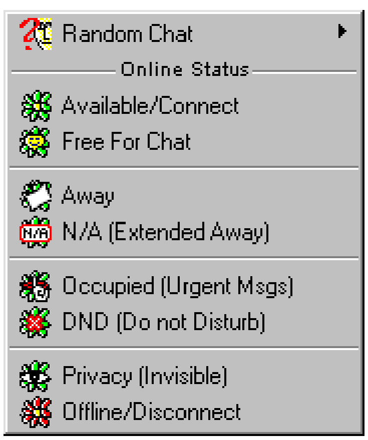


Figure 16: ICQ™ availability possibilities

2.3.7 Fax machines

Although fax machines seem to be a somehow antique technology[19], they are still very popular[20], especially since machines are common that combine phone, answering machine, and fax in one desktop device. Additionally, many users today have modems that are able to send and receive faxes directly from and to the PC.

Table 7: Characteristics of fax machines

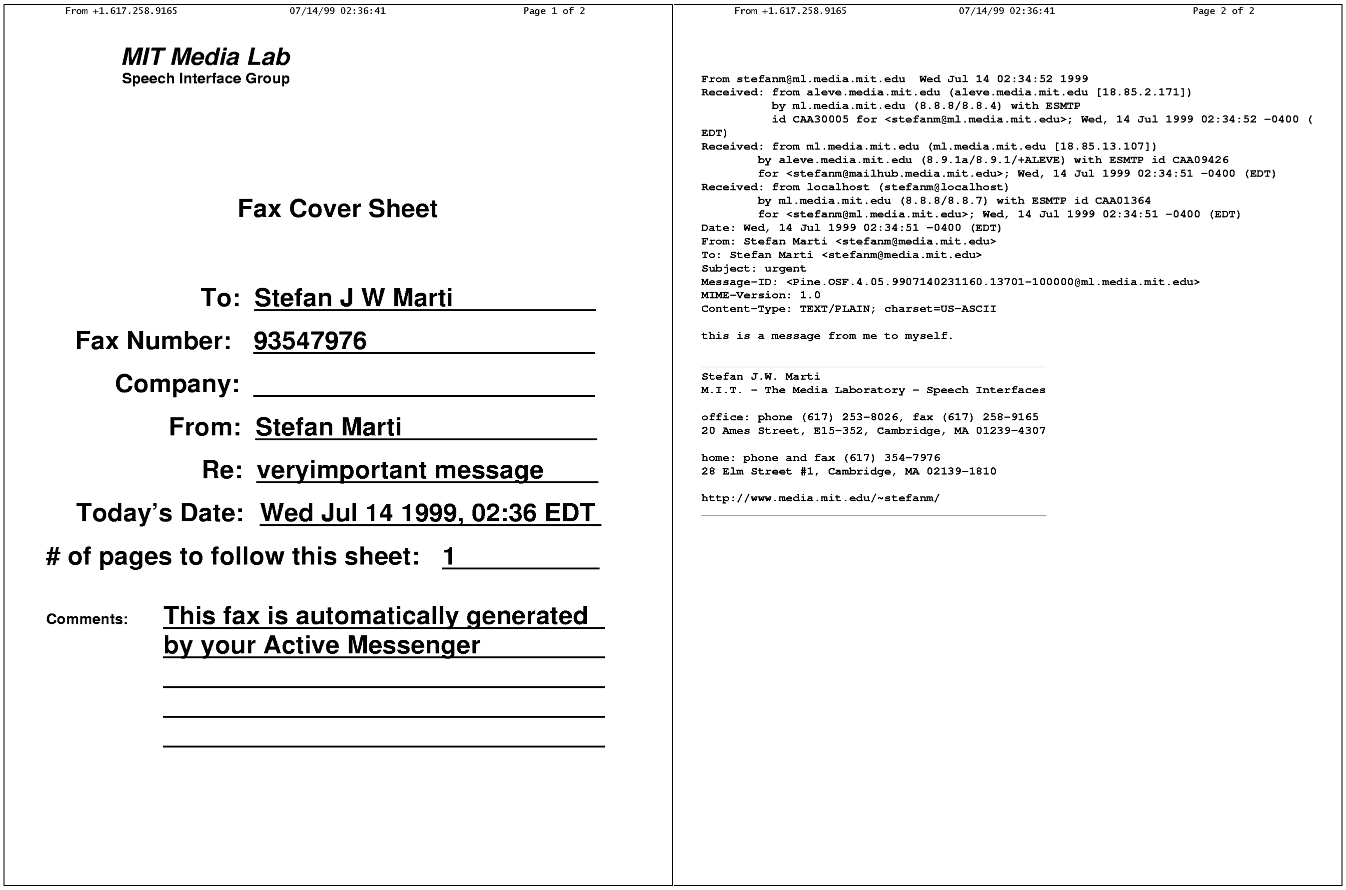
	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
Fax machine	Yes	Yes	Yes	Yes	No

Although we know if a fax was sent successfully and has arrived, we can't know if the user has read it.
To send faxes, we are running a HylaFAX[21] server that can send either the body of the email message itself, or add an extensive cover page (see Figure 17: **Sample fax, created by Active Messenger, including a cover sheet**).

[19] A leader of a European research lab laughed out loud when he learned that Active Messenger could send faxes. He thinks that faxing is an anachronism.
[20] "Today there are about 40 million fax machines installed worldwide. Leading market researcher, BIS Strategic Decision Inc projects that the fax transmission business will increase by more than 20% per year over the next four years:

- Fax transmission costs worldwide are approximately US$45 billion a year
- Approximately 50% of all international telecommunication traffic is from fax
- Revenue generated from international fax communication is over 10 times greater than any other form of electronic messaging
- 50% of fax users say they are faxing more
- There are over 40 million fax machines in the world
- Fax volume doubles every two years outside of the United States
- Fax traffic is the major form of communication between countries where the time difference is equal or greater than the eight-hour workday." [29]

[21] http://www.hylafax.org/



From +1.617.258.9165 07/14/99 02:36:41 Page 1 of 2

MIT Media Lab
Speech Interface Group

Fax Cover Sheet

To: Stefan J W Marti

Fax Number: 93547976

Company:

From: Stefan Marti

Re: veryimportant message

Today's Date: Wed Jul 14 1999, 02:36 EDT

# of pages to follow this sheet: 1

Comments: This fax is automatically generated by your Active Messenger

From +1.617.258.9165 07/14/99 02:36:41 Page 2 of 2

```
From stefanm@ml.media.mit.edu  Wed Jul 14 02:34:52 1999
Received: from aleve.media.mit.edu (aleve.media.mit.edu [18.85.2.171])
          by ml.media.mit.edu (8.8.8/8.8.4) with ESMTP
          id CAA30005 for <stefanm@ml.media.mit.edu>; Wed, 14 Jul 1999 02:34:52 -0400 (
EDT)
Received: from ml.media.mit.edu (ml.media.mit.edu [18.85.13.107])
        by aleve.media.mit.edu (8.9.1a/8.9.1/+ALEVE) with ESMTP id CAA09426
        for <stefanm@mailhub.media.mit.edu>; Wed, 14 Jul 1999 02:34:51 -0400 (EDT)
Received: from localhost (stefanm@localhost)
        by ml.media.mit.edu (8.8.8/8.8.7) with ESMTP id CAA01364
        for <stefanm@ml.media.mit.edu>; Wed, 14 Jul 1999 02:34:51 -0400 (EDT)
Date: Wed, 14 Jul 1999 02:34:51 -0400 (EDT)
From: Stefan Marti <stefanm@media.mit.edu>
To: Stefan Marti <stefanm@media.mit.edu>
Subject: urgent
Message-ID: <Pine.OSF.4.05.9907140231160.13701-100000@ml.media.mit.edu>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII

this is a message from me to myself.

_______________________________________________
Stefan J.W. Marti
M.I.T. - The Media Laboratory - Speech Interfaces

office: phone (617) 253-8026, fax (617) 258-9165
20 Ames Street, E15-352, Cambridge, MA 01239-4307

home: phone and fax (617) 354-7976
28 Elm Street #1, Cambridge, MA 02139-1810

http://www.media.mit.edu/~stefanm/
_______________________________________________
```

Figure 17: Sample fax, created by Active Messenger, including a cover sheet

This figure clearly shows the "overhead" of a single-line email message when sent to a fax machine. Therefore, if the user knows that a cover sheet is not necessary because the fax machine is personal, adding a comment behind the fax number lets Active Messenger send only the email message without any cover page. However, depending on the physical location of the fax machine, it can happen that a single-line fax message with only a few header fields can get lost in a stack of full-letter-sized fax messages.

2.3.8 Voice pager by Motorola

The voice pager by Motorola [20] (see Figure 19: **Voice pager by Motorola**) is a portable electronic device that receives voice clips as pages, up to a total length of four minutes. A script uses a telephone interface utility to dial up the voice pager answering machine and to leave the synthesized message.

Table 8: Characteristics of the voice pager by Motorola

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
Playing message to Voice pager [20]	No	Yes	No	No	No

The script that sends the message to the voice pager is based on the same code that Phoneshell [26] uses. To send a message, it calls up the voice pager number. Once connected to the voice mail system of the voice pager user, it waits until the announcements are over, and then a DECtalk™ unit synthesizes the email message body. No interaction is required. Figure 18: Transcript of a sample voice page shows an example of a message delivered as a voice page.

> Getting your messages. Ok, I have found one veryimportant message: Veryimportant message: Message one from John Doe about "dinner tonight."
>
> *Hey John, we will have dinner tonight with our parents, so feel free to stop by if you want! Ken.*
>
> No more messages. Rebuilding your mail file. Exiting MailTalk now. Goodbye!

Figure 18: Transcript of a sample voice page

Although the voice pager only receives pages and therefore is not two-way, the device responds to the Motorola system confirming message receipt. Unfortunately, we do not have access to this information. Because the coverage for the Motorola voice pager is USA nation wide and the delivery is buffered, Active Messenger assumes that all messages will be delivered eventually.

© 1999 Motorola

Figure 19: Voice pager by Motorola

The voice pager is an interesting channel that complements the mainly text-based other paging systems. Although the idea of a "portable answering machine" seems to be useful, many users of this device have complained about its bad user interface, especially the alerting mechanisms. Both of them, the audible alert as well as the vibration mode, are fairly loud. If the user does not listen to a message after the initial alert, the device continues the alerts until the message is played back.

2.3.9 Wired and cellular phones

Another possibility to get a message to a user is to call her up on the phone and let a text-to-speech module read the message to her directly. This is possible for any kind of phone, be it wired or cellular.

Table 9: Characteristics of wired and cellular phones

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
Playing message to phone	Yes	Yes (answering machine)	Yes	Yes	Yes
Playing message to cellular phone	Yes	Yes (voice mail)	Yes	Yes	Yes

The procedure as is almost the same as sending a message to a voice pager, except that the user can interact with the system calling her up by pressing the telephone keys. The user interface is almost identical to Phoneshell [26][22]. Once the user is logged in, the system announces the importance level of the message and reads out the header, immediately followed by the email body. During any point of the session, the user can navigate within the message, get additional information about when it was sent, can type in replies, leave voice mail replies, etc.

Depending on the user's actions during the phone call (keys pressed), Active Messenger can distinguish different levels of success for the message delivery.

Currently, the main problem is to detect if a person has picked up the phone or not, or if an answering machine is recording the call.

[22] Like Phoneshell, the application reads the email message from a mail-spool-like file. The Phoneshell code modifies this pseudo mail spool file at the end of the session. Afterwards, Active Messenger parses it similar to how it parses the standard mail spool file. See also chapter 2.3.10.

2.3.10 UNIX mail spool file

The mail spool file is where all incoming email messages in a UNIX system are stored automatically. The file itself is just a concatenation of all email messages in plain ASCII, including all email header fields. A new line and a special line beginning with the word “From “ are inserted to separate the messages[23]. See example in Figure 20: **Sample mail spool file with one email message**.

Because all messages are stored in the mail spool file automatically, Active Messenger doesn’t have to forward them to this channel—but at the same time can’t prevent them from getting there.

The mail spool file gives useful information about if a message is opened, or opened and read. Mail reader programs like PINE[24] access this mail spool file and modify it automatically. They can insert a header field for each message that says if the mail spool file was opened with the new message (*status: O*), or if the message itself was opened and therefore read (*status: RO*). See example in Figure 20: **Sample mail spool file with one email message**.

```
From johndoe Mon Jul 19 22:36:36 1999
Received: from aleve.media.mit.edu (aleve.media.mit.edu [18.85.2.171])
   by mn.media.mit.edu (8.8.8/8.8.4) with ESMTP
   id WAA05366 for <johndoe@mn.media.mit.edu>; Mon, 19 Jul 1999 22:36:33 -0400 (EDT)
Received: from mn.media.mit.edu (mn.media.mit.edu [18.85.13.107])
   by aleve.media.mit.edu (8.9.1a/8.9.1/+ALEVE) with ESMTP id WAA2
   for <johndoe@mailhub.media.mit.edu>; Mon, 19 Jul 1999 22:36:32 -0400 (EDT)
Received: from localhost (johndoe@localhost)
   by mn.media.mit.edu (8.8.8/8.8.7) with ESMTP id WAA17024
   for <johndoe@mn.media.mit.edu>; Mon, 19 Jul 1999 22:36:32 -0400 (EDT)
Date: Mon, 19 Jul 1999 22:36:32 -0400 (EDT)
From: John Doe <johndoe@media.mit.edu>
To: John Doe <johndoe@media.mit.edu>
Subject: hi world
Message-ID: <Pine.OSF.4.05.9907192236080.7640-100000@mn.media.mit.edu>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
Status: RO
X-UID: 9475

This is how a mail spool file looks like.
-John
```

Separator line

Status header line

Figure 20: Sample mail spool file with one email message

[23] The space after “From “ is important since there is another standard header field that starts with “From:” The latter is different from the former because the “From” is followed by a colon.

Additional information can be obtained by looking at the *access time* and *modification time* of the mail spool file itself. If a new message is appended to the file, the modification time changes. Similarly, if the user moves a message from the inbox to another folder: because the message is deleted from the mail spool file, the modification time changes as well. However, the access time is independent from the modification time. If a mail reader program like PINE, or a user opens the mail spool file, e.g., by tailing it, the access time of the file changes[25].

Table 10: Characteristics of a UNIX mail spool file

	Is device two-way?	Is device buffered?	Info about *device in range*?	Info about *message received*?	Info about *message read*?
UNIX mail spool file	Yes	Yes	Yes	Yes	Yes

In summary, Active Messenger can obtain accurate information about the status of each message by regularly looking at the file attributes of the user's mail spool file, possibly reading and parsing the file itself, and comparing the results with the last parse of the file. Note that the UNIX mail spool file is actually the only channel for which Active Messenger can determine if the user has read a message[26].

2.4 Program structure

Figure 21: **Active Messenger flow chart** shows the general structure of the main Active Messenger code.

There are three kinds of elements: the initialization elements (red), the main loop elements (green), and the involved files (yellow).

---

[24] http://www.washington.edu/pine/

[25] It was actually a programming challenge to mask out the automatic accesses of PINE to the mail spool file when a new message arrives.

[26] The same is true for reading a message to the user on the phone, since this method also uses a special kind of mail spool file that can be parsed the very same way as a standard mail spool file.

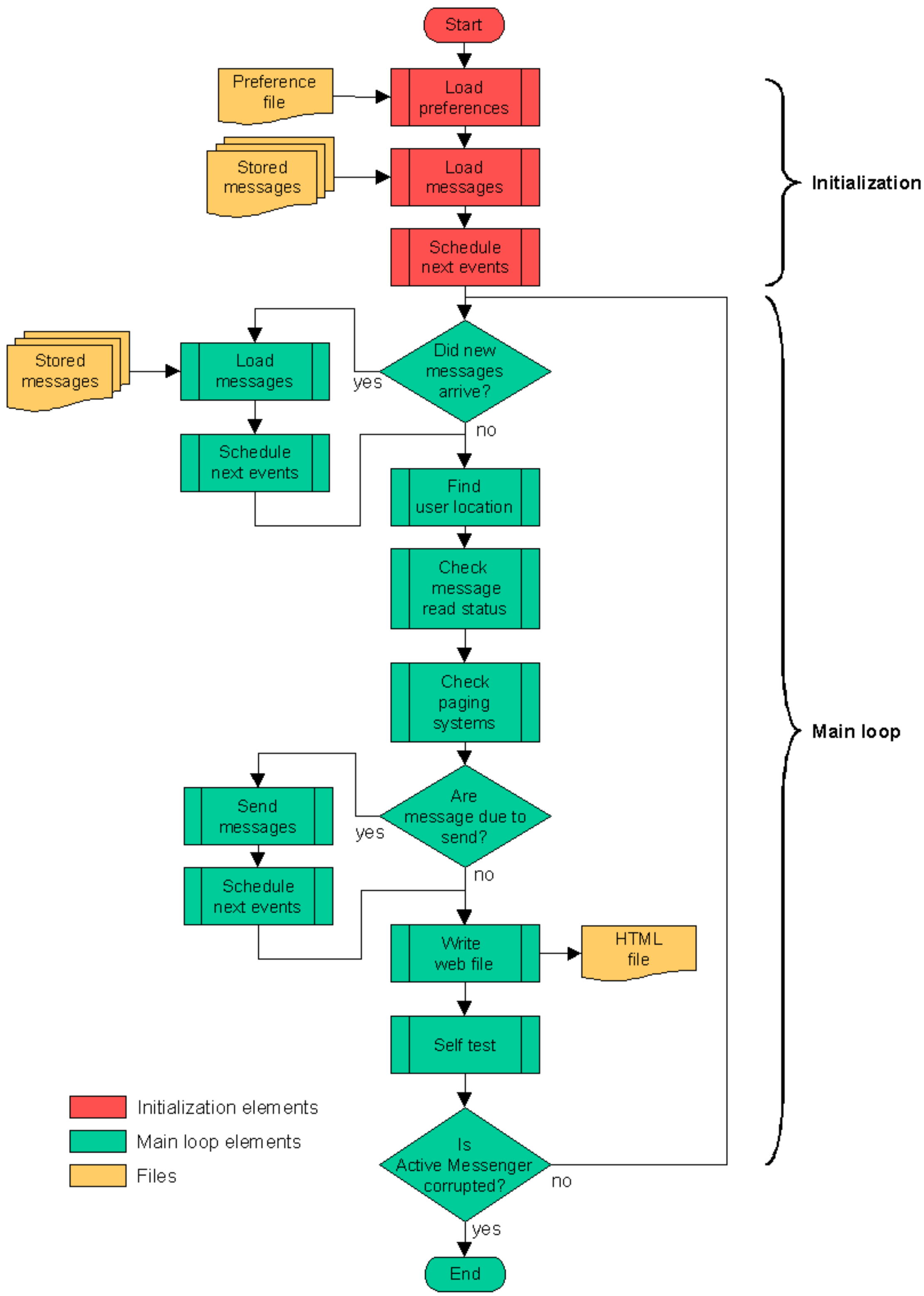


Figure 21: Active Messenger flow chart

2.4.1 Initialization

When Active Messenger starts up, it first loads the user preference file. After that, it loads all email messages from the files that were stored by a helper script (chapter 2.4.4). Loading a message means parsing the content of a message and keeping parts of the information in program memory, e.g., when it arrived, where it came from, the importance category of the message etc. The body of the message itself is not hold in program memory.

Then, for each message, an initial event is scheduled. This event is most often a time when a message has to be sent and a channel or device where it has to be sent to (chapter 2.6.5).

2.4.2 Main loop

After the initialization, the Active Messenger goes into an infinite loop where the following modules are executed sequentially:

- **Load messages** (chapter 2.6.1)**.** If new messages have arrived, load them into the message list and schedule the initial event, e.g., where and when to send them.
- **Find user location** (chapter 2.6.2)**.** Where may the user be?
- **Check message read status** (chapter 2.6.3)**.** Go through all messages: What is the likelihood that the message is read? Checking the mail spool file, Canard, SkyTel™, and other resources.
- **Check status of paging systems** (chapter 2.6.4)**.** Did the Canard or SkyTel™ handsets come back in range? Is it necessary to resend older messages?
- **Send messages** (chapter 2.6.6)**.** Go through all messages: If a scheduled event is due, send the message to the specified device.
- **Schedule next events** (chapter 2.6.5)**.** After sending a message, schedule the next event.
- **Write web file** (chapter 2.6.7)**.** Store the current status in a file that is accessible on the web.
- If user preference file has changed, **reload user preferences**. Like that, changes can be made to the behavior of the agent without necessarily restarting the program. By editing the user preference file, the user can change most of the internal variables.

After each run through the main loop, Active Messenger sleeps for 12 seconds, and then enters the same loop again[27].

The modules are described in detail in chapter 2.6.

2.4.3 Files

These are the main files Active Messenger requires for proper functioning or writes during execution:

- **User preference file**: In this preference file, all necessary information is stored for Active Messenger to work successfully. If it is missing, or any entries are missing, the agent uses default values, so if a user doesn't like editing large preference files, it can be kept, as Figure 7: **Sample user preference file** shows. The function and possible entries are described in chapter 2.7.1.
- **HTML file**: At the end of each main loop, Active Messenger writes a file that is readable from the web. It is not an interactive web page, but a monitor page that shows what state the agent is in, what happened in the past and what it plans to do in the future. What Active Messenger writes to this file is described in chapter 2.6.7. How the user sees the web page, including screen shots, is described in chapter 2.7.2.
- **Log file**: For important events, Active Messenger creates an entry in the main log file. Each entry starts with the current time and describes what happened or what Active Messenger will do next. More details are described in chapter 2.7.3.
- **Stored messages**: All incoming messages are stored in full in a specific directory. Chapter 2.4.4 describes why and how they are stored.

[27] With this delay, Active Messenger writes the HTML file, checks for incoming messages, and tests all peripheral sytems about four times a minute, which turned out to be sufficiently fast. It would be possible to run it without this sleeping period, but then Active Messenger would use irresponsibly much more CPU time without being much more efficient. In an early stage of the software development, Active Messenger was running without this delay on the main mail machine of the MIT Media Lab. The process was automatically killed after 30 minutes because it used 76.6% of the CPU time of this important workstation. There are other delays built into Active Messenger at various locations. The reason for them is not to slow down the performance of the system, but to emphasize important events. Stopping the otherwise continuous screen output of the agent for a few seconds gives the user time to read the screen.

2.4.4 Storing incoming email messages

Independently and asynchronously from the main PERL script, all incoming messages are immediately stored in a special Active Messenger directory by a program called *AM_store_messages* that is executed by the user's Procmail. Figure 22: **AM_store_messages flow chart** shows an overview of this program.

This program takes a full email message from UNIX Standard Input. First, it gives it a unique number by looking up the number of the last message stored and increasing it by one. This number is also the file name under which the message is stored in a specific directory. The user can specify how many messages she wants to store. The default is 100 messages. The more messages that are stored, the longer the "memory span" of Active Messenger will be. Once 100 messages are stored, the first one will be overwritten and is lost for resending. However, because Active Messenger keeps track of the incoming messages by storing part of the message in program memory and adding information to each message entry, storing more messages as files proportionally increases the amount of program memory Active Messenger uses. The more messages are stored, the more resources Active Messenger uses. This may or may not be a problem, depending on the available RAM space of the workstation Active Messenger is running on.

The script does also a first-level filtering. Not all incoming messages are important enough so that they have to be forwarded in any way. By reducing the amount of initially stored messages, Active Messenger is able to "remember" more important messages in the past. For the first-level filtering, the helper program reads from the user's preference file a list of email addresses that can be ignored. Then it extracts the sender from the incoming email and proceeds only if the email address is not on the ignore list. Like that, the user can reduce easily the amount of messages that are stored by Active Messenger in the first place. There are other possibilities to reduce the amount of initially stored messages, e.g., by adding rules to the .procmailrc file. However, it was intended to keep the configuration of the Active Messenger simple and in one place: the user preference file.

If the incoming message is not on the user's ignore list, it is stored to a file. The last step is to feed the message into *Clues_filter* (see chapter 1.2.2). This utility gives back the importance level of the message in the form of a message category. This category is also written to a file for each message[28]. After that, *AM_store_messages* terminates.

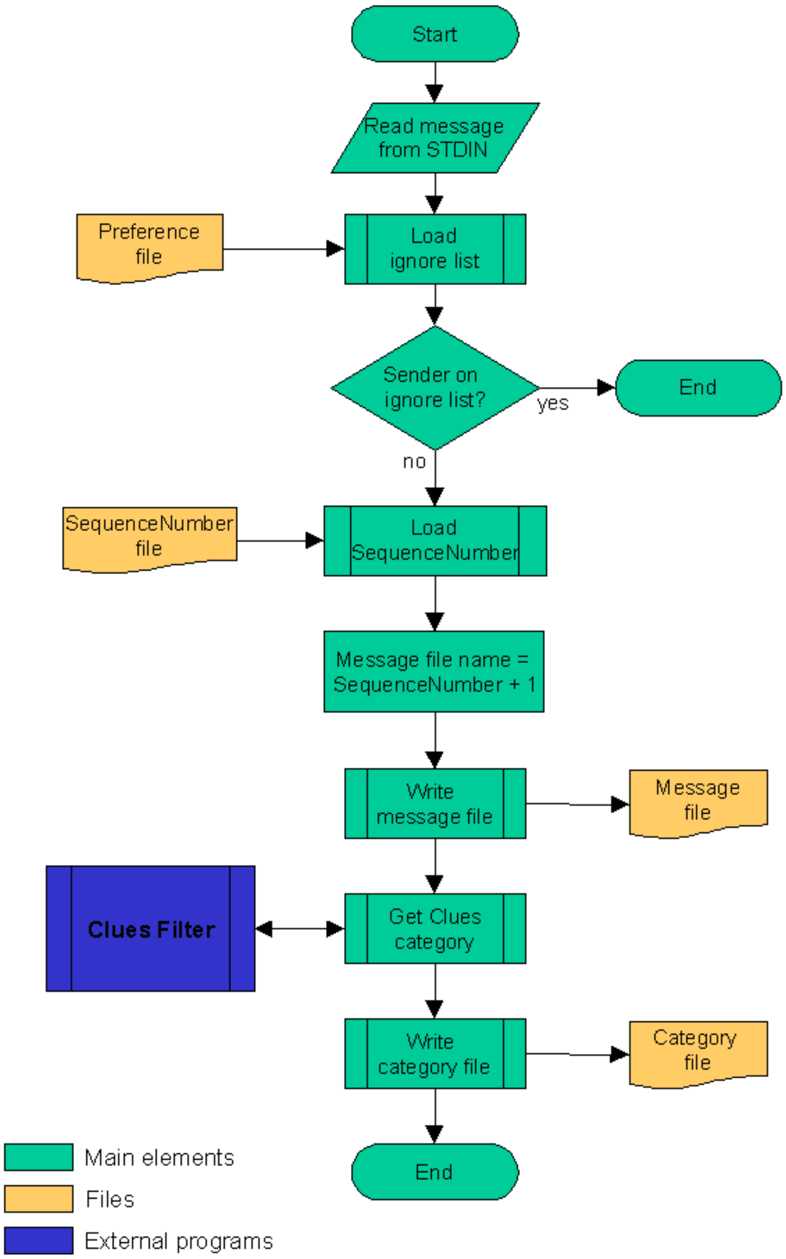


Figure 22: AM_store_messages flow chart

[28] The reason why the category is written to a file is that *Clues_filte*r needs a few seconds to compute a message category. This is acceptable for a single message, but if Active Messenger starts up and loads, e.g., 100 messages, it would take at least three to four minutes to extract the categories of all messages. Therefore, the script that stores the incoming messages computes the category right away and stores this information in a separate file. Therefore, when Active Messenger loads a message from a file into its memory, it looks also for the corresponding category file. If it is not there for some reason, or if it seems to be corrupted, it feeds the message into *Clues_filter* again.

2.4.5 Starting and restarting Active Messenger

Because Active Messenger keeps most data in RAM memory, it loses its sense of what happened in the past when it restarts. Therefore, it is important that Active Messenger run continuously.

Initially, the user can start Active Messenger by executing the PERL script manually, from a local file or over the network. After that, another script called *Runapp* checks if the application is running and restarts it automatically if Active Messenger is not running anymore. There are three main reasons why Active Messenger would not be running anymore:

- The machine where Active Messenger was running on was rebooted.
- Active Messenger encountered a programming error.
- Active Messenger exited deliberately.

The reason why and when a machine is rebooted is out of control for the programmer of an application. If Active Messenger dies because of a programming bug, the code has to be debugged. However, the third reason has to be explained, because it is not intuitive why Active Messenger sometimes has to exit deliberately.

Because it is imperative that Active Messenger run continuously, much effort was spent on making the code robust and not to crash easily. One of the reasons why Active Messenger can become inoperable is if it "hangs" at a certain point of the code execution. Good programming should avoid that. However, there are cases in which the programmer has no influence on hanging, e.g., when the agent executes external programs or commands that hang themselves. To prevent Active Messenger from stalling because external programs do not respond anymore, all external program calls are timed out. This means, Active Messenger waits only a certain amount of time for an external program to return[29]. After that time, Active Messenger continues without waiting for the result of the external call.

Unfortunately, these abandoned processes become defunct processes, also known as "zombie" processes. They do not use CPU time, only kernel book keeping space. They can't be killed until the parent process, in our case Active Messenger, terminates. Some operating systems allow only a limited amount of processes per user. "Zombie" processes add up to the amount of processes owned by the user. If the

[29] The user can specify for each external call the amount of time until timeout. For testing purposes, Active Messenger can log the time that each external program uses. This information, written to a file, is an excellent source of information to find out about the overall "healthiness" of Active Messenger and its peripheral systems.

process limit is reached, no more processes are allowed. At this point, Active Messenger can't take any actions anymore, e.g., can't send messages, or can't check external resources anymore.

To avoid this situation, Active Messenger exits (and is hopefully restarted) if there are too many "zombie" processes. The agent is aware of these processes and can count them because each timed out external call creates a "zombie." Once Active Messenger exits, all "zombies" are released. Afterward, Active Messenger has to be restarted, either manually or automatically by *Runapp*.

If the machine was rebooted where Active Messenger was running on, *Runapp* also can restart Active Messenger. However, in the case of a newly restarted machine, the *Runapp* process is owned by root, as every startup process. Therefore, all applications started by *Runapp* are also owned by root. This causes a problem because Active Messenger relies on environment variables and access privileges that are user specific. User root can't run Active Messenger. For this reason, another C file was written called *Wrapper* that can be run as root, but sets the user ID and other environment variables according to the arguments that were given to the program, and then executes Active Messenger in the user's name.

2.5 Data structure

The main data structure, the message list, holds all necessary information that is relevant for a specific message. For each message, 34 parameters are stored, e.g., who sent the message, when did it arrived, how was it read (from which channel or device), what and when will be the next action to take, what actions were already taken for this message.

Other data structures hold additional information, e.g., a list of all errors that occurred, all user preferences, and other device-specific information.

2.5.1 Message list

The message list is a two-dimensional array (Table 11: **Message list as two-dimensional array**) that is implemented in PERL as

$message[MessageNumber][FieldNumber].

The first dimension is the message number. A message number is assigned to each incoming message by the helper file *AM_store_messages*, which stores the incoming message also to a file, see chapter 2.4.4.

The default size is 100 entries, which is equivalent to the 100 messages stored to files. However, the user can specify the size. Because a new incoming message overwrites all former entries at its list position, Active Messenger keeps track only of the messages that are currently stored to files.

The second dimension describes attributes of the message in detail. It is a record of (currently) 34 fields. Active Messenger initializes the values of the fields when a new message comes in, overwriting the old values. During the "life span" of a single message, different modules of Active Messenger edit and update the fields continuously.

Table 11: Message list as two-dimensional array

MessageNumber

FieldNumber	0	1	2	...	99
0					
1					
...					
33					

The 34 fields can be grouped in six sections:

- Basic message attributes (fields 0 – 5)
- Attributes describing if a message is read (fields 6 – 9)
- Last event attributes (fields 10 and 11)
- Next event attributes (fields 12 – 16)
- History of the message (field 17, a two-dimensional array on itself)
- Device specific attributes (fields 18 – 33)

A detailed description of the fields follows in Table 12: **Message list**.

Table 12: Message list structure

Field	*Description*	*Example*	*Value range*
Basic message attributes			
0	**Message ID**: A unique string of the "Message-ID:" header field identifies each message.	"Pine.OSF.206.9207140231160.137013542340@ml.media.mit.edu"	Any unique string
1	**Arrival time, alphanumeric**, from the "Date:" message header field.	"Sun Mar 7 09:10:07 1999"	Standard date format: *"weekday month day hours:min:secs year"*
2	**Arrival time, in seconds**: This time is computed by Active Messenger by transforming the alphanumeric date (field 1) into seconds since the beginning of "UNIX time," which was at Dec 31 19:00:00 1969 EDT.	"920853001"	Integer
3	**Message category**, computed by Clues (chapter 1.2.2).	"important"	User specific
4	**Clues status**: This reflects the "healthiness" of the Clues utility. If there are several messages which don't have "0" here, Active Messenger exits, because it can also indicate that the computer is out of memory.	"0"	0 = OK 1 = Clues not running 2 = Clues segmentation fault 3 = return was empty 4 = request timed out
5	**Sender of message**, from the "From:" message header field.	"John Doe <johndoe@media.mit.edu>"	String
Attributes describing if a message is read			
6	**Message read likelihood**: This shows the current estimation of Active Messenger on how likely it is that the user read the message.	"85"	0 - 100
7	**Message read from where**: A message can be read from different places, like from the mail spool file, a pager, etc.	"spool file on k2"	Strings, e.g., "spool file of host" "heard on phone" "canard"
8	**Message read when**, in seconds	"920853001"	See field 2
9	**Message read how (details)**: More details about how the message was read.	"gone"	Different strings, depending on how it was read, e.g., if from the mail spool file, it can be *opened* (O), *opened and read* (RO), or already "expunged (GONE). See also chapter 2.3.10.
Last event attributes			
10	**Last event what**	"fax"	Any device or channel, user specific
11	**Last event when, in seconds**	"920853001"	See field 2

Next event attributes			
12	**Next event what**	"skytel"	Any device or channel, user specific
13	**Next event when, in seconds**	"920853001"	See field 2
14	**Next event number or address**	"johndoe@canard.media.mit.edu"	Any string
15	**Next event details**, not in use.		Any string
16	**Next event why**: Explanation why the next channel or device is valid; expresses a time range that is true for the next event time.	"M-F 10-12"	Specific time range formats described in chapter 2.7.
History of the message			
17	**History of past events**. This field is a two-dimensional array itself. Active Messenger transfers the "next event" fields here after the event is over, so it adds a record for each past event. Each record has 4 fields:		
	0 **Past event, what**	"iridium"	From field 12
	1 **Past event, when (seconds)**	"920853001"	From field 13
	2 **Past event number/address**	"764-9243"	From field 14
	3 **Past event, why**	"not F"	From field 16
Device specific attributes			
18	**Potential PagerSequenceNumber:** This is the number a message is given when it is sent to a pager. It is important to Knothole because this number identifies replies. See also chapter 1.3.	"66"	0 - 99
19	**Potential SkyTel™ ID**: When a page is sent through SkyTel™'s web interface, the sender gets back this number as a receipt, and she can check the delivery status of a certain message by referring to it.	"4323"	N/A
20	**Arrived at Canard?** Available by checking the Canard web pages.	"yes"	" " = not sent "no" = sent but not arrived "yes" = arrived
21	**Arrived at Canard when**, seconds	"920853001"	see field 2
22	**Arrived at SkyTel™?** Is obtained by looking up a message by referring to its message number on the web page.	"no"	" " = not sent "no" = sent but not arrived "yes" = arrived
23	**Arrived at SkyTel™ when**, secs	"920853001"	See field 2
24	**Potential fax ID**: The HylaFAX agent (see chapter 2.3.7) gives out this ID after sending.	"67"	
25	**Sent to Canard when**, seconds	"920853001"	See field 2
26	**Sent to SkyTel™ when**, seconds	"920853001"	See field 2
27	**Sent to SMS when**, seconds	"920853001"	See field 2
28	**Sent to Iridium™ when**, seconds	"920853001"	See field 2

29	**Sent to Fax when**, seconds	“920853001”	See field 2
30	**Sent to Voice Pager when**, seconds	“920853001”	See field 2
31	**Sent to Phone when**, seconds	“920853001”	See field 2
32	**Sent to Phone number**: This is the last phone number the message was sent to. It is important because the device “phone” can mean many different numbers.	“(617) 546-8573”	Any string
33	**User’s reaction to phone call**: The user may or may not have pressed any keys. See chapter 2.3.8.	“responded”	“heard it” “responded” “no reaction”

2.5.2 User preferences hash

All user preferences are loaded dynamically from the preference file and stored in a hash structure. More precisely, it is a *hash of hashes of hashes of arrays of hashes*. This rather complex structure was chosen because the user preferences are individual and not predictable. Hashes seem to be the most appropriate data structure for this purpose. After having loaded them from the file into the hash structure, Active Messenger translates the retrieved values into the internally used variables.

The structure of the user preferences hash reflects closely the structure of the preference file itself, see chapter 2.7. Each section of the preference file is loaded into one element of the main hash. Mandatory sections are Mapping, Locations, Devices, and Files:

- **Mapping**. Each category is assigned a sequence of channels and devices.
- **Locations**. List of all possible locations.
- **Devices**. List of all devices that are not yet defined at a specific location.
- **Files**. All user specific path names and other preferences.
- All other sections are locations, like **home** or **office**, where location-specific devices can be defined.

Table 13: **User preferences hash structure** shows an example of such a hash structure.

On the status monitor web page, the whole content of the user preferences hash is displayed, see chapter 2.7.2.

Table 13: User preferences hash structure

$preferences{hash1}	***{hash2}***	***example***	***[array]***	***example***	***{hash3}***	***example***
$preferences{**Mapping**}	{**other**}	= “canard (5)”	[**0**]	= “canard (5)”	{**device**}	= “canard”
					{**delay**}	= “5”
	{**important**}	= “fax (2), skytel”	[**0**]	= “fax (2)”	{**device**}	= “fax”
					{**delay**}	= “2”
			[**1**]	= “skytel”	{**device**}	= “skytel”
					{**delay**}	= “10”
	{**veryimportant**}	= “vpager (6), skytel, phone(3)”	[**0**]	= “vpager(6)”	{**device**}	= “vpager”
					{**delay**}	= “6”
			[**1**]	= “skytel”	{**device**}	= “skytel”
					{**delay**}	= “10”
			[**2**]	= “phone(3)”	{**device**}	= “phone”
					{**delay**}	= “3”
$preferences{**Locations**}	{**locations**}	= “home, office”	[**0**]	= “home”		
			[**1**]	= “office”		
$preferences{**home**}	{**cellphone**}	= “818 2222, T 3-4, S 17:00-24”	[**0**]	= “818 222”		
			[**1**]	= “T 3-4”		
			[**2**]	= “S 17:00-24”		
	{**phone**}	= “234234, M 17:00-22:00”	[**0**]	= 234234”		
			[**1**]	= “M 17:00-22:00”		
	{**fax**}	= “123233, F-M 8:00-1:32, not W”	[**0**]	= “123233”		
			[**1**]	= “F-M 8:00-1:32”		
			[**2**]	= “not W”		
$preferences{**office**}	**...**					
$preferences{**Devices**}	{**vpager**}	= “435-3345”				
	{**iridium**}	= “2342@iridium.com”				
	{**SMS**}	= “131@omnipoint.com”				
	…					
$preferences{**Files**}	{**userdir**}	= “$ENV{‘HOME’}/”				
	{**finger_ml**}	= “0”				
	{**READ_RO**}	= “95”				
	…					

Here is an example on how the information is stored:

- The *first-level hash* $preferences holds all sections, e.g., “Mapping.”
- The section “Mapping” contains a *second-level hash* with all possible categories, e.g. “other,” “important,” and “veryimportant.”
- Each category, e.g., “important,” contains an *array* of all devices, e.g., “fax (2), skytel.”

- Each array element, e.g. “fax (2)”, contains a *third-level hash* with two elements, “device” and “delay,” where the device name, e.g., “fax,” and the associated delay, e.g., “2”, is stored. (If there is no delay time specified, Active Messenger assumes a standard delay of 10 minutes until a message is sent to the specific channel or device.).

Here is an example of how information is accessed in this data structure: As we have seen in the ”Mapping” section, the category “important” is defined by the channel sequence “fax (2), skytel.” Therefore, the delay time, 10 minutes, for the second channel of the sequence, “skytel,“ would be stored at *$preferences{Mapping}{important}[1]{delay}* = 10.

2.5.3 Known addresses hash

Each device can be identified uniquely only by its number or address, since the device name itself is not unique. E.g., there are several devices called “phone,” and it is possible that the user has several different ICQ accounts. The known addresses hash is the data structure that represents all available devices.

It is a hash of an array with three elements. Table 14: **Sample known addresses hash** shows the format and an example of how the data structure can look like.

Table 14: Sample known addresses hash

$known_addresses{$hash}	***[array]***	***example***	***description***
$known_addresses{**343 1443**}	[**0**]	= “phone”	**Device** that corresponds to the address.
	[**1**]	= “0”	The last **time** a message came from this address, in seconds. Each time a new message arrives from a known address, the time is updated here.
	[**2**]	= “home”	**Location** of the device. This can be at several locations at the same time.
$known_addresses{**32233@icq.com**}	[**0**]	= “icq”	
	[**1**]	= “9142344”	
	[**2**]	= “home, parents”	
$known_addresses{**343 1443 (no cover)**}	[**0**]	= “fax”	
	[**1**]	= “9142344”	
	[**2**]	= “home”	

The table also shows how Active Messenger deals with phones with included fax machines. In this case, the two devices have the same phone number, so it doesn't make them unique. When reading the preference file, Active Messenger adds to each fax number a string, either "(no cover)" or "(cover sheet)" that associated this number to a fax rather than to a phone.

Active Messenger updates the known addresses hash each time the user preference file is read. Additionally, the field indicating when the last message came from this address is updated continuously by several modules:

- Each email message that is routed through Active Messenger is compared with the known addresses. If it is from the user, the time field is updated.
- The Phoneshell log (see Caller ID list below) is analyzed. Any new entry in the log will make Active Messenger update the time field of a corresponding number in the known addresses hash.
- The Knothole [14] log is analyzed. Knothole parses all incoming email messages and writes a log entry if it comes from a pager of the user.

As with most of the Active Messenger data structures, the whole content of the known addresses hash is displayed on the status monitor web page, see chapter 2.7.2.

2.5.4 Other data structures

Other data structures include:

Caller ID list. Each time the user calls up Phoneshell [26] (see also chapter 2.3.9 Wired and cellular phones), the number where she calls from is written to a file. Active Messenger reads this file and pushes the new number onto this list. If the phone number is a number from the Known addresses hash (see chapter 2.5.3), and the associated location is unique, Active Messenger assumes this to be the new location of the user.

User location list. A new location of the user can come from a caller ID entry of Phoneshell (see Caller ID list above), or from other sources described in chapter 2.6.2. Active Messenger keeps track of where the user is, and the user location history list holds this information. It is implemented as a two-dimensional array. If a new location is detected, a new list element consisting of four fields is appended to the history list.

$location_hist[array]	*[array]*	*example*	*description*
$location_hist[**0**]	[**0**]	= “920853001”	**Time** of this entry.
	[**1**]	= “54”	**New location**
	[**2**]	= “920853001”	**New location details and explanations.**
	[**3**]	= “920853994”	**First time** at this location to compute the duration the user has been at a location.

Note that several concurrent algorithms could find the user’s location. In such a case, Active Messenger does not add a new entry, but modifies only the location detail field and updates the time of the entry. Like that, each location entry is associated with a duration. Furthermore, the time without location information is computed easily. The list with these characteristics is visualized on the web page, see chapter 2.7.2.

Error list. Active Messenger keeps a list of all errors that occurred. The elements are strings, beginning with a time stamp. Each new error is pushed as a string onto this list. The error list is relevant for determining if Active Messenger has to exit deliberately (see chapter 2.4.5 Starting and restarting Active Messenger). If there are too many errors, which is equivalent to timed out processes, which is in turn equivalent to “zombie” processes, it is better for Active Messenger to be restarted.

SkyTel™ status list. Active Messenger keeps track of all communication channels, including the SkyTel™ handsets. The SkyTel™ status list keeps track of which message was sent to this device, when, if it arrived, and if yes, when. If two messages in a row don't arrive, Active Messenger disables the sending to SkyTel™ devices. The list is implemented as a two-dimensional array. Each event generates a new list element consisting of four fields:

$Skytel_status[array]	*[array]*	*example*	*description*
$Skytel_status[**0**]	[**0**]	= “yes”	**Status**: Did the message arrive?
	[**1**]	= “54”	**Message number.**
	[**2**]	= “920853001”	**Time** when this message was **sent.**
	[**3**]	= “920853994”	**Time** when this message **arrived**, if at all.

Postponed SkyTel™ messages list. Once Active Messenger has disabled SkyTel™ sending, all messages that should be sent to SkyTel™ are stored to this list. It contains fields about the postponed message as well as about why this specific message was postponed.

Known computers hash stores all computer names that are associated with a location.

There are also about 20 more lists and hashes that Active Messenger generates to keep track of different information.

It is important to see that the content of these lists is lost when Active Messenger exits. Therefore, much effort was put into making the agent stable so that it does not have to be restarted often.

2.6 Modules

What follows is a detailed description of the seven main modules of Active Messenger.

2.6.1 *Load messages*

This module loads all email messages from the files that were stored by a helper script (chapter 2.4.4). Loading a message means parsing the content of a message and keeping parts of the information in program memory, e.g., when it arrived, where it came from, the importance category of the message etc. The body of the message itself is not hold in program memory.

This module executes the following steps:

- Initialize the message list fields properly: old values have to be overwritten.
- Open the message file and store the relevant header fields in the message list.
- If there is a category file, open and read the category and store it in the message list.
- Get the Clues category if we have to compute it or the current value is corrupt.
- Call the NextEvent module. This module will compute the next event and fill it in the message list.
- Look if it is a “Quack” message from Canard. These messages are sent from the Canard system to the user when the pager comes back in range. See also chapter 2.3.2 Canard paging system.
- Look if it is a location message from user. The user can send herself a message with her current location. See chapter 2.6.2 *Find user location*.
- Look if it is a message from a SkyTel™ pager. Such a message would indicate that the pager is able to send and receive messages. If such a message comes in, Active Messenger enables the sending of SkyTel™ messages, regardless of what the status was before.

Figure 23: **Load message module flow chart** shows the flow chart of this module.

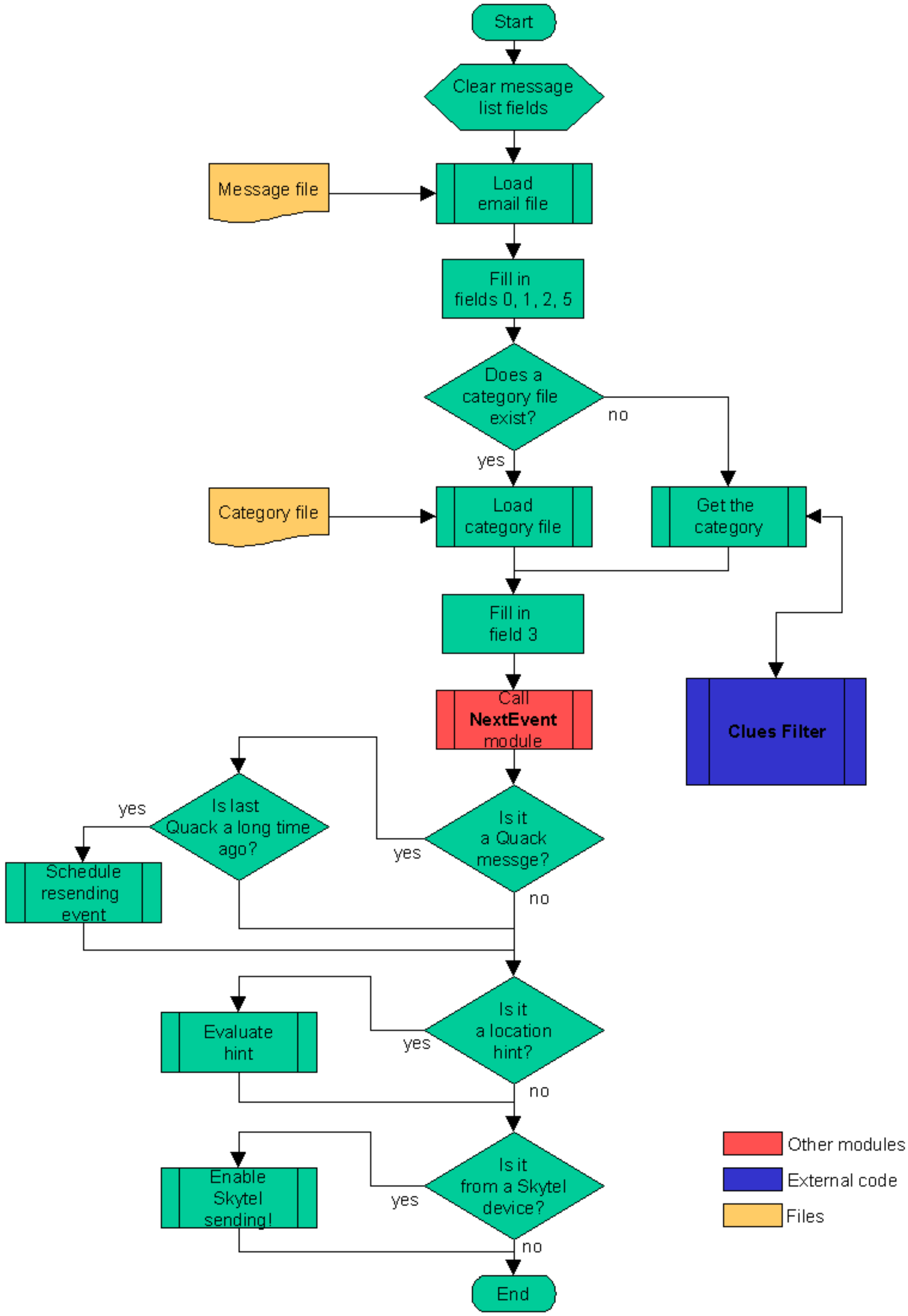


Figure 23: Load message module flow chart

2.6.2 *Find user location*

To optimize the sending of messages, Active Messenger keeps track of where the user is. The user location history list holds this information (see chapter 2.5.4). Currently, Active Messenger has the following possibilities to detect a user's location:

- Where is the user logged in? UNIX *finger* command and the *locate* utility.
- From where did the user call? Caller ID information from Phoneshell [26].
- Direct hint from the user, sent by email.

Finger is a common UNIX server that typically gives back the last login time at a specific workstation. A sample result of a finger request is shown in Figure 24: **Sample finger result**. Different machine types give out different information, but Active Messenger "fingers" users only on one machine, the Media Lab main mail machine.

```
Login name: johndoe                      In real life: John Doe
Office: E99-852, 253-1234                Home phone: 556-3942
Directory: /u/johndoe                    Shell: /bin/tcsh
On since Jun 29 15:35:05                 23 hours Idle Time
   on ttyrb from rariru.media.mit.edu
On since Jul 18 13:58:32
   on ttys1 from claclicIu.media.mit.edu
Plan:
  no plan yet
Projects:
  dunno right now
```

Figure 24: Sample finger result

Active Messenger parses the finger result and tries to detect if the user is currently online. First, it separates out the entries that describe the connections the user has with this machine; all other information is discarded. The agent stored these entries in a list, but relevant are only entries that do not mention that the user is "idle" or when she was connected last. If Active Messenger finds an entry that indicates that the user is online, it parses out from which computer the user is logged in. The name of this computer is compared with the computer names Active Messenger is aware of[30]. If the agent finds a match, the location associated with this computer becomes the current location of the user.

Locate is a command line program similar to *finger*, but it gives back not only the last login time, but also the machine. This program is based on Activity Server [16] that was developed at the Media Lab 1991.

[30] A list of computer names that the user has specified in the preference file is generated when the file is loaded.

Activity Server "fingers" users and performs other tasks to get accurate location information of the user[31]. The answers are plain text like "John was last seen about four and a half hours ago on computer Bingo," or "John is active on Bongo." Because Active Messenger is only interested in information about the user being online, it discards all *locate* answers that do not have "is active" or "remotely logged in" in it. If the user is active, Active Messenger assumes that she is in her office and sets the location accordingly. If the user is logged in remotely, the computer name from where she is logged in is parsed out and compared with the computer names Active Messenger is aware of, similar to the *finger* subroutine described above.

Both location finder algorithms *finger* and *locate,* can be disabled individually. The reason is that *finger* only checks the Media Lab's main mail machine, but not all users use this machine to read their email at all. Checking *locate* has to be disabled sometimes because it is an external command that is not always running properly.

The third possibility for Active Messenger to find out the location of the user is to check a log that is written by Phoneshell. If the user calls up this Media Lab proprietary system that enables access to voicemail, email, calendar, rolodex and other services and utilities, the phone number of the calling party is logged and written to a file after the phone session. When a new entry is made, Active Messenger reads this file and compares the number in it with all numbers it knows from the user preference file, stored in the known addresses hash (see chapter 2.5.3). If there is a match, and the location associated with the number is unique, this location becomes the new location of the user.

The user can also send an email hint to Active Messenger, saying that she is at a certain location. Active Messenger parses all incoming messages for a keyword that would indicate such a hint (see chapter 2.6.1 *Load messages*). So, if a message comes from the user herself, and the first line of a message starts with the string "L: ", the following word is considered as a location. If this location is found in the user preference hash, and therefore is a valid location, it becomes the new location of the user.

For Active Messenger, the user is at a location as long as she doesn't appear on another known location. However, the agent is aware of for how long the user was at a location, and for how long there was no location information at all between two known locations. This is visualized on the status monitor web page, see chapter 2.7.2.

---

[31] The above described finger subroutine would be obsolete if Activity Server would also finger the main mail machine of the Media Lab. Originally, Activity Server was fingering this workstation as well. But somehow, it corrupted this important computer, so this machine was excluded from the list of machines to be fingered.

It is obvious that external systems like GPS (outdoors) or active badges (indoors) would help Active Messenger to find the correct location of the user. See also chapter 4.4.

2.6.3 *Check if messages are read*

An important part of Active Messenger is that it tries to check if the user has read a message. If a message is read, Active Messenger stops sending this message to further channels. Because only a few communication channels provide this information directly (see Table 1: **Characteristics of some communication channels**), Active Messenger also infers from the user's behavior if a message is read. This means that if a message is read is not always clear. It is not a binary value, but rather a likelihood value. Active Messenger keeps track of the "message read likelihood" for each message (see field 6 in Table 12: **Message list structure**). This information gets updated continuously by several subroutines. Its value ranges from 0%, which means "definitively not read," to 100%, "definitively read."

Active Messenger proposes a percentage value for each different kind of back-channel information, see Figure 25: **Different "message read" levels**. However, these values are heuristic and can be modified in the user's preference file. The basic settings are based on certain assumptions about the user's behavior, which may or may not be appropriate for any user. Therefore, the user has to experiment with the settings and try to find her own optimal percentage values.

Because Active Messenger stops sending messages that are read, it is important to set the "message read threshold" properly. All messages above this threshold are regarded as "read by the user," more precisely, "read by user with a sufficiently high enough likelihood." The higher the "message read threshold" level is, the more channels and devices Active Messenger will use until it decides that a message is indeed read by the user. Or even shorter: the higher the "message read threshold" level, the more "obnoxious" Active Messenger will behave.

The most important characteristic of each different kind of back-channel information is if the associated "message read" value is above the "message read threshold" or not. If it is above this threshold, Active Messenger will stop sending this message to further devices. If not, Active Messenger will note that event, but it will keep sending it to other devices. Therefore, by adjusting the "message read threshold," the user defines which back-channel information is enough to stop Active Messenger, and which isn't.

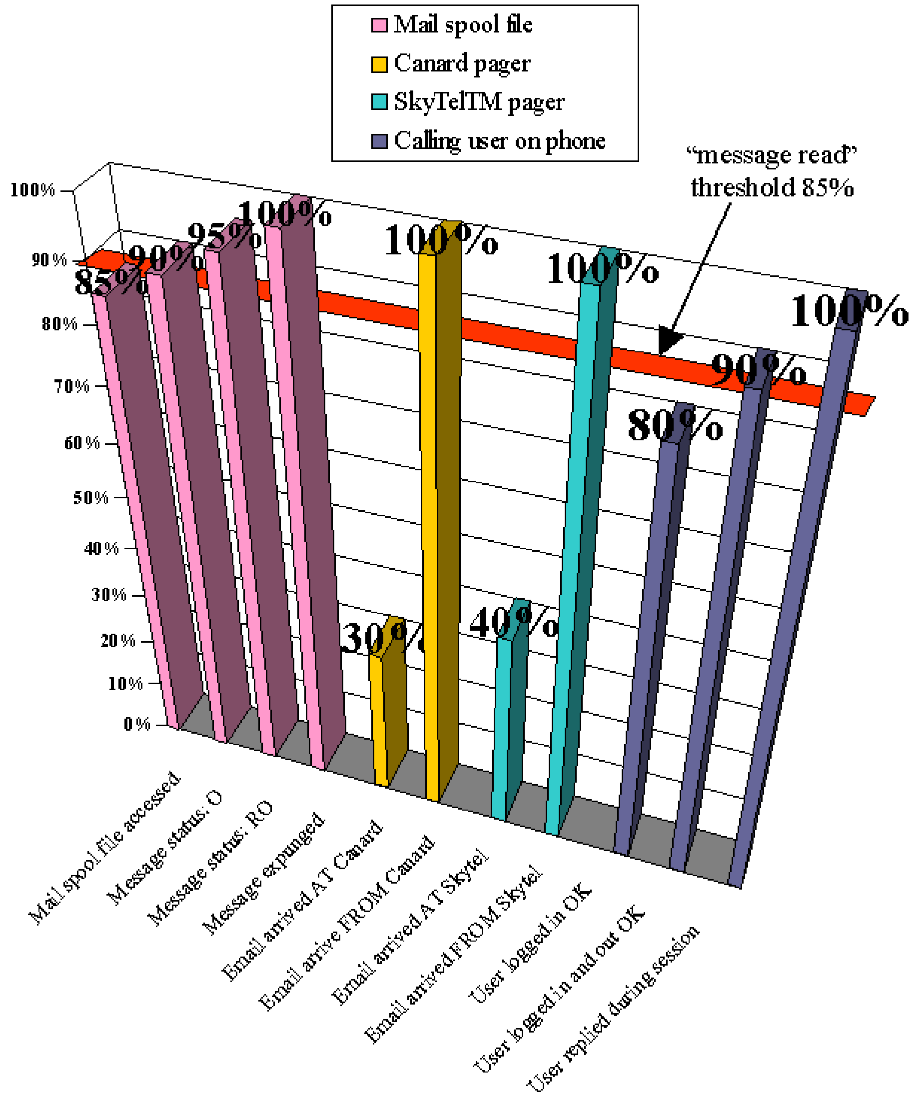


Figure 25: Different "message read" levels

Currently, there are four systems that give feedback about what happened to a message that was sent to it: the mail spool file, the Canard web page, the SkyTel™ web pages, and the module that calls up the user on a phone and synthesizes the message.

2.6.3.1 Mail spool file parsing

As described in chapter 2.3.10, standard mail reader programs like PINE access this mail spool file and modify it automatically. PINE inserts a header field for each message that indicates the status of the message: read, or just opened. The absence of such a status header field means that the message is not even opened. If the message is deleted from the mail spool file, it means it was moved to another folder. Additionally, if the user opens the mail spool file manually, the access time of the file changes.

Active Messenger parses the user's mail spool file[32] and tries to match the messages with the ones in its message list. It does that by comparing the message ID field 0 in the message list with the "Message ID:" header lines found in the mail spool file[33]. After having found a match, it looks for a "Status:" header line and its value, "O" or "RO." If it can't find such a line, the message is not opened yet. All messages that Active Messenger has in its message list should be in the mail spool file originally. If the agent can't find a message in the mail spool file, then it must have been expunged already, which means that the user has deleted it, or moved it to another mail folder.

Active Messenger can derive five different states for each message from the mail spool file:

- **Unread**. The mail spool file was not opened at all, the message can't be read. Active Messenger suggests a "message read" probability of 0%.
- **Access time change**. The mail spool file was accessed, probably by the user, but no further actions may have been taken up to now. It is possible that the user has read a message if she looked at the mail spool file manually. Not many users do that. Access file change effects all messages the same way. Active Messenger suggests a "message read" probability of 85% for all messages in the mail spool file.
- **Message opened**. The user has opened the spool file in a mail reader program, but probably didn't look at the new message body. Active Messenger suggests a "message read" probability of 90% for this message.
- **Message opened and read**. The user has clearly opened the message and must have had a glance at it. Active Messenger suggests a "message read" probability of 95%.
- **Message expunged**. The user has moved the message from the mail spool file either by deleting it, or by moving it to another folder. Active Messenger suggests a "message read" probability of 100%.

2.6.3.2 Canard web page parsing

The Canard web page provides information about which messages were received at the Canard server, and which arrived at the pager. Figure 26: **Sample Canard summary view web page** shows that the first

---

[32] Note that the mail spool file has to be local to the file system Active Messenger is running on, since the directory "/var/spool/mail" is usually not exported.
[33] Matching, e.g., 800 email messages from the mail spool file to 100 emails in the message list is a computationally intensive task that can take several seconds, even on a relatively fast workstation.

message hasn't arrived at the pager yet, but the second one has because there is an additional line "**Received**: noc".

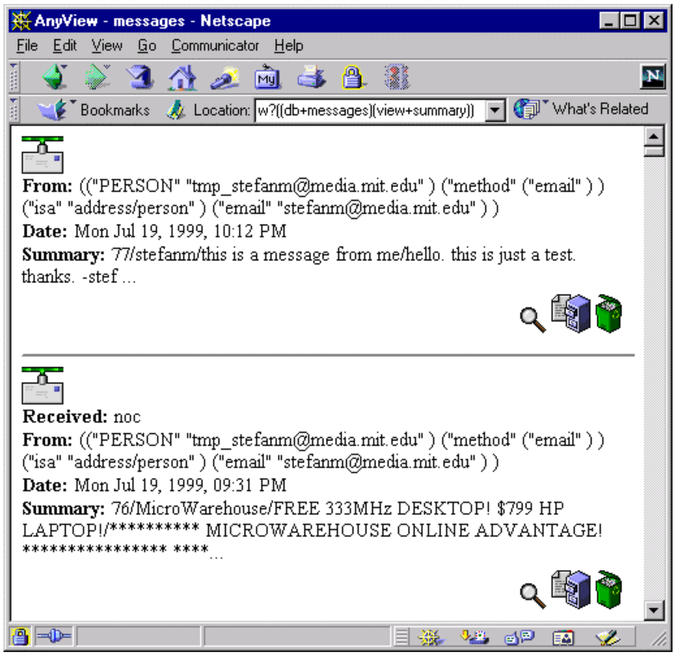


Figure 26: Sample Canard summary view web page

Active Messenger downloads this HTML file and parses it. It can identify each message by looking at the first two characters of the **Summary:** field of each message. This number is the *PagerSequenceNumber* that Knothole as well as Active Messenger adds to each message that will be sent to a pager. Like that, Knothole can identify incoming messages from the pager and forward replies to the right person. Active Messenger generates this number when a message is sent to a mobile device for the first time, and stores it in field 18 of the message list[34].

The agent parses the HTML code and tries to match the messages listed in there with the messages it has sent. If it finds a match, depending on if the line "**Received**: noc" is present or not, it sets field 20 ("arrived at Canard?") to "yes" or "no." However, if a message sent to Canard does not show up on the Canard web page, field 20 is left blank.

Active Messenger checks the Canard web page around four times a minute, but only if there are messages sent to Canard that haven't arrived yet. If the last message that hasn't arrives to Canard is more than 10 minutes ago, Active Messenger assumes that it may have been lost[35]. From then on, the agent checks

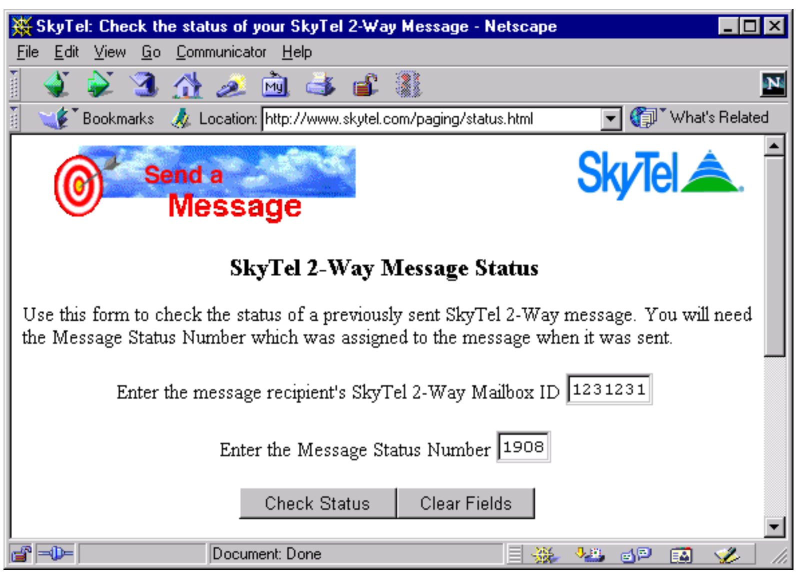
[34] From there on, every time this message is sent to a pager, the same *PagerSequenceNumber* will be used to make identification of the message for the user as well as for Knothole possible.
[35] This can happen easily, since Canard is only buffered for 10 minutes.

only once every 10 minutes to lessen the workload put on the Canard web server with these automatic requests.

2.6.3.3 SkyTel™ web page parsing

The SkyTel™ service has a web site where the user can query the status of a page. This is only possible if it was sent through the web interface, because the requesting party needs to know the ID number of the message. See Figure 27: **Sample SkyTel™ status request** and Figure 28: **Sample SkyTel™ status response**.

Active Messenger has to check each message individually that was sent to SkyTel™ but hasn't arrived yet. For each message, it sends a request to the server and then parses out the HTML code that comes back. The time that comes back is one hour ahead of the local time, so a text string like "Mon Jul 16 16:41:59 1999" has to be transformed to UNIX seconds, and then one hour is added. The result is stored in field 8 and 23 of the message list, and field 22 ("arrived at Skytel") is set to "yes." However, if the message hasn't arrived yet, field 22 is set to "no." If SkyTel™'s answer indicates that there is no such message, field 22 is left empty.



© 1999 SkyTel™

Figure 27: Sample SkyTel™ status request

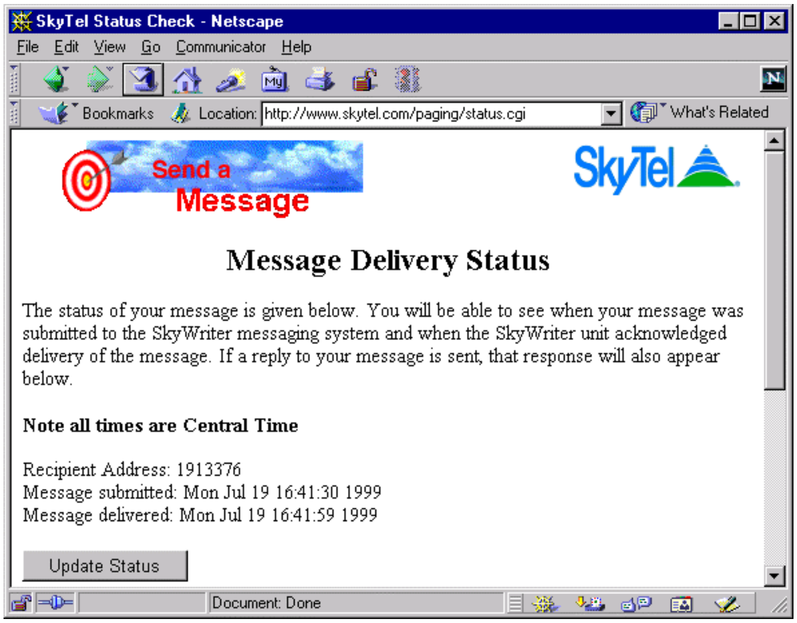


Figure 28: Sample SkyTel™ status response

2.6.3.4 Phoneshell pseudo mail spool file parsing

The module that calls up a user on the phone is based on the same code that Phoneshell [26] uses. It also uses a very similar user interface, as well as the same data structure. Phoneshell reads the messages from a mail spool file. Therefore, the current text-to-speech interface also reads the message to be read to the user from a pseudo mail spool file. So if Active Messenger decides to read the user a message on the phone, it first writes the message to the pseudo mail spool file, containing only one message. After the session, the module may or may not have modified this pseudo mail spool file. Active Messenger then parses the file and looks if a "Status:" field was inserted, and if yes, with which value. This process is very similar to the parsing process of a standard mail spool file, see chapter 2.6.3.1.

Depending on the actions that the user has taken, Active Messenger can differentiate between four different "success levels" for the message delivery:

- The user didn't show any reaction. An answering machine possibly may have answered the call. There is only a slight chance that the user has heard the message. Active Messenger proposes a "message read" likelihood of 0%.
- The user logged in successfully, but didn't show any further reaction. She may or may not have stayed on the phone until the end of the synthesized message, and therefore may or may not have heard it. Active Messenger proposes a "message read" likelihood of 80%.
- The user logged in successfully and quit the system correctly after having heard the full message. Active Messenger proposes a "message read" likelihood of 90%.

- The user logged in and out successfully, as well as pressed some keys during the session, e.g. was navigating within the message. Active Messenger proposes a “message read” likelihood of 100%.

In addition to these four algorithms, Active Messenger uses a more indirect strategy to determine if a message was read. Although the agent knows if a page has arrived at a Canard or SkyWriter™ device, there is no guarantee that the user has read this messages. However, if the user sends back a message from this device, Active Messenger assumes that the user read all messages previously sent to this pager. This strategy works for messages sent to Canard, SkyWriter™, SMS, and ICQ™.

All subroutines that check if a message is read are executed sequentially. However, Active Messenger updates the “message read likelihood” field only if a higher value is found. E.g., the mail spool file reports that the user has opened the message (90%). Later, the Canard web page may say that the very same message has arrived at the pager (30% read). Because the prior value is higher, Active Messenger will note that the message arrived at the Canard pager (message list fields 20 and 21), but the main “message read level” field 6 will keep the higher value.

2.6.4 *Check status of paging systems*

Not all messages that are sent to pagers do arrive there. The pager could be out of range, turned off, or otherwise unavailable. Because the user usually pays for the paging service per message, or even per character, Active Messenger tries to avoid sending message to systems that currently can’t forward them to the handsets properly.

Active Messenger monitors the paging systems and tries to determine if they will forward messages. In general, it does that by comparing the messages that were *sent*, with the message that *arrived* at the handset[36]. After having compared them, the agent enables or disables further sending. A disabled device also is marked on the web interface: the header of its column is shown in bright red, where as an enabled device is shown in white, see also chapter… However, the disabling is implemented differently for SkyTel™ and Canard, because the associated costs for a page are different. There is no disabling mechanism at all for the rest of the paging devices, like SMS or Iridium™ because these systems do not give enough information about if a message arrived or not.

---

[36] Since every page needs a certain time to get to the pager, Active Messenger allows a “grace period” of two minutes for paging systems to deliver a message.

2.6.4.1 Resending Canard messages

Because the Canard service is free for Media Lab personnel, the Active Messenger detects if messages do not arrive at Canard, but otherwise doesn't change its behavior and keeps sending new messages.

However, users do not like missing messages, and Active Messenger tries to avoid that. If a Canard pager comes back in range, the Canard server notifies Active Messenger by sending a special message to the user. This message has "Quack" in the subject line, because it comes from a duck. So whenever a Canard pager comes back in range, Active Messenger will know about it because it detects the "Quack" message. The agent then resends the last ten pages that were sent to Canard, but never arrived there, and were not read otherwise.

However, the Canard system is not very reliable in sending Quack messages. Sometimes, after the pager has been out of range, the handset can't register properly and is in one-way mode only. This causes the Canard system to register the pager several times in a row, and the user gets multiple Quack messages. Even worse, sometimes the Canard system fails and sends Quack messages without a pager coming back in range at all. Therefore, Active Messenger is not just looking for an incoming Quack message, but has to analyze the whole sequence of Quack messages, and tries to determine if the pager really came back in range or not. What the agent tries to detect is a single, isolated Quack message. So resending of unread Canard messages happens only if a Quack arrives, the time difference to the last Quack message is above a certain limit (default is 1.5 minutes), and there are no other Quack messages within a certain time *after* it arrived. Active Messenger does that by scheduling a resending of Canard messages, and waiting until the time has passed without another Quack message arrival. The sequence of Quack messages, the scheduling of a resending, and the resending itself are visualized on the web page, see chapter…

2.6.4.2 Disable sending, postpone, and resend SkyTel™ messages

Pages sent to SkyTel™ are relatively expensive, so the goal for Active Messenger is to send messages to SkyWriter™ handsets only if the system seems to work properly. The agent uses the following strategy to determine that:

- If a message is sent to SkyTel™, Active Messenger waits for two minutes. If the page doesn't arrive after that time, Active Messenger notes this event.
- After that, if a second message is sent to Skytel™ and doesn't arrive either, even after the "grace period" of two minutes, Active Messenger switches to "sending disabled" mode.
- In "sending disabled" mode, Active Messenger postpones all messages that should get sent to SkyTel™, and stores them in a list.

- Even in “postponing” mode, Active Messenger keeps checking regularly, if at least one of the last two messages did arrive. Theoretically, they could arrive even after a very long time, because SkyTel™ claims that their SkyWriter™ service is fully buffered. If the SkyTel™ web page reports that at least one of the two messages has arrived, Active Messenger would switch back immediately to “sending enabled” mode. Additionally, it would resend the postponed messages, if they are not yet read otherwise.
- However, it can happen that a SkyTel™ device comes back in range, but none of the last two messages is being delivered[37]. In this case, the pager could actually receive and send messages, but Active Messenger can’t know it. So if the user is not sure if Active Messenger is aware of that the pager is back in range, she can send a message to herself, be it a reply or a Knothole information request. Regardless of what the current sending mode is, if a message arrives from a SkyWriter™ device, Active Messenger switches to “sending enabled” mode and resends all unread postponed messages.

2.6.5 Schedule next event

The following two modules, *Schedule next event* (this chapter) and *Send message* (next chapter) are closely related. *Schedule next event* is called for each message during the initialization phase of Active Messenger (chapter 2.4.1, see also Figure 21: **Active Messenger flow chart**) and when new messages arrive. Additionally, *Schedule next event* is called immediately after *Send message* (see next chapter).

Active Messenger starts sending a message by first scheduling a “next event” for it. A next event is defined by a time, a device, and a number or address where the message has to be sent to (fields 12, 13, and 14 of the message list). About four times a minute, the agent goes through the message list and checks if one or several of the scheduled “next events” are due. If one is due, the corresponding event is executed, which most often means that the message is sent to a device at a certain number or address. Right after the sending, all the data in the “next event” fields (12 – 14) is transferred to the “last event” fields (10,11), as well as to the “history” fields (17). Then, the next event is computed. If there is a next event, the relevant time, device, and number is put again in the “next event” fields, and so on.

Example: Let’s assume that the user has the following line in her preference file:

```
----
Mapping
important = canard, vpager (13), phone (14), fax (35)
```

This describes the channel sequence for important messages. Roughly, it means that if a message is important, it will be sent to the Canard pager, after that to a voice pager, then to a phone, and then to a fax. The numbers in brackets mean the delay until a device or channel is used.

Let's assume furthermore that she has the following entries in here preference file:

```
----
home
canard  = johndoe@canard.mit.edu, anytime
vpager  = 654-4567, not 0-7
phone   = 423-7755, not M-F 22-8, not SU
fax     = 423-7755, not 2-7:30
```

This means that at home, she has the channels **canard**, **vpager**, **phone**, and **fax** available. For each channel, a number or address is specified, and the time when it is ok to use the device.

The following Figure 29: **Channel sequence example**, shows what happens if a new email message arrives on Monday 6:57am, and the user is currently at home.

[37] Or the SkyWriter™ web page does not report them as delivered.

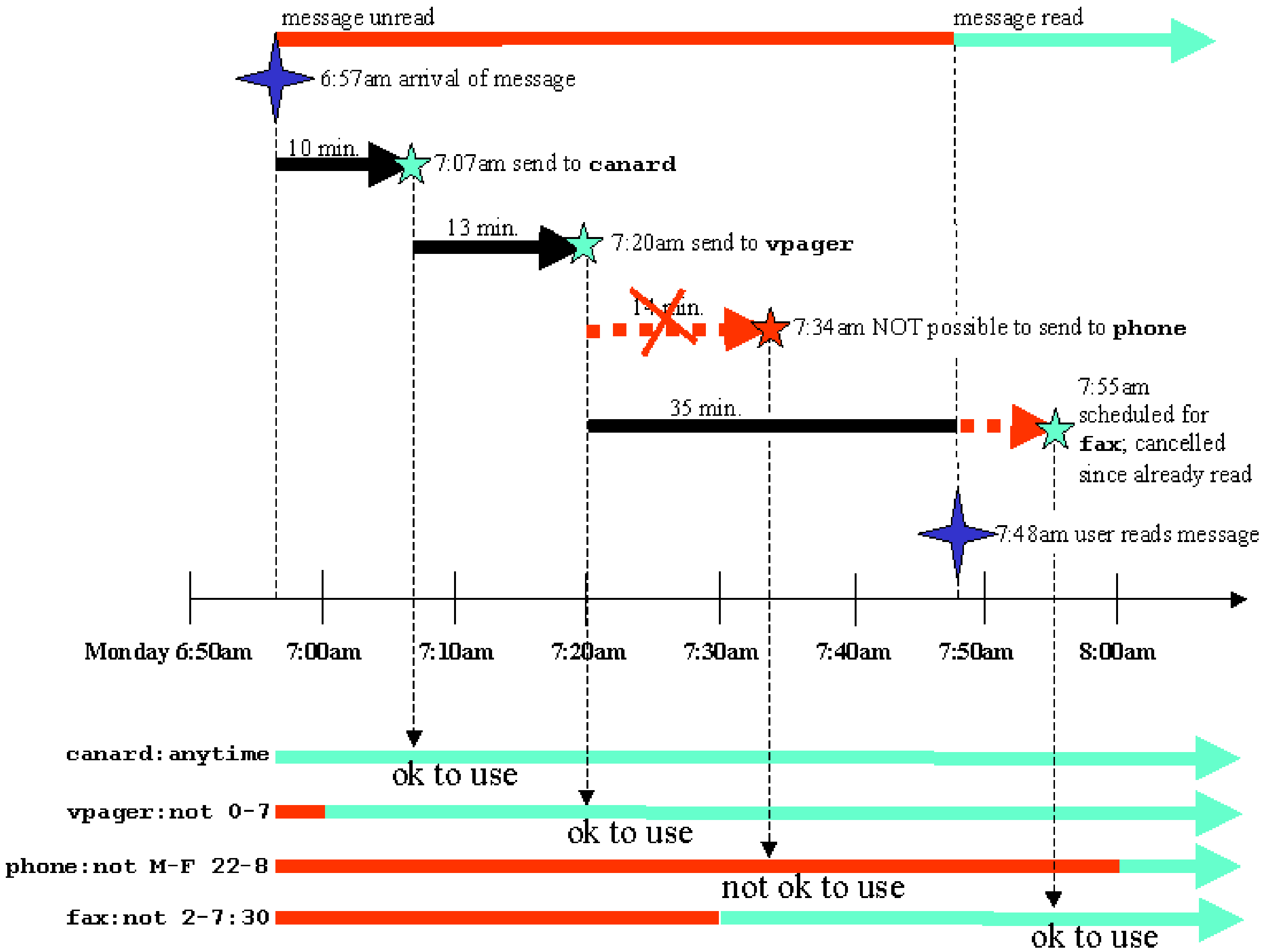


Figure 29: Channel sequence example

First, Active Messenger determines the message category. If the message is **`important`**, the agent looks up what the "last event" was. If there is none, it assumes that the message hasn't been sent anywhere yet. The first channel would be **`canard`** in **`10 minutes`**. (If no time after is specified in brackets, Active Messenger assumes a standard delay of 10 minutes.) Before the agent can schedule this event, it checks if **`canard`** is allowed at that time at that location. The preference file says **`anytime`** is ok for **`canard`** at **`home`**, so Active Messenger schedules this "next event, " and waits.

After 10 minutes, if the user hasn't read the message otherwise, e.g., by reading it from the mail spool file, Active Messenger sends it to the Canard pager. Right after the sending, Active Messenger looks up the next channel, which is **`vpager`** in **`13 minutes`** from then. It's now 7:07am, the time of the sending would be 7:20am, which is a valid time, because the voice pager only wants to get messages after

7:00am. Then the agent waits again and checks all available back channels if the message gets read somehow.

After 13 minutes, if the message's "read level" is still not above the read threshold, Active Messenger calls up the voice pager number and synthesizes the message with the text-to-speech module. Right after that, the agent tries to schedule the next event, which would be **`phone`**. The phone call would be in **`14 minutes`** at 7:34am. Unfortunately, the user does not allow Active Messenger to call her up on the **`phone`** at home from Monday until Friday after 10pm and before 8am. Therefore, this channel is currently not available, and the agent skips it. The next entry would be **`fax`**. It is now 7:20am, the delay for sending faxes is specified by the users as **`35 minutes`**, so Active Messenger schedules a fax sending for 7:55am. Then the agent waits again.

The user, however, happens to log in to her computer and read this email message at 7:48am. The "message read" level raises over the threshold, and the message is regarded as read. Therefore, Active Messenger cancels the fax sending.

This is a very general example. There are many special cases.

E.g., if the channel sequence contains an entry like **`cellphone`**, and the user has no **`cellphone`** entry in the location section where she is right now, the message won't get sent there.

However, it will get sent if there is a **`cellphone`** entry in the Devices section. The Devices section can have the same kind of entries as a normal location section, like **`home`**. The difference is that the devices that are listed in the Devices section are not location specific, like **`cellphone`**. If Active Messenger can't find a device in the location section, then it looks for it in the Devices section. If it finds an entry there, the agent uses this channel. However, entries on the location section have priority over the ones in the more general Devices section.

Furthermore, channels that are currently disabled are skipped automatically. E.g., if Active Messenger has disabled the sending to SkyTel™ devices, messages that are supposed to go there are postponed (see chapter 2.6.4.2).

Active Messenger can modify the initial delay time too. Usually, the user specifies for each channel how long the agent has to wait until it can use this channel. If the user does not specify any time delay, a standard delay of 10 minutes is assumed. However, it does not make sense to wait for the first channel if the user is not active on her computer anyways. Therefore, the delay of the very first device of a channel sequence is reduced gradually if the user is not active. The reduction is proportional to the user's idle time: the longer the user is idle, the shorter the delay until the first channel is used. A message will be sent to the first channel immediately—means, with zero seconds delay—if the user is idle for 15 minutes or more.

2.6.6 Send message

When an event is due, Active Messenger passes the name of the device and its address or number, stored in the fields 12 and 14 of the message list, to specific subroutines. These subroutines that do the actual sending, are the agent's "device drivers."

Depending on the channel or device, a message is sent in different ways.

Messages to **Canard**, **SMS**, and **Iridium**™ devices are sent using the UNIX "mail" command. But first, these messages are piped through NSP, a program that was written 1991 by Jerry Sweet. NSP modifies the format of a message to make it appropriate for pagers. E.g., truncates the message to 2000 characters, and adds the *PagerSequenceNumber* at the beginning. This number is used to identify pages, see chapter 2.6.3.2. It is usually generated by NSP, which is looking it up from a file that is stored in the user's directory, and increases the number at the same time. However, only the first time a message gets sent to a pager, a new PagerSequenceNumber is generated. This number is stored in field 18 of the message list. If the same message gets sent to another pager, the same number is used again to make it easier for the user to recognize the pages. NSP also removes included text, as well as certain kinds of signatures. By sending a message through NSP, it gets also stored in a special directory that Knothole uses to generate replies. If the message was sent successfully, the subroutine gives back the time of the sending, which is stored in the specific fields of the message list.

Messages to **SkyWriter™** handsets are sent through the SkyTel™ web interface to obtain the message specific ID number, see chapter 2.6.3.3. Without this number, it is not possible to query afterwards the status of a message. But first, the original message also gets routed through a special version of NSP that does the entire pager specific formatting. For SkyTel™, the message length gets limited to 300 characters. The message is then sent to the web interface. Sending a message through a CGI script of a web server actually means passing the whole message content as part of a URL. Therefore, all special characters of a message have to be replaced to be valid characters of a URL.

Messages to **fax machines** are passed to a HylaFAX server. This server dials up the user's fax machine, and deals with busy lines and other fax specific problems. How many times the server retries can be specified, as well as if a cover page should be added, see Figure 17: **Sample fax, created by Active**

Messenger, including a cover sheet. Although the faxing process is completely managed by the HylaFAX server, it still passes back a four-digit ID number in case the user wishes to check the fax specific log files.

Messages to **voice pager and wired or cellular phones** are all passed to an external script that is based on Phoneshell [26] code. This script triggers also the text-to-speech synthesis, using a DECtalk unit. First, the message to be read to the user has to be stored in a special format, so that it looks like a mail spool file. Then, the script is called. The only difference between phones and voice pagers is that for the voice pager, the system does not wait for user input. However, for voice pagers, it has to wait until the announcements of the voice pager's voice mail system are over, and then the DECtalk unit synthesizes the email message body immediately. No interaction is required. When the script is executed to call up phones, the user can interact with the system similarly to the original Phoneshell, see chapter 2.3.8 Voice pager by Motorola and chapter 2.3.9 Wired and cellular phones. After the session, the script may or may not have modified the message file, depending on the user's reactions during the session. Active Messenger analyzes the file immediately after the session, as described in chapter 2.6.3.4 Phoneshell pseudo mail spool file parsing.

Immediately after sending a message, the next event is scheduled by calling the module *Schedule next event*, as described in chapter 2.6.5.

2.6.7 Write web file

Originally, it was planned to let the Active Messenger regularly dump all internal data to a file, similar to a "core dump." However, this idea became obsolete because all data is written to a web accessible web page, four times a minute.

The last module of the agent writes this status page. Although the web page itself may look like generated by a CGI script, it is just a file that simply is written four times a minute, at the end of every main loop of the Active Messenger. The file header includes a statement that makes the web browser reload the web page regularly. The default refresh rate is once every two minutes, but can be specified by the user.

It has to be written to a directory that is world-accessible. Usually, it is in the user's *public_html* or *WWW* directory. It reflects the current status of the Active Messenger, displaying all internal variables in a graphical way. A detailed description of the content of the web page follows in chapter 2.7.2.

Currently this web page is publicly accessible. Because it reveals a plethora of information about the user, including all her phone numbers, email addresses, but also her location over time, as well as the sender of all email addresses she has received, being publicly accessible is only acceptable during the development phase of the agent. However, the web page is not interactive, and Active Messenger's functionality does in no way depend on this web page. The web page is only here so that the user can check if the Active Messenger does what it is expected to do.

2.7 User interfaces

The Active Messenger is an agent program that does not require an extensive user interface. However, the user has to specify her needs, and quite often, she wants to know what her agent is doing right now.

The user preference file (chapter 2.7.1) is the only file the user has to edit to run Active Messenger. All necessary information can be specified there.

The Active Messenger web page (chapter 2.7.2) visualizes the current status of the agent, as well as what happened in the past and what is planned for the future.

The Log file (chapter 2.7.3) is for debugging purposes only. In this file, most important events are logged in detail.

2.7.1 Preference file

The user preference file is the only file that the user has to edit in order to run Active Messenger. A simple one was shown earlier on page 29, Figure 7: **Sample user preference file**. This small file provides enough information to start up the agent because Active Messenger uses default values for missing preference file entries. However, if the user wants to control all variables, she can specify them as additional entries. An extensive user preference file is shown in Figure 30: **Extensive sample user preference file**.

The user preference file is an ASCII file that has to be stored in the users home directory. Each time the user changes the preferences file, Active Messenger reloads it and updates all values. Therefore, the user can make major changes in the Active Messenger's behavior without restarting the agent itself and losing the "knowledge" the agent has obtained about past events.

The generic user preference file, like the one shown in Figure 30: **Extensive sample user preference file**, should be self-explanatory. It contains examples and instructions about how to use and modify the file.

Although a preference file can have many lines, the structure of the file is simple. It consists of several sections. Each section of the preference file is loaded into one element of the main hash, as described in chapter 2.5.2. Possible sections are **Mapping**, **Locations**, **Devices**, and **Files**:

- **Mapping**. Each importance category is assigned a sequence of channels and devices.
- **Locations**. List of all possible user locations.
- **Devices**. List of all devices that are not yet defined in a specific location.
- **Files**. All user specific path names and other preferences.
- All other sections are locations, like **home** or **office**, where location-specific devices can be defined.

2.7.1.1 Mapping

This is the section that provides Active Messenger with all necessary information about how to proceed with a message of a certain category. Each line in the Mapping section represents a category. E.g., the line

```
important = canard (5), skytel (12), fax (30)
```

lets Active Messenger use the following channel sequence after an **`important`** message comes in:

Message comes in,

→ wait **5** minutes,

→ send to **Canard** pager,

→ wait **12** minutes,

→ send to **SkyWriter™** device,

→ wait **30** minutes,

→ send to **fax** machine.

At any time, the user reading the message terminates the channel sequence. A detailed example for such a channel sequence is shown in chapter 2.6.5 *Schedule next event*.

2.7.1.2 Locations

This is the section where the user lists the names of the locations that will be specified later in detail. E.g., the line

```
locations = home, office, parents
```

prepares Active Messenger that there will be a **home** section, an **office** section, as well as a **parents** section. This is necessary because each location is a section on its own, and Active Messenger can't know the location names in advance.

2.7.1.3 Sections like *home, office, parents*

In the sections whose names were specified in the Locations section (see chapter 2.7.1.2), location-specific devices and channels are listed, e.g., **phone** and **fax**. If a channel sequence lets the Active Messenger send a message to a **phone**, the agent has to determine first the location of the user. If the current location is, e.g., **home**, Active Messenger looks up the phone number in the **home** section. The lines

```
-----
home
phone = 613 121-3453
```

tell the agent that it can call the user at home at 613 121-3453.

2.7.1.4 Devices

This is the section that lists all channels and devices that are not yet defined at a specific location (see chapter 2.7.1.3). Here the user can also define default devices and their numbers or addresses. Portable or mobile devices such as **cellphone** or **vpager** are specified here, whereas location based devices like **phone** and **fax** are better specified in a location section. E.g., the line

```
canard = johndoe@canard.mit.edu
```

lets Active Messenger know that the user has a Canard pager, as well as its address. It also means that this device can be used in any location and anytime, unless the user limits it in a location section.

2.7.1.5 Files

This is the section that contains all user specific path names and other preferences. For most of them, Active Messenger has default values, so missing entries would not make the agent crash. E.g., the line

```
locate_user = 1
```

makes Active Messenger locate the user regularly, which is also default value. `0` would disable locating her. Here is a list of all variables that can be specified:

Table 15: Preference file default values

Variable	*Preset or example value*	*Comments*
userdir	/u/johndoe/	The user's home directory.
AM_directory	/u/johndoe/AM/	The user's Active Messenger directory.
preferencefile	/u/johndoe/AM/Preferences	This file here.
dif_file	/u/johndoe/AM/Log	The file where new events are logged.
dir_for_originals	/u/johndoe/AM/Messages/	Directory where incoming messages are stored, see 2.4.4.
SequenceNumber	/u/johndoe/AM/SequenceNumber	Counter for these messages.
timer_log	/u/johndoe/AM/timerlog	Logs timings of all external calls, see 2.4.5.
pagerlogfile	/u/johndoe/.PagerLogfile	Log written by Knothole, see 1.3 and 2.5.3.
procmailrc	/u/johndoe/.procmailrc	Resource file for procmail, where you also specify your categories, see 1.2.1.
nsp	/u/active_messenger/nsp_am	External program that does formatting for Canard, Skytel, SMS. See 2.6.6.
canard_username	johndoe	Canard username.
canard_password	thesecretpassword	Canard password.
fax_username	John J W Doe	Name that will appear on faxes sent by Active Messenger to you, see Figure 17.
max_stored_messages	99	Number of messages that are expected to be stored in .AM_Messages. The program that stores the messages, AM_Store_Messages, has to have the same number. You specify it in your .procmailrc.am file as the argument to AM_Store_Messages.
locate_timeout	120	If a "locate" call doesn't return after these seconds, it is timed out, see 2.6.2.
finger_timeout	120	If a "finger" call doesn't return after these seconds, it is timed out.
skytel_timeout	120	If a Lynx call to the SkyTel™ web page doesn't return after these seconds, it is timed out, see 2.6.3.3.
canard_timeout	120	If a Lynx call to the Canard web page doesn't return after these seconds, it is timed out, see 2.6.3.2.
fax_timeout	120	If a "fax send" call doesn't return after these seconds, it is timed out.
clues_timeout	120	If a "clues_filter" call doesn't return after these seconds, it is timed out, see 1.2.2 and 2.6.1.
vpager_timeout	240	If a vpager call doesn't return after these seconds, it is timed out.
timer_log_enable	1	Enables the log file that registers the length of all external calls.
allowed_idle	900	If no new location information for this amount of seconds, AM adds another entry to the location history (even if it is at the same location as before), see 2.6.2.
init_sending_damp	900	If the user is idle for more than this amount of seconds, the delay to send a message to the first channel is reduced to zero seconds. Idle times less than this amount of seconds is scaled down proportionally: the longer the user is idle, the shorter the delay until the first channel is used. See 2.6.5.
skytel_graceperiod	120	If a message sent to SkyTel™ hasn't arrived after this time, the SkyTel™ status changes to "non-receiving." See also 2.6.4.2.
canard_graceperiod	120	If the last message sent to Canard doesn't arrive within this amount of seconds, set the "Canard" heading on web page to red.

finger_ml	1	Enable (1) or disable (0) fingering ML, see 2.6.2.
locate_user	1	Enable (1) or disable (0) locating user, see 2.6.2.
max_clues_errors	5	If there are more than 5 clues status errors, AM terminates, see 2.4.5.
max_timeout_errors	5	If there are more than 10 timeouts, AM terminates, see 2.4.5.
SINGLE_QUACK_BACKW	90	Minimum distance in seconds between current and last Quack to accept it as a single one, see 2.6.4.1.
SINGLE_QUACK_FORW	0	Minimum distance in seconds to wait after current Quack to accept it as a single one, see 2.6.4.1.
Actions upon certain events		
Canard_on	canard_in_range	Program that is executed if Canard comes back in range.
Canard_on_timeout	30	If the external call "canard_on" doesn't return after these seconds, it is timed out.
Skytel_on	stop_something	Program that is executed if SkyTel™ comes back in range.
Skytel_on_timeout	30	If the external call "skytel_on" doesn't return after these seconds, it is timed out.
Web page stuff		
web_status_page	/u/johndoe/www/AmPage.html	The file name of the AM web status page.
Show_messages	20	How many messages should be displayed.
Smallest_font_size	1	Smallest font size that AM can use for the AM status web page.
Refresh_rate	120	Refresh rate in seconds for the AM status web page.
Coloring	readstatus	Options are 'readstatus' or 'category'.
Color_scheme	red	Options are 'blue-green' or 'red'.
Values for determining the "message read" likelihood, see 2.6.3		
READ_NO_SPOOL	100	Value that is set if a message is expunged from the spool file, see 2.6.3.1.
READ_MSG_ARRIVED	100	Value that is set if another message arrived from this device.
READ_RO	95	Value that is set if a message is marked "read" in the spool file, see 2.6.3.1.
READ_O	90	Value that is set if a message is marked "opened" in the spool file, see 2.6.3.1.
READ_SPOOL_AC	85	Value that is set if the mail spool file is accessed, see 2.6.3.1.
READ_ARRIV_SKYTEL	20	Value that is set if the message arrived to Skytel, see 2.6.3.3.
READ_ARRIV_CANARD	15	Value that is set if the message arrived to Canard, see 2.6.3.2.
HEARD_RESPONDED	100	Value that is set after calling up user, and pseudo spool file has "Status: RO." See 2.6.3.4.
HEARD_IT	90	Value that is set after calling up user, and pseudo spool file has "Status: O." See 2.6.3.4.
HEARD_NO_REACTION	80	Value that is set after calling up user, and pseudo spool file has still no status line. See 2.6.3.4.
HEARD_NO_READABLE	30	Value that is set after calling up user, and pseudo spool file is not readable anymore. See 2.6.3.4.
READ_THRESHOLD	95	All messages above this threshold are being regarded as read, and no more actions are taken to use further channels. Useful thresholds are 80 to 100. 100 means in this example that only messages are read which are expunged from the mail spool file, or if another message has arrived from the pager it was successfully sent to. See also 2.6.3.
PINE_PERCENTAGE	92	An open PINE session accesses the spool file after 2.5 minutes automatically if new messages have arrived. Unfortunately, it's not exactly 150 seconds. This variable defines how exact the time interval has to be. E.g., "92" means that the accuracy of the interval being 150 seconds (or a multiple of it) has to be at least 92%. Usually, accuracy is about 98%. If

		your system is busy and does not detect PINE mail spool file accesses successfully, decrease this level. See 2.6.3.1.
Verbose level of the screen output		
quiet	0	If quiet is true, no comments are printed to screen.
quiet1	1	Quiet for sub GetNextEvent. See 2.6.5.
quiet2	1	Quiet for sub LoadMessages. See 2.6.1.
quiet3	0	Quiet for sub LoadPreferences. See 2.5.2.
quiet4	1	Quiet for sub IsItOk.
quiet5	1	Quiet for Show Message list after initialization.
quiet6	1	Quiet for Show READ status of the message list.
quiet7	0	Quiet for sub Read_MessageFromKnownAddress.
quiet8	1	Quiet for sub that matches messages list with mail spool file.
quiet9	1	Quiet for sub CheckForKnownAddresses. See 2.5.3.
quiet10	1	Quiet for sub CheckSkytel. See 2.6.3.3
quiet11	1	Quiet for sub CheckCanard. See 2.6.3.2.
quiet12	1	Quiet for sub CheckCluesCategories.
quiet13	1	Quiet for sub Quack lists.
quiet14	1	Quiet for sub CheckFingerML. See 2.6.2.
quiet15	1	Quiet for sub CheckLocate. See 2.6.2.
quiet16	0	Quiet for sub GetCategoryOrder.
quiet17	0	Quiet for sub GetSkytelStatus.
quiet18	1	Quiet for sub CheckPagerLogfile. See see 1.3 and 2.5.3.

What follows on the next three pages is Figure 30: **Extensive sample user preference file**. It shows a preference file of a user that specifies every internal variable.

```
                           ########################################
                           # Active Messenger user preference file #
                           ########################################

# In this file you define how Active Messenger (AM) works for you. There are several sections.
# Each section starts with "----". For each section, there can be entries. Each entry is a key
# value pair, one line, separated by a "=". The value can consist of several entries, separated
# by a comma. Each line starting with a "#" is ignored (comment). Everything after a "#" is
# also regarded as comment and therefore ignored.

--------------
Mapping

# Here you map your categories to your devices: Each incoming message is given a category by
# the Clues filtering program. You have defined these categories individually in your
# .procmailrc file. It makes sense to use the same categories for AM.
#
# If a new message comes in, Clues filter assigns a category to it. Then AM sends it to the
# first channel of this category. If you do not read the message, AM sends it to the next
# channel in the category, etc., until there are no more channels available for this category.
#
# The numbers in brackets specify the delay before the channel is used. If you put no brackets,
# AM takes a standard delay of 10 minutes before it uses this channel.

personal       = canard
timely         = canard (8), skytel
important      = canard (5), skytel (12), fax (30)
veryimportant  = canard (0), vpager (5), sms (25), iridium (30)
ultraimportant = canard (0), sms (5) , iridium, phone, cellphone

--------------
Locations

locations      = home, office, parents

# Before AM sends a message to a certain channel (see above), it tries to infer where the
# user may be right now.
#
# If AM comes to the conclusion that you must be at "home," it looks up in the section "home"
# if the channel you want to use is allowed: you can specify that a certain device should ONLY
# be used at certain times, or NOT be used at certain times. This is especially useful for
# phone and fax.
#
# Furthermore, you can tell AM to use location specific phone or fax numbers or email
# addresses. E.g., if you tell AM to send something to your phone, the phone number is
# different if you’re in the "office" or at "home."
#
# Each location is a section on its own, means, starts with "----".
#
# AM determines a location by comparing (1) the computer names it knows with information from
# "locate" and "finger," and (2) the phone numbers it knows with the Phoneshell caller ID log.
#
# Each entry has a device (phone), a number (345345), and a time (8-17), possibly several
# ones separated by commas. Time can be also just a day, or several consecutive days.
#
# Days are: M, T, W, R, F, S, SU.
# Several consecutive days are, e.g., M-F, SU-T (no spaces)
# Start and end times are, e.g., 17-18:30 (no spaces, no seconds), ranging from 00:00 to 23:59,
# or "anytime," or "never."
#
# Examples:
#     "S-M 5:00-17:00"  : Saturday until Monday 5am until 5pm
#     "20:20-07"        : 8:20pm until 7am
#     "SU 09-10:09      : Sunday 9am until 10:09am
#     "not T-SU"        : not on Tuesday until Sunday
#     "F"               : on Fridays
#     "never"           : never
# "Not" means that every other time is possible.
# An entry in the location section overrides the default device entry, e.g., a phone number at
# "location = home" overrides the default "phone" number. If a device is missing in a location,
# the default device is OK to use all the time.
```

```
-------------
home
phone          = 568-5031, not 22-9
fax            = 396-8499 (no cover), not 23-8
computer       = dialup
-------------
office
phone          = 553-5386, anytime
fax            = 495-5244, M-F 10-23
computer       = magama, klingklong
-------------
parents
phone          = 01141 62 234-4325, not 17-2  # this is Swiss time 23:00 - 8:00
fax            = 01141 62 293-4367, anytime
computer       = wcom.net

-------------
Devices
# This section is not mandatory. Here you can define default devices and their numbers or
# addresses. If you have location specific numbers or addresses, you better enter them in the
# locations section. (Keep in mind that if you list a device here, it can be used at ANY
# location you are, unless you limit its use in the location sections.)
canard         = johndoe@canard.mit.edu
skytel         = 4963278@skytel.com
sms            = 16178183421@omnipoint.net
iridium        = 881536082914@iridium.com
-------------
Files
# Here are file names and other basic configurations. AM has defaults for each of them.
userdir           = /u/johndoe/                  # The user's home directory
AM_directory      = /u/johndoe/AM/               # The user's Active Messenger directory
preferencefile    = /u/johndoe/AM/Preferences    # This file here
dif_file          = /u/johndoe/AM/Log            # The file where new events are logged
dir_for_originals = /u/johndoe/AM/Messages/      # Directory where incoming messages are stored
SequenceNumber    = /u/johndoe/AM/SequenceNumber # Counter for these messages
timer_log         = /u/johndoe/AM/timerlog       # Logs timings of all external calls
pagerlogfile      = /u/johndoe/.PagerLogfile     # Log written by Knothole
procmailrc        = /u/johndoe/.procmailrc       # Where you specify your categories
nsp               = /u/active_messenger/nsp_am   # Does formatting for Canard, Skytel, SMS
max_stored_messages = 99# Number of messages that are expected to be stored in .AM_Messages.
                        # The program that stores the messages, AM_Store_Messages, has to have
                        # the same number. You specify it in your .procmailrc.am file as the
                        # argument to AM_Store_Messages.
canard_username   = johndoe
canard_password   = thesecretpassword
fax_username      = John J W Doe
locate_timeout    = 120 # If a "locate" call doesn't return after these secs, it is timed out
finger_timeout    = 120 # If a "finger" call doesn't return after these secs, it is timed out
skytel_timeout    = 120 # If a Lynx call to the Skytel web page doesn't return after these
                        # seconds, it is timed out
canard_timeout    = 120 # If a Lynx call to the Canard web page doesn't return after these
                        # seconds, it is timed out
fax_timeout       = 120 # If a "fax send" call doesn't return after these secs, it is timed out
clues_timeout     = 120 # If a "clues_filter" call doesn't return after these secs, it is
                        # timed out
vpager_timeout    = 240 # If a vpager call doesn't return after these seconds, it is timed out
timer_log_enable  = 1   # Enables the log that registers the length of all external calls
allowed_idle      = 900 # If no new location information for this amount of seconds, AM adds
                        # another entry to the location history (even if it is at the same
                        # location as before).
init_sending_damp = 900 # If user is idle for more than this amount of seconds, the delay to
                        # send a message to the first channel becomes zero. Idle times less
                        # than that are scaled, so that no idle time means the whole delay.
skytel_graceperiod= 120 # If a message sent to Skytel hasn't arrived after this time,
                        # the Skytel status changes to "non-receiving"
canard_graceperiod= 120 # If last message sent to Canard doesn't arrive within this time,
```

```
                        # set the "Canard" heading on web page to red
finger_ml         = 1   # Enable (1) or disable (0) fingering ML
locate_user       = 1   # Enable (1) or disable (0) locating user
max_clues_errors  = 5   # If there are more than 5 clues status errors, AM terminates
max_timeout_errors= 5   # If there are more than 10 timeouts, AM terminates
SINGLE_QUACK_BACKW= 90  # Minimum distance in seconds between current and last Quack to accept
                        # it as a single one
SINGLE_QUACK_FORW = 0   # Minimum distance in seconds to wait after current Quack to accept it
                        # as a single one

# Actions upon certain events

canard_on         = canard_in_range
canard_on_timeout = 30  # If canard_on doesn’t return after these secs, it is timed out
skytel_on         = stop_something
skytel_on_timeout = 30  # If skytel_on doesn’t return after these secs, it is timed out

# Web page stuff

web_status_page   = /u/johndoe/public_html/AmStatusWebPage.html
show_messages     = 20   # How many messages should be displayed
smallest_font_size= 1    # Smallest font size AM can use for the AM status web page
refresh_rate      = 120  # Refresh rate in seconds for the AM status web page
coloring          = readstatus  # Options are 'readstatus' or 'category'
color_scheme      = red  # Options are 'blue-green' or 'red'

# Values for determining the "message read" likelihood

READ_NO_SPOOL     = 100  # If a message is expunged from the spool file
READ_MSG_ARRIVED  = 100  # If another message arrived from this device
READ_RO           =  95  # If a message is marked "read" in the spool file
READ_O            =  90  # If a message is marked "opened" in the spool file
READ_SPOOL_AC     =  85  # If the mail spool file is accessed
READ_ARRIV_SKYTEL =  20  # If the message arrived to Skytel
READ_ARRIV_CANARD =  15  # If the message arrived to Canard
HEARD_RESPONDED   = 100  # After calling up user, pseudo spool file has "Status: RO"
HEARD_IT          =  90  # after calling up user, pseudo spool file has "Status: O"
HEARD_NO_REACTION =  80  # after calling up user, pseudo spool file has still no status line
HEARD_NO_READABLE =  30  # after calling up user, pseudo spool file is not readable anymore
READ_THRESHOLD    =  95  # All messages above this threshold are being regarded as read, and no
                         # more actions are taken to use further channels. It makes sense to
                         # have a threshold of 80 to 100. 100 means in this example that only
                         # messages are read which are expunged from the mail spool file.
PINE_PERCENTAGE   =  92  # An open PINE session accesses the spool file after 2.5 minutes
                         # automatically if new messages have arrived. Unfortunately, it's not
                         # exactly 150 secs. This variable defines how exact the time interval
                         # has to be. E.g., "92" means that the accuracy of the interval being
                         # 150 secs (or a multiple of it) has to be at least 92%. Usually,
                         # accuracy is about 98%. If your system is busy, decrease this level.

# Here you can define how verbose the screen output of AM should be

quiet             = 0    # If quiet is true, no comments are printed to screen
quiet1            = 1    # Quiet for sub GetNextEvent
quiet2            = 1    # Quiet for sub LoadMessages
quiet3            = 0    # Quiet for sub LoadPreferences
quiet4            = 1    # Quiet for sub IsItOk
quiet5            = 1    # Quiet for Show Message list after initialization
quiet6            = 1    # Quiet for Show READ status of the message list
quiet7            = 0    # Quiet for sub Read_MessageFromKnownAddress
quiet8            = 1    # Quiet for sub that matches messages list with mail spool file
                         # There are many more variables to quiet down AM screen output.

--------------
Ignore

# Here you can specify the names of people AM should ignore.

Junk
Dorkman@dork.com
sally@hotmail.com
```

Figure 30: Extensive sample user preference file

2.7.2 Web page

The Active Messenger status monitor web page reflects the current status of the agent. It also shows what happened in the past, and what events are planned for the near future.

The web page is updated continuously, the default is every two minutes. However, the actual HTML file is written more often, typically four times a minute[38]. Therefore, if the user thinks that the web page shown is not the most recent one, she can hit the browser's "reload" or "refresh" button, and the web page will be updated from the most recent HTML file.

The HTML code contains currently neither links, nor Java, nor JavaScript elements.

The page consists of several tables, which don't fit all in a browser window. However, the main idea was to put the most important and most recent information in the upper left corner of the web page. Figure 31: **Status page, upper left side**, shows this part.

On top, right after the logo, there is information about the current time, when the agent was started, for how long it is running now, and on which machine it is running. Furthermore, it displays the user name, its current location, for how long she has been there, and for how long Active Messenger has no location information.

The main element on the web page is the message list table. It is wider and longer than a normal web browser window; however, the most important information is in the upper left corner. Each row of the table represents a message, the most recent one on top to the list. For each message, 30 parameters are displayed in 30 columns. These parameters are taken directly from the message list and its fields, see chapter 2.5.1. The width of the whole table varies, depending on past events.

The rows of the main table are colored. The coloring scheme is user configurable. There are two coloring options:

- The lower the message read level is, the darker the color.
- The higher the importance of the message is, the darker the color.

This reflects two different kinds of user preferences: the first focuses on the "unread-ness" of the message, the second one on the importance of the message. In the first case, the message color changes

[38] Depending on the user preferences, the size of the HTML file can get up to 80 Kbytes. Depending on the network connection, the download time can be significantly longer than the interval between two updates, which is typically around 15 seconds.

whenever Active Messenger thinks the "message read" level has changed, from dark red (unread) to white (read). Therefore, unread messages stick out. In the second case, important messages stick out. Because the importance level never changes the color of each row does not change over time.

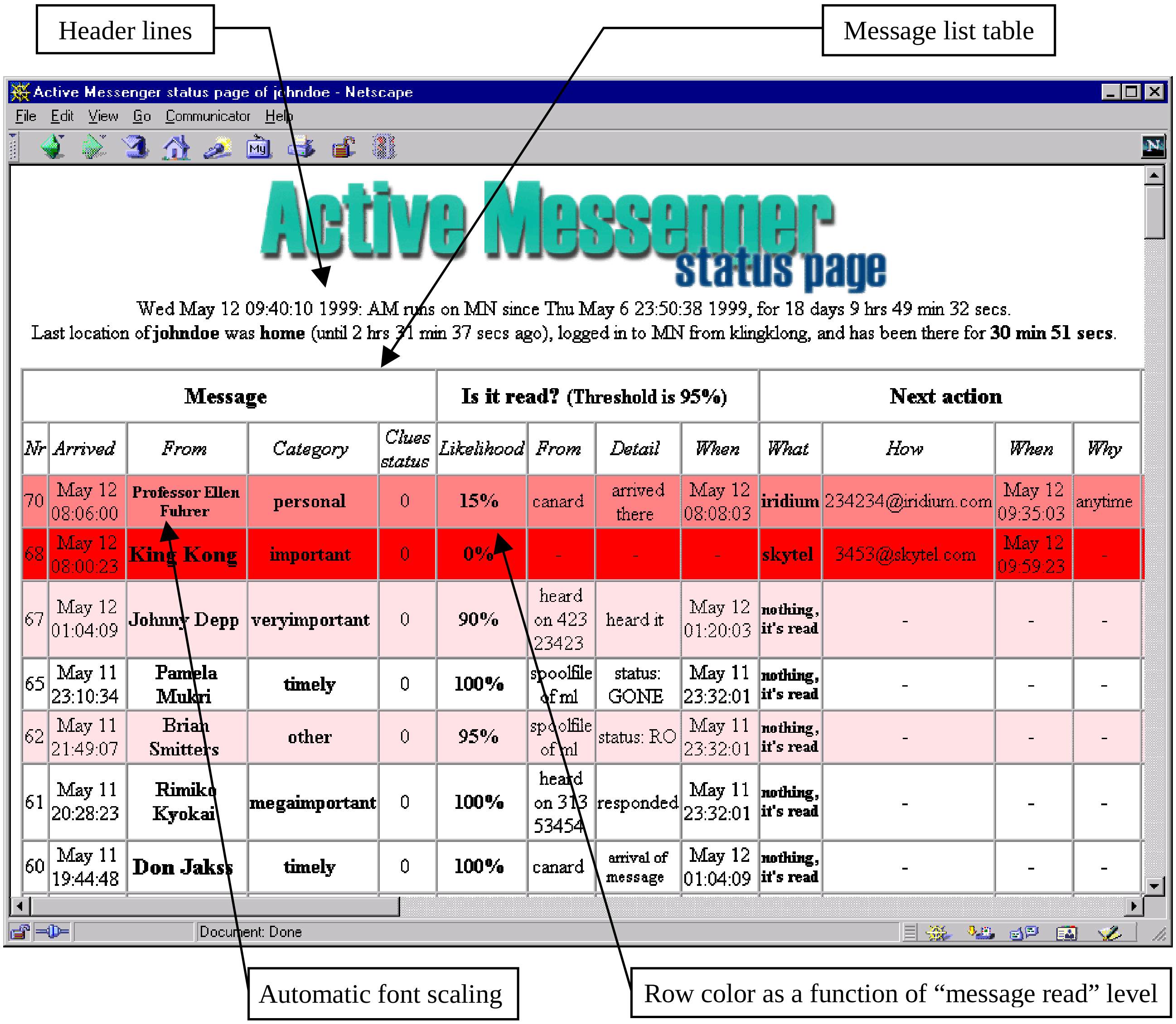


Figure 31: Status page, upper left side

The middle part of the message list table, shown in Figure 32: **Status page, middle part**, lists the history of a message. It is the same data that is stored in the history field of the message list, see chapter 2.5.1, and Table 12: **Message list structure**.

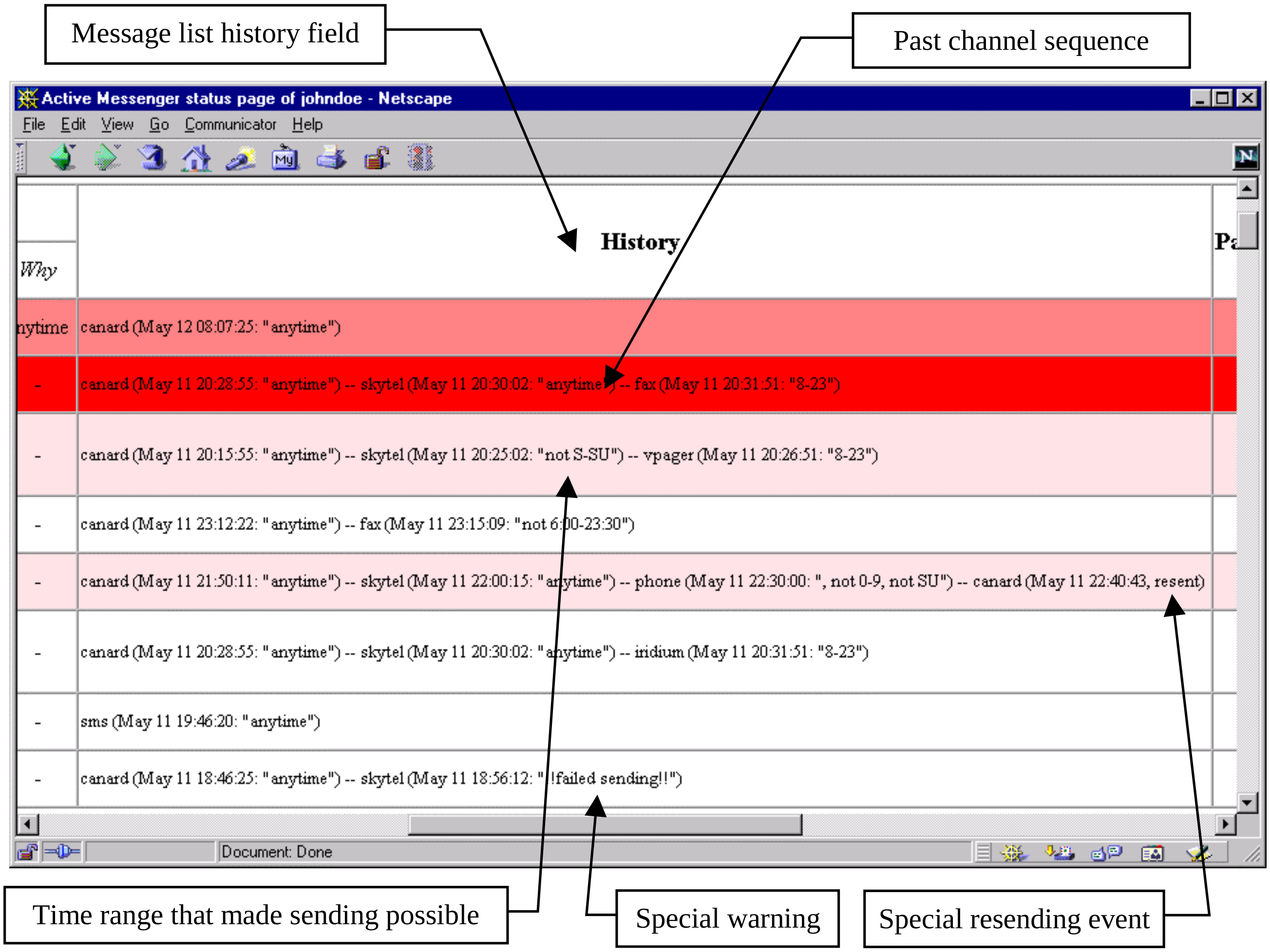
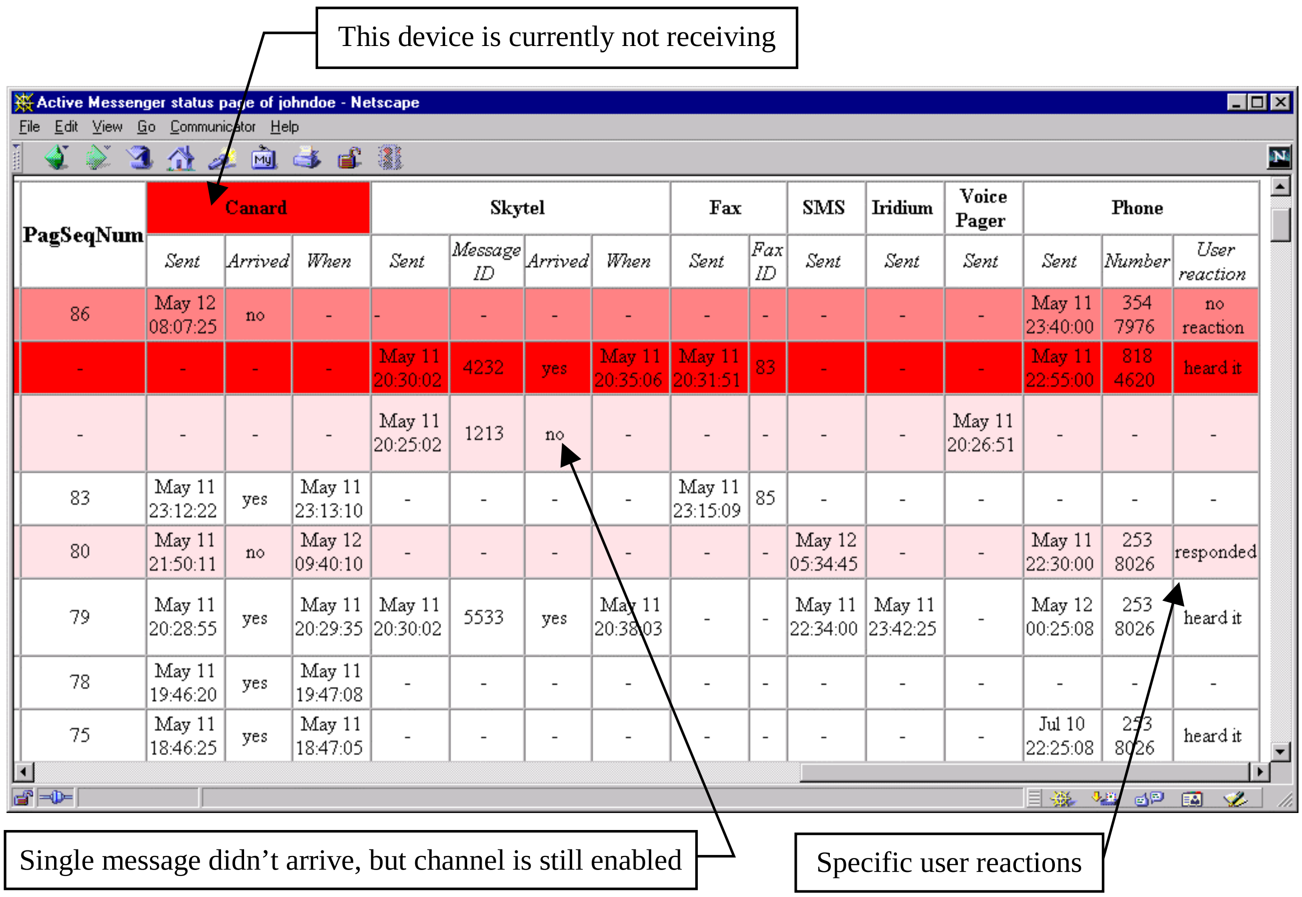


Figure 32: Status page, middle part

It is a concise summary of what has happened to the message in the past: to which channel it was sent, when, as well as the user specified time range that made the sending possible. In general, it reflects the part of the channel sequence that is already done. However, special events like resending of a message are also listed, as well as unsuccessful message deliveries.

The rightmost part of the message list table, shown in Figure 33: **Status page, right side**, shows device specific details. This data is also taken from the message list. It visualizes the status of the channels: enabled channels have white headers, disabled channels have red headers.



PagSeqNum	Canard			Skytel				Fax		SMS	Iridium	Voice Pager	Phone		
	Sent	*Arrived*	*When*	*Sent*	*Message ID*	*Arrived*	*When*	*Sent*	*Fax ID*	*Sent*	*Sent*	*Sent*	*Sent*	*Number*	*User reaction*
86	May 12 08:07:25	no	-	-	-	-	-	-	-	-	-	-	May 11 23:40:00	354 7976	no reaction
-	-	-	-	May 11 20:30:02	4232	yes	May 11 20:35:06	May 11 20:31:51	83	-	-	-	May 11 22:55:00	818 4620	heard it
-	-	-	-	May 11 20:25:02	1213	no	-	-	-	-	-	May 11 20:26:51	-	-	-
83	May 11 23:12:22	yes	May 11 23:13:10	-	-	-	-	May 11 23:15:09	85	-	-	-	-	-	-
80	May 11 21:50:11	no	May 12 09:40:10	-	-	-	-	-	-	May 12 05:34:45	-	-	May 11 22:30:00	253 8026	responded
79	May 11 20:28:55	yes	May 11 20:29:35	May 11 20:30:02	5533	yes	May 11 20:38:03	-	-	May 11 22:34:00	May 11 23:42:25	-	May 12 00:25:08	253 8026	heard it
78	May 11 19:46:20	yes	May 11 19:47:08	-	-	-	-	-	-	-	-	-	-	-	-
75	May 11 18:46:25	yes	May 11 18:47:05	-	-	-	-	-	-	-	-	-	Jul 10 22:25:08	253 8026	heard it

Figure 33: Status page, right side

It also displays in detail the reactions of the user to phone calls, etc. For Canard and SkyTel™, one can see easily which messages were sent there, as well as if they have arrived or not.

On the Active Messenger status web pager below the message list table, there are other tables that show other important lists and hashes. This includes the error list, the known addresses hash, history of SkyTel™ status, postponed SkyTel™ messages, “Quack” messages and Canard resending, the history of the user location, the content of the current user preference file, as well as the ignore list.

The following Figure 34: **Status page, error list** shows the error table, which is an exact copy of the internal error list, see chapter 2.5.4. If there are too many entries in this list, Active Messenger exits and has to be restarted, see chapter 2.4.5. The table is ordered inverse chronically, with the most recent error on top of the list.

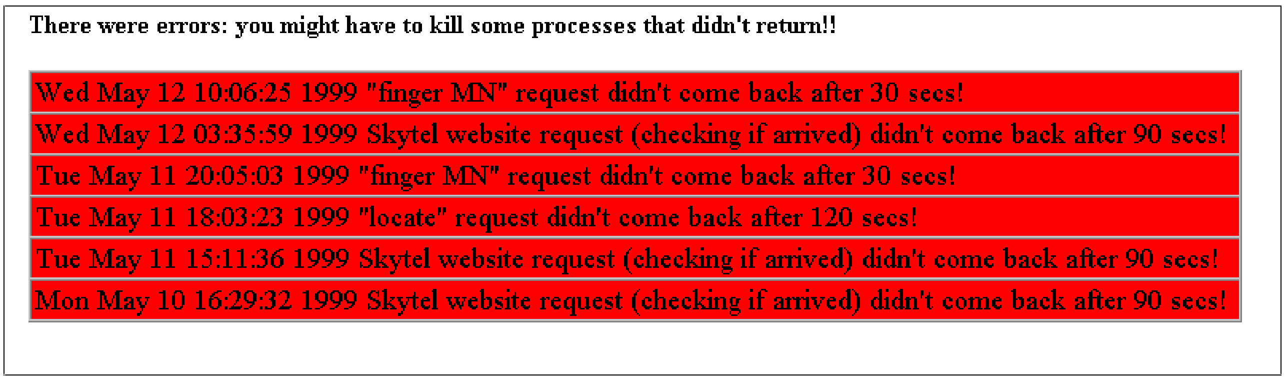


Figure 34: Status page, error list

The following Figure 35: **Status page, known user addresses**, shows the known addresses of the user and the most recent messages. The green colored rows are devices or channels from which Active Messenger has already received messages or calls[39]. This table visualizes the content of the known addresses hash, see chapter 2.5.3. This table is ordered alphabetically, with the addresses of the devices as the only truly unique identifier. This example shows also that the number of phones with built-in fax machines, e.g., 617 663-0066, is made unique by adding an additional string at the end of the fax number, “cover sheet” or “no cover,” depending on the user preferences. This string distinguishes phone numbers from fax numbers.

---

[39] Obviously, certain channels are not able to send messages at all, like Iridium™. However, Active Messenger treats them all the same, as channels or devices.

Known user adresses and most recent messages:

Address	*Device*	*At location(s)*	*Last message received from there*
161234542@omnipoint.net	sms	default	Sun Jul 4 07:18:40 1999
14576@skytel.com	skytel	default	Fri Jul 16 16:35:49 1999
653-6423	phone	office	Mon Jul 19 13:48:23 1999
238-8363 (cover sheet)	fax	office	no message received yet
617 663-0066	phone	home	Mon Jul 19 09:20:56 1999
617 663-0066 (no cover)	fax	home	no message received yet
818 453 4533	cellphone	default	no message received yet
8816456455@iridium.com	iridium	default	no message received yet
johndoe@media.mit.edu	canard	default	Fri Jul 9 12:02:50 1999

Figure 35: Status page, known user addresses

The status of the SkyTel™ paging system is described in a separate list, as shown in Figure 36: **Status page, SkyTel™ status list**. The list is ordered chronologically, the most recent message first, which is also colored green. It shows which messages were sent to a SkyWriter™ device, and which of them have arrived there. This list is necessary because the enabling and disabling of SkyTel™ sending is a rather complex process, see chapter 2.6.4.2 Disable sending, postpone, and resend SkyTel™ messages. Although SkyTel™ claims to be buffered, several messages in this example did never arrive, e.g., messages 5, 7, and 95. Therefore, the user sending a message back to the Active Messenger must have enabled Skytel™.

However, once this channel is disabled, Active Messenger postpones the messages that would usually be sent to SkyTel™. Figure 37: **Status page, postponed SkyTel™ messages**, shows such a list of postponed messages. It not only includes which message was postponed when, but also why. This refers to the fact that two messages not arriving at the pager cause Active Messenger to disable the sending. Once the system is receiving messages again, the agent will resend the postponed message, unless the user has read them already.

Did the last Skytel message arrive: yes
Is Skytel sending enabled: yes
History of Skytel status:

Last message	*Sent*	*Did it arrive?*	*When*
34	Jun 10 19:45:16	**yes**	Jun 10 19:45:44
32	Jun 10 19:30:33	**yes**	Jun 10 19:30:59
27	Jun 10 18:35:15	**yes**	Jun 10 19:02:29
27	Jun 10 18:35:15	**no**	-
22	Jun 10 16:51:09	**yes**	Jun 10 16:51:30
14	Jun 10 15:33:10	**yes**	Jun 10 15:33:29
9	Jun 10 13:53:09	**yes**	Jun 10 14:50:44
9	Jun 10 13:53:09	**no**	-
7	Jun 10 13:51:54	**no**	-
5	Jun 10 13:03:12	**no**	-
94	Jun 10 11:32:54	**yes**	Jun 10 11:33:14
95	Jun 10 11:26:31	**no**	-
91	Jun 10 10:55:45	**yes**	Jun 10 11:02:37
91	Jun 10 10:55:45	**no**	-
88	Jun 10 10:14:27	**yes**	Jun 10 10:16:14
61	Jun 9 18:08:23	**yes**	Jun 9 18:08:44
55	Jun 9 14:18:53	**yes**	Jun 9 14:19:14

Figure 36: Status page, SkyTel™ status list

These messages are postponed since the last Skytel message didn't arrive:

Postponed message				**Why postponed**		
Message number	*When postponed*	*Delay before sending*	*Skytel address*	*Did last Skytel message arrive*	*What was its message number*	*When was it sent*
82	May 12 10:40:38	3	1234@skytel.com	no	88	May 9 15:06:49
80	May 12 10:28:29	0	1234@skytel.com	no	88	May 9 15:06:49
68	May 12 00:42:35	7	1234@skytel.com	no	88	May 9 15:06:49
67	May 12 00:33:13	2	1234@skytel.com	no	88	May 9 15:06:49

Figure 37: Status page, postponed SkyTel™ messages

Similarly, the resending of Canard messages is visualized in Figure 38: **Status page, Quack messages and resending Canard messages**. Since resending is triggered by "Quack" messages (see chapter 2.6.4.1), this table lists all quack messages, as well as when Active Messenger decided to resend email messages. Note that not each incoming quack message should trigger resending, but only single ones.

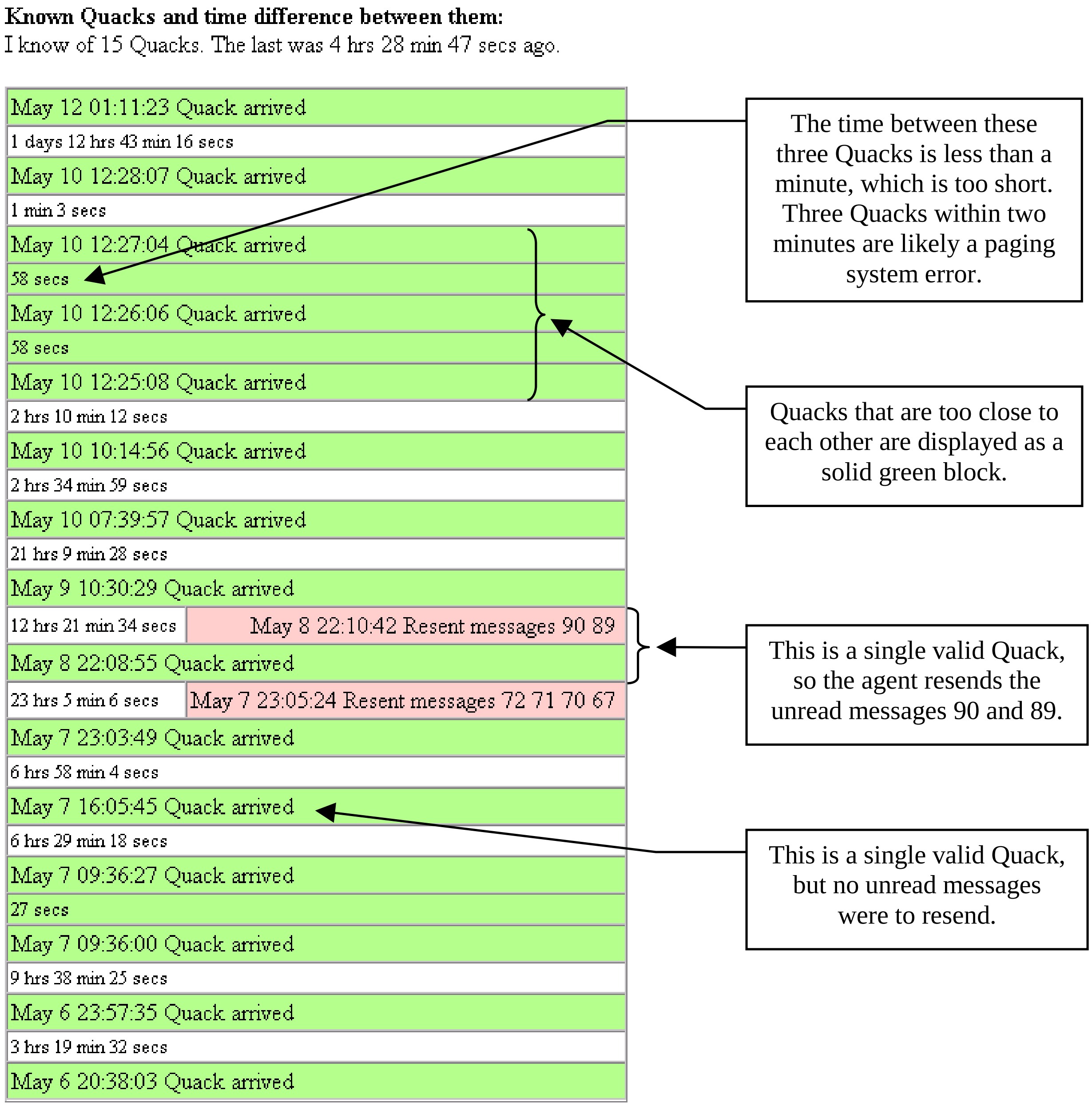


Figure 38: Status page, Quack messages and resending Canard messages

Another list that is visualized on the web page as a table is the location of the user over time. The current and most recent user location is the first green colored row. Each earlier location is an additional white row, separated by a red row that indicates the amount of time without location information. However, the only relevant information for the agent is the current location, because Active Messenger uses it to look up location-specific phone and fax numbers, see chapter 2.6.2 *Find user location*.

History of user location:

No location information for the last 1 min 23 secs				
Location	*Explanation*	*From*	*Until*	*For how long*
office	active on bluemoon	Jul 19 15:24:31	15:58:51	34 min 20 secs
No location information for 20 min 13 secs				
office	active on bluemoon	Jul 19 14:47:19	15:04:18	16 min 59 secs
No location information for 20 min 58 secs				
office	active on bluemoon	Jul 19 13:49:56	14:26:21	36 min 25 secs
No location information for 1 min 5 secs				
office	caller ID 2345365	Jul 19 13:48:51	13:48:51	0 secs
No location information for 4 hrs 26 min 53 secs				
home	caller ID 8874454	Jul 19 09:21:58	09:21:58	0 secs
No location information for 0 secs				
home	remotely logged in from dialup	Jul 19 09:19:33	09:21:58	2 min 25 secs
No location information for 1 days 10 hrs 8 min 30 secs				
home	remotely logged in from dialup	Jul 17 22:52:14	23:11:03	18 min 49 secs
No location information for 5 hrs 6 min 18 secs				
home	hint from user	Jul 17 17:45:56	17:45:56	0 secs
No location information for 6 hrs 50 min 19 secs				
outoftown	remotely logged in from slip	Jul 17 10:47:00	10:55:37	8 min 37 secs
No location information for 1 hrs 54 min 55 secs				
outoftown	remotely logged in from slip	Jul 17 08:28:54	08:52:05	23 min 11 secs
No location information for 9 hrs 19 min 3 secs				
outoftown	remotely logged in from slip	Jul 16 20:42:03	23:09:51	2 hrs 27 min 48 secs
No location information for 4 days 8 hrs 57 min 56 secs				
office	active on bluemoon	Jul 12 09:59:14	11:44:07	1 hrs 44 min 53 secs
No location information for 2 days 14 hrs 25 min 3 secs				
office	active on bluemoon	Jul 9 17:25:26	19:34:11	2 hrs 8 min 45 secs

How recent is current location information?

Current user location

Figure 39: Status page, user location history

At the very bottom of the web page, there are tables that are less relevant for the user, but still useful for the developer of the agent:

- Caller ID list: the telephone numbers Phoneshell writes to a log, see chapter 2.5.4. Note that not all phone numbers necessarily give hints to the user's location.
- Computer names Active Messenger knows of, as well as the locations attributed to them.
- The ignore list, see chapter 2.4.4.
- Complete content of the user preference file, see chapter 2.5.3.

Caller ID list:

Jul 26 18:00:24	812-1122
Jul 25 21:15:44	921-343-8654
Jul 25 19:45:14	921-343-8654
Jul 25 08:28:18	681-0004
Jul 23 22:31:07	681-0004
Jul 23 18:03:16	681-0004
Jul 23 17:47:13	681-0004
Jul 21 07:39:46	921-343-8654
Jul 20 17:40:05	812-1122
Jul 20 17:36:12	998-4443
Jul 19 17:25:09	812-1122
Jul 19 13:48:23	254-9876

Computer names I know of:
home: broink
office: kimbin, mascara, reuter
outoftown: slip

Ignore list:
klugscheisser@
dork@
henrys@
tagg@cuncon
rester@softmail
clumsy
bb23@

Figure 40: Status page, Caller ID list, computer name list, and ignore list

2.7.3 Log file and screen output

Although the information provided on the web page is usually detailed enough to keep track of what the agent did, Active Messenger also writes a log file. Figure 41: **Sample log file** shows some lines of it.

```
++++++++++++++++++++++++++++++++++++++++++++++
Wed Jul 14 18:17:23 1999: AM starting up....
Your are johndoe, and this program runs on MN.
++++++++++++++++++++++++++++++++++++++++++++++
Wed Jul 14 18:17:29 1999: New message 1 arrived:
  From John Doe <johndoe@media.mit.edu>, "veryimportant".
  Next what: canard. When: Wed Jul 14 18:17:29 1999. Number: johndoe@canard.mit.edu.
Wed Jul 14 18:17:29 1999: New message 2 arrived:
  From Operator <root@.media.mit.edu>, "own".
  Next what: no channels listed.
  !!!!This is a QUACK message!
Wed Jul 14 18:17:38 1999: New modification of mail spool file. Parsing it...
  55 messages total, 51 read, 0 opened, 4 unread.
Wed Jul 14 18:17:44 1999: New "message read" percentage for message 1:
  Old: 0%. New: 100% due to "status: GONE" in spoolfile of MN.
Wed Jul 14 18:18:45 1999: Sent message 30 to Canard pager.
  address: johndoe@media.mit.edu, PagSeqNum: 77.
Wed Jul 14 18:19:07 1999: New "message read" percentage for message 30:
  Old: 0%. New: 15% due to "arrived there" of canard.
Wed Jul 14 18:19:51 1999: New modification of mail spool file. Parsing it...
  52 messages total, 51 read, 0 opened, 1 unread.
Wed Jul 14 18:19:57 1999: New "message read" percentage for message 31:
  Old: 0%. New: 100% due to "status: GONE" in spoolfile of MN.
:::Wed Jul 14 18:42:22 1999: New user location detected: office
  It is from "finger MN": logged in to MN from klonk.
Wed Jul 14 18:56:16 1999: New modification of mail spool file. Parsing it...
  53 messages total, 51 read, 0 opened, 2 unread.
:::Wed Jul 14 21:30:38 1999: New user location detected: home
  It is from "finger MN": logged in to MN from brimbrom.
Wed Jul 14 21:30:53 1999: New modification of mail spool file. Parsing it...
  53 messages total, 52 read, 0 opened, 1 unread.
Wed Jul 14 23:19:07 1999: New message 37 arrived:
  From Operator <root@media.mit.edu>, "own".
  Next what: no channels listed.
  !!!!This is a QUACK message!
Wed Jul 14 23:19:14 1999: New modification of mail spool file. Parsing it...
  55 messages total, 52 read, 0 opened, 3 unread.
Wed Jul 14 23:19:21 1999: !!! Resending messages to Canard pager!
  Resending these messages: 36.
!!! Resending message 0:
Wed Jul 14 23:19:21 1999: Sent message 36 to Canard pager.
  address: johndoe@media.mit.edu, PagSeqNum: 81.
:::Wed Jul 14 23:53:24 1999: New user location detected: home
  It is from "finger MN": logged in to MN from brimbrom.
Thu Jul 15 02:43:41 1999: New Phoneshell Caller ID:
  "245-4674" at Jul 15 02:42:45.
:::Thu Jul 15 02:43:42 1999: New user location detected: home
  It is from Phoneshell: caller ID 2454674.
:::Thu Jul 15 02:49:57 1999: New user location detected: office
  It is from "locate": remotely logged in from mamma.
>>>Thu Jul 15 11:29:19 1999: Sent message 44 to voice pager.
  The number was: 2345343.
??? Fri Jul 16 21:32:25 1999 phone call didn’t come back after 240 secs!
Sat Jul 17 20:15:37 1999: New modification of mail spool file. Parsing it...
  45 messages total, 43 read, 0 opened, 2 unread.
```

Figure 41: Sample log file

Each entry starts with a time stamp, followed by two lines of text. If it is an unusual event, some special characters are printed first to make stick out the entry visually from the normal entries. For critical events, Active Messenger writes very extensive log entries, e.g., when a message is sent to a voice pager or to a phone.

Note that the log file is not meant to be readable by a standard user, but mainly by the developer to debug flaky modules and locate errors.

The Active Messenger writes also to the screen. The amount of data is typically about 100 lines per main loop, which means 400 lines per minute approximately. An example of the output of a single main loop is shown in Figure 42: **Sample screen output**. Upon special events, the screen output is increased. The user has also the possibility to get more specific information about a module by enabling a verbose mode for single modules, see Table 15: **Preference file default values**.

The screen output is meant for a person maintaining the agent. Therefore, all agents currently running are executed from within the UNIX *screen* utility[40] that allows detaching the screen output from the console, and attaching it later to another console. Like that, a user can start an agent, detach the window, and later look at the current screen output on any other console, possibly logged in from another computer.

```
=========Start of main loop, Tue Jul 27 14:19:26 1999======

++++NEW MESSAGES ARRIVED? Let's look if new messages have arrived...
No. The Sequence Number file was not modified since last loaded.

++++DETERMINE CURRENT LOCATION
Calling subroutine CheckFingerMN...
Done calling subroutine CheckFingerMN.
Skipping "locate": location info is recent enough, less than a minute old.
Calling subroutine CheckPhoneshell...
Let's check the Phoneshell log file /speech/desk/data/pshell_admin/johndoe.number.
  I have found the log file.
    The Phoneshell log hasn't changed.
Done calling subroutine CheckPhoneshell.

==> The Current location is office (logged in to MN from sandman).

++++MESSAGES READ? Let's look if messages were read...
Let's mark all messages READ which arrived before the mail spool file was last accessed.
The "last accessed" time of this file: Tue Jul 27 14:01:38 1999.
User isn't active ("logged in to MN from sandman") or
location info is not recent (0 secs old), and finger or locate are in use.
Therefore, I won't check the spool file.

Let's mark all messages READ which in the spool file have status
Opened = O, or read-and-opened = RO, or are expunged.
The "last modified" time of this file: Tue Jul 27 14:01:37 1999.
It hasn't been modified since the last loop. Therefore, nothing to update.
```

40 http://sunsite.utk.edu/neighborhoods/gnu/screen/screen_toc.html.

```
++++CHECKING CANARD AND SKYTEL...
Calling subroutine CheckSkytel...
Done calling subroutine CheckSkytel.

Calling subroutine CheckCanard...
   Yes, we have to check the Canard summary web page!
Done calling subroutine CheckCanard.

Calling subroutine IsCanardReceiving...
   No arrival after last sent.
Done calling subroutine IsCanardReceiving.

Is Canard receiving? no.

Calling subroutine GetSkytelStatus...
   Looking for most recent message sent to Skytel.
   There were no messages sent to Skytel yet.
Done calling subroutine GetSkytelStatus.

Did last Skytel message arrive: nothing sent yet.

Calling sub CheckPagerLogfile.
Let's check the .PagerLogfile /u/johndoe/.PagerLogfile.
 I have found the file.
   The .PagerLogfile hasn't changed!
Done calling sub CheckPagerLogfile.

Is Skytel sending enabled: yes.

++++CHECKING QUACKS...
I know of 23 Quacks. The last Quack was 2 hrs 21 min 31 secs ago.
The difference between the last two Quacks is 24 min 18 secs.
No time scheduled to resend Canard messages.

++++MESSAGES DUE TO SEND?
Now I set every message newly above 95% read to "done".
  Done setting messages above threshold to read!

Now I send due messages that are not sufficiently read (below 95%.)
 Done looking for messages to send!

++++DUMPING MESSAGE LIST TO HTML FILE.

++++NEW USER PREFERENCES? Let's look if the preference file has changed...
No. The Preference file was not modified since last loaded.

++++CHECKING VALIDITY OF CLUES CATEGORIES...
Calling subroutine CheckCluesCategories...
  Done calling subroutine CheckCluesCategories.

++++IS AM WEDGED AND HAS TO DIE??
There are 3 zombie processes.
There are no clues status errors.

===========End of main loop, Tue Jul 27 14:19:33 1999======

         ...waiting for 12 secs...
```

Figure 42: Sample screen output

3. Iterative design and user evaluation

This chapter is about the iterative design process, illustrated with anecdotes, and evaluation, as "evidence by actual usage."

The following sections are excerpts from a diary that was written by the author during the development of the Active Messenger.

The first drafts of the concept of Active Messenger were written down six months ago. The first user started using Active Messenger three months ago. Currently, there are two users running Active Messenger, several others may follow soon.

3.1 Evaluation of Active Messenger

Evaluating Active Messenger is difficult because the current user population is small. The reason for the small user population is that the agent is still a prototype. However, it is stable and behaves adequately for more than a month now, so the developer plans to make the software available to more users soon.

Although the current user population is small, the two people that use the agent on a daily basis for up to three months now are able to provide relevant feedback about the success of the project and give some evidence of its usefulness.

There are two questions:

- Does Active Messenger do what the *developer* wants it to do?
- Does Active Messenger do what the *user* wants it to do?

The answer to the first question is "yes." In the current version, the agent performs in a way the programmer wants it to. There are no known problems that keep the agent from doing what the developer supposed it to do. However, this is not enough. The agent was not built for the programmer, but for the user.

The answer to the second question is also "yes," at least for the current users: Active Messenger does what the user wants it to do. There is one proof for that: **The users would not use Active Messenger if it would not perform better than the system they used previously. They would not hesitate to turn the agent off, because a working communication infrastructure is absolutely essential for their professional as well as personal life**. If their previous system would perform better, these users would switch back immediately, especially because switching back is very simple: calling in or logging in and copying a file would disable the agent. However, none of the users turned Active Messenger off within the last month. This indicates that the agent does what the user expects it to do and performs even better than the previous system.

The two users have been using Active Messenger extensively over several months now.

- User A makes Active Messenger process around 90% of all new messages. This means, the user has set up her Clues filtering in a way so that 90% of all incoming messages fall into categories for which the agent has rules. These rules tell the agent to forward such a message in one way or the other. For user A, these 90% mean 48 messages per day in average. Because she has used the agent for 21 weeks now, the agent has processed already over 7000 messages for her.
- User B makes Active Messenger store and forward around 38% of all incoming messages, which stands for about 50 messages per day for this specific user. She started using the agent 14 weeks ago, so Active Messenger has processed almost 5000 messages for her.

3.2 Surgery

This project is different from many other Media Lab projects: Active Messenger was developed interactively with the help of the users. This is good because the developer has immediate feedback about if an idea or concept works and improves the overall performance of the agent. The bad thing is that it can be like "surgery on living flesh": a very sensitive task. Each time a new version is started up, the functionality of the whole system has to be checked carefully, because each bug decreases the users' satisfaction with the system. A working communication infrastructure is very important, and Active Messenger is the centerpiece of it, because it manages all incoming text messages.

Iterative design can be frustrating because it means, "modifying the rules during the game." The users have new ideas and insights when using the system, which often means that code segments get obsolete. However, the current status of the project, a reliably working agent, indicates that it was the right way to go.

3.3 Ease of use

The very first user of Active Messenger complained about many details. Beside minor things like "I hate upper case file and directory names," she complained that she had to create a directory to put the configurations file in. An installation wizard should do that for her.

Furthermore, she complained about the size of the generic preference file. It is too big and has too many entries. Obviously, first time users don't want to deal with settings like "percentage read from spool file," mainly because they don't understand the meaning. The solution would be to start Active Messenger in a "basic" mode, and then give the user the possibility to switch to an "advanced" mode, like ICQ™ does.

The preference file can be reduced to a minimum, using the defaults of the PERL script. If all comments and explanations are removed, it shrinks to a few lines, see Figure 7: **Sample user preference file**. In summary, there are users who want to have an extensive self-explanatory preference file, and other users who prefer a small preference file and a separate manual, perhaps in the form of a UNIX manual page.

However, there are two reasons why there are so many user options in the preference file (see Table 15: **Preference file default values**):

- *Full control over what happens*. Active Messenger is a prototype, and the developer needs to adjust the parameters to find the optimal basic settings. Since the user preference file is loaded dynamically, all variables of a running Active Messenger can be modified without rebooting the agent. Rebooting is not desired because Active Messenger would lose all information of what has happened so far. Running Active Messenger for a long time is also prerequisite for testing certain features. This is related to the problem that the complexity of Active Messenger makes it very difficult to test how the agent behaves in extreme situations. It requires a lot of patience (time) and adjusting of external parameters (e.g., the Canard paging system) to force it to get in an extreme program state.
- *Different users expect it to work in different ways*. Therefore, Active Messenger has to be adjusted to their expectations.

3.4 No user is the same

One of the primary goals for Active Messenger was to avoid redundancy. If a user is sitting in front of a terminal, she doesn't want to get messages on her pager. Active Messenger avoids that by looking at the information given by the Activity Server, and by checking the status of the mail spool file. However, it is important to see that not every user wants this feature. Some may want to be notified by the vibration of the pager if a new message has arrived. This problem was solved by having "read threshold" levels that can be specified by the user. For some of them, a change of the mail spool file access time keeps Active Messenger from sending a message to the next device. For some other users, the arrival of a message at Canard or SkyTel™ is enough to stop Active Messenger. For others, only opening the message in a mail reader program stops the agent.

3.5 Complexity

With over 5000 lines of PERL code, Active Messenger has reached a size which inevitably leads to complexity. Here is a relatively trivial example. On April 30 1999, a user writes email to the developer saying, “something strange happened last night.” The question was how could a message arrive at Iridium™ before it arrived at Canard, Canard being the first channel? One possible explanation was that the Canard server had delayed the message accidentally, so that even if the message was sent to Canard first and to Iridium™ afterwards, it arrived at Canard second. Second explanation, and more likely to be the case, is that the message was sent first to Canard, then to Iridium™. After that, a Quack arrived which indicates that the pager came back in range (see chapter 2.6.4.1), which in turn made the agent send the same (still unread) message to Canard a second time. The web page would show these events, but the user couldn't remember the number of the message. This example shows how difficult it can be to verify the correct behavior of the Active Messenger, even in simple cases.

3.6 Confidence

As a consequence of the agent's complexity, it is difficult to follow its “reasoning.” Therefore, a lot of effort was put into a web page that explains what the agent does. However, the web page is only for controlling purposes, and a web browser is not available all the time, especially not a the very moments a message delivering agent I most useful. Therefore, the user has to trust that Active Messenger is doing the right thing. If she doesn't trust the agent, she will stop using it sooner or later. This happened once for Active Messenger, when the agent was temporarily so unreliable that the user switched back to her prior configuration.

Several times, the web page was good for more than just displaying the activity of Active Messenger. It can also prove that Active Messenger is working. Another proof could be the log file. It was modified several times to log operations that failed and were difficult to track, e.g., the sending of messages through the SkyTel™ web interface. Currently Active Messenger writes to the log the whole web file that SkyTel™ sends back, so the user can check manually the original web page if the sending was successful.

The part of Active Messenger code that is essential for proper functioning is small, not more than a third of the current program. However, because nobody trusts “experimental code” without having insight in the “reasoning” of the system, the parts of the program that monitor and visualize what is going on are absolutely necessary. Perhaps in the future, one could run a version of Active Messenger without these parts to make it use fewer resources, but only if the user trusts the agent blindfold.

3.7 Data storage

3.7.1 No ASCII ToDo list

The initial idea for data storage was that Active Messenger would generate a ToDo list, regularly update and delete the entries that are done, and store the whole list to a file. Then it would sleep for a short time, and reload the ToDo list. However, it turned out that storing a ToDo list in an ASCII file would be too much file I/O. A better way would be to have an internal representation of the list, as well as of the state of the system. However, the structure of the agents gets more abstract and opaque like this.

3.7.2 No external database

Other suggestions included the use of an external database where all events and the message list are stored. A MySQL server was installed and tested successfully. The advantage of an external database would be that if the computer crashes where Active Messenger runs on, the agent would be able to continue after rebooting, without worrying about the storing variables, etc. However, this idea was given up because it would rely too much on networked machines. With the network going down, the agent would become incapable of doing its job.

3.8 Parallel versus sequential architecture

Therefore, the only possibility left was to keep all data in program memory. One possible implementation was to use shared memory to store all the data. Several processes running in parallel would access and modify this data. Eventually, it was decided that the architecture would be more transparent if Active Messenger called these processes sequentially. There would be no race conditions, and the agent could be realized as one PERL script. The only question was if such a sequential architecture would be fast enough. After writing the core code, it turned out to be sufficiently fast, even with a built-in12 second pause after each loop.

3.9 Stability

After Active Messenger crashed because an external program didn't return, protective time outs were added to all external calls. However, it took a long time to find the appropriate values, because timing out means creating a “zombie” process, which in turn can cause other problems. Therefore, finding the right time out values is a trade-off between making Active Messenger wait for external calls long enough without letting it stall for too long. In order to find out more about the temporal behavior of external calls take, Active Messenger is measuring the time each external call takes. The user can enable a log file that writes down these values. The log was useful several times when an external system started to become unreliable and crashed eventually. With the log, the user could trace back this outage easily.

Due to the timing out of external programs, on April 30 1999, Active Messenger successfully survived a crash of the Media Lab's main mail machine. Several *locate* and *clues filter* requests were hanging and had to be killed manually on MM, but otherwise, Active Messenger kept working. Earlier on the same day, a user was "sprayed" with 80 "Quack" messages from a defective Canard system. Active Messenger kept working fine, it resent messages only twice, each time 10 messages. This was an improvement over the system that was used before Active Messenger. The 80 "Quack" messages would have triggered 80 summaries sent to the pager.

However, timing out external programs turned out not to be good enough. During the night from May 2 to May 3 1999, an important workstation froze. Active Messenger kept working fine until the limit of processes was reached, 64 are the maximum. This means, it kept trying to "locate" the user, which in this case includes opening a remote shell on another workstation. All "locate" requests timed out after two minutes, and eventually, no processes were left for this user anymore. She couldn't even log in remotely to the machine to shut down the agent, but had to go to the office where an Xterm window was still open. From there, she could terminate Active Messenger and all other waiting processes.

3.10 Memory span

If a user receives many messages, e.g., 100 a day, the memory span of Active Messenger is exactly one day. Older messages are overwritten and can't be accessed anymore. If the user is away for several weeks, this becomes a big problem. One solution would be to increase the amount of messages that the agent stores, as a file and in the message list. But this increases also the RAM needs of the program. Another solution would be to decrease the amount of messages that Active Messenger holds in memory. The "ignore list" at the very end of the user preference file serves this purpose. Active Messenger ignores all email from these addresses. However, only the hardware resources of the workstation it is running on limit the number of messages that the agent can keep in memory.

3.11 Prioritization and categorization of email... and people

Although categorization of email seems to be an organizational process, there are also psychological implications. The partner of an Active Messenger subscriber was very disappointed when he saw that he had to share "his" category, "superimportant," with the best friend of the user. He probably felt that being her partner would be reason enough to be put in a special category on his own, or at least one with a higher importance than where the friends are in.

3.12 What should stick out most on the status page: important messages or unread messages?

There is no clear answer. Some users prefer the former, some the latter. Therefore, the user can specify the coloring. In the first case, the most important messages stick out visually because they are colored in bright red. In the second case, a message that is not red is very bright. The "brightness" of the message decreases proportionally to the probability that the user has read it.

3.13 Category dependent read threshold level

A suggestion was that the "message read" threshold should be relative to a category: the more important the category, the higher the percentage has to be to mark the message read. However, Active Messenger may get opaque with this feature. It would be almost impossible to determine what the agent *should* have done. It is already difficult to detect "misbehavior" of the Active Messenger, and a flexible "read threshold" would probably make it even worse.

3.14 Sending to the first device

A user observed and reported the following behavior change: If a message from a group member arrives, a special auditory cue is played on her workstation. As specified in her preference file, such a message is sent to a pager after a few minutes. To avoid that, she reads the message immediately if she is at her desk. Therefore, this configuration seems to be an additional incentive to read such email messages immediately. The user is undecided about if this is a positive or negative behavior change. However, after this experience, she sets the initial delay higher, so that she has more time to reply until a message gets sent to the first device.

After this event, additional changes were made to Active Messenger. Because one of the primary goals was not to send email to a pager if the user is active on the terminal, it was important to delay the sending to the first device by a few minutes to give the user time to read it. However, if the user is not logged into a computer, important messages are not sent to the pager instantaneously. The solution was to reduce the initial delay proportionally to the user's idle time. The longer the user is idle, the less the initial delay. If she is idle for more than 15 minutes, an arriving message gets sent to the first device immediately. The initial delay can be set relatively high, so that the user has enough time to read a message on the screen, without being stressed out by the agent. On the other hand, if she is away, a message is sent to the first device without any further delay.

However, even this strategy can fail. One time, Activity Server went down, and Active Messenger didn't get any new location information. It was found out because all "locate" requests returned after 0 seconds, which is not possible. The user was irritated because Active Messenger forwarded all messages immediately to the first device, Canard, due to the fact that if the user is idle for more than 15 minutes, the first delay is reduced to zero minutes. Because "locate" didn't give any meaningful information back, Active Messenger assumed correctly that the user was not active. As a consequence, "locate" can be disabled in the preference file. However, it would be better if Active Messenger could detect a faulty server and ignore it as long as it is not functioning, perhaps even sending a warning to the user.

3.15 Location modifies channel sequence

In the current implementation of Active Messenger, the channel sequences are always the same. Location specific entries can reduce only the use of a certain device, but not modify the whole channel sequence for a category. However, if the user is out of town, she may want to change the forwarding rules for messages of a certain category, e.g., "important." A possible solution would be to allow entries like "important = iridium, canard" in the location sections to modify the default channel sequence. However, this would make the behavior of Active Messenger less traceable.

There is also another way of realizing this behavior. If the SkyTel™ status is "not receiving," and therefore the channel is skipped, the agent sets the delay to the next channel to zero minutes. E.g., if the channel sequence is "canard, skytel, iridium (240)," and Skytel™ is skipped because sending is disabled, the message is sent immediately to Iridium™. Otherwise, if Skytel™ is active, Iridium™ likely won't get used at all. It is like switching from Skytel™ to Iridium™, once Skytel™ is out of range.

3.16 Disabling message sending to SkyTel™

SkyTel™ claims that each message sent to a SkyWriter™ device will arrive eventually. However, our experience does not support that claim. Several times, messages that were sent to SkyTel™ did never arrive, even if the pager came back in range and was able to send messages. Because SkyTel™ messages are expensive, Active Messenger tries to minimize the sending.

The first try was to stop sending messages if the last one didn't arrive, and start sending again when it arrives. However, because there are messages that never arrive, SkyTel™ can't be enabled anymore, unless the agent is restarted. The situation gets even worse if the user is away for a long time, and more than 100 messages have arrived since the last message was sent to SkyTel™. In this case, the agent is completely locked up and is not able to give the channel free again. This was unacceptable, so two features were built in to prevent a complete locking of SkyTel™. First, only if two messages in a row don't arrive, the sending is disabled. Second, if a message arrives from the pager, the channel is enabled again, regardless of its previous state.

Another method could be implemented: It is possible that a SkyWriter™ device comes back in range and can send messages, but the central server does not register the hand set. In this case, sending a message to the device would make it register properly with the server, which in turn may resend all the messages that haven't been delivered yet. So if the sending of messages to SkyTel™ is disabled, the agent could send a few test messages to "wake up" a not properly registered hand set.

3.17 Increasing idle time decreases probability of user having read a message

The ideas at the beginning of the agent's development were not always the best. E.g., an early idea describing how an agent could find out if the user has read an incoming message from the mail spool file, is shown in Figure 43: **Likelihood of a message getting read depending on user idle time**. Each incoming email message is assigned a likelihood of getting read. This likelihood is inverse proportional to the idle time of the user. The longer the user is idle, the less likely she has read a message. The likelihood becomes zero when the user disconnects.

However, this idea became obsolete because Active Messenger can detect much more accurately if a message is read by parsing the mail spool file. Nevertheless, if users start reading email from sources other than the mail spool file, such a feature would become relevant again.

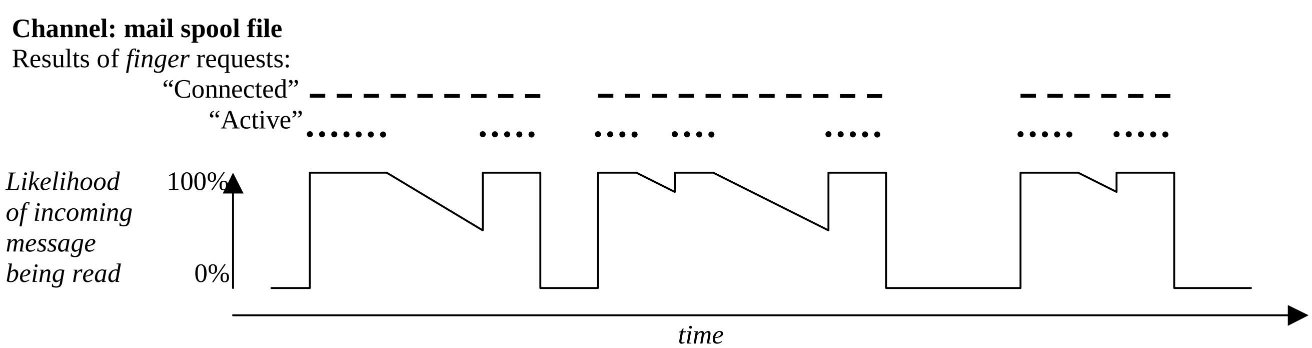


Figure 43: Likelihood of a message getting read depending on user idle time

3.18 Phoneshell and mail reader change the file access time

If a user calls up Phoneshell, she doesn't necessarily read all messages. However, they all get tagged because Phoneshell changes the mail spool file access time. The same thing can happen if a PINE mail reader program accesses the mail spool file irregularly. A simple but useful solution was found. If an access time change occurs, the agent marks the messages only if the user is active on a computer terminal and the location information is less than ten minutes old. This successfully blocks out changes of the mail spool file access time.

3.19 Running several agents at the same time

It happened several times that a user accidentally was running several agents in parallel. There are several reasons why this would happen, e.g., because a user has restarted Active Messenger with her pager, etc. However, Active Messenger is not built to run in several instances for the same user. Nevertheless, it is possible to run several instances, and they actually work almost normally, which indicates the agent's high stability. One way to find out about multiple agents is that two agents update the web page in turn. Future improvements of the agent could include an alerting mechanism so that several agents on behalf of the user recognize each other.

3.20 A final note

As every human artifact, software programs may have errors. But even if they would be perfect, it very often comes down to what the user does with them. E.g., May 6 1999, 9:30pm, a user calls up the developer saying, "Active Messenger is not doing what it should!" After looking at her web page, it became obvious that Active Messenger was not running at all since 2:15pm! The user has rebooted her computer several times because the mouse was locked up, and then just forgot to restart the agent. Of course, Active Messenger could be started up automatically after rebooting a machine. However, the defective mouse is clearly another problem: After several unsuccessful reboot processes, the user found out that the optical mouse was not working because a business card was stuck to it from underneath...

4. Future work and wish list

This section describes current limitations of Active Messenger and suggests initial approaches to overcome them.

4.1 Extending user population, user testing

The main problem with Active Messenger is that the user population is too small to draw statistically significant conclusions with high validity. Therefore, it is important to get more users. Once the agent is spread more widely, a survey could tell more about the users' reactions. Here are some questions that could be asked:

- Does the user get important messages more timely or not?
- Does the agent help the user manage her communication infrastructure better or not?
- Does the agent change the users' view about having many communication channels available or not? If yes, how does it change it?
- Are the devices and channels less obtrusive during the users' daily life if Active Messenger manages them?
- How useful is the Web status monitor page? Do users look at it at all?
- The agent converts messages from one channel to another: Is anything lost in a message when it is received through a channel quite different than what it was intended for? If so, how much does it matter, compared with the opportunity to get the right messages at the right time?

4.2 Web interface

The World Wide Web (WWW) is very poplar. According to Nielsen//NetRating[41], during June 1999, 63.4 million people surfed the Web. These people use the WWW as their main Internet service. Therefore, it is possible that some people would probably prefer a Web interface for Active Messenger to a preference file on a UNIX system. Through such an interface, the user would not only see the current status of the agent, but also manage its configuration. This includes modifying all preferences and internal variables. Such a Web interface would work in parallel with the user preference file, or even replace it.

---

[41] http://www.nielsen-netratings.com/

A promising approach would be to enhance the existing status monitor web page to make it interactive. This could be realized by writing a CGI script that generates the Web page dynamically.

Furthermore, such a Web interface has to run on a secured server under the Secure Sockets Layer (SSL[42]) to provide a safe way to transmit sensitive information and to protect the interface with a password.

4.3 Internet Message Access Protocol (IMAP)

Currently, Active Messenger requires a UNIX mail spool file to work properly. However, many users have switched to server based mail systems like IMAP[43]. IMAP stands for *Internet Message Access Protocol*. It is a method of accessing email that is kept on a (possibly shared) mail server. It allows a client email program to access remote message stores as if they were local. For example, email stored on an IMAP server can be manipulated from a desktop computer at home, a workstation at the office, and a notebook computer while traveling, without the need to transfer messages or files back and forth between these computers. However, the main difference between a UNIX mail reader like PINE and IMAP is that there are Web interfaces available for IMAP, e.g., IMP. Therefore, the user can access her email from any Web browser.

It is planned to enhance the agent so that IMAP accounts can be used as well.

4.4 Interface to comMotion

Active Messenger tries to detect the location of the user by registering the computer names where the user connects from, as well as by logging the phone numbers from where the user calls in to Phoneshell [26]. However, absolute position detection would make the location module work much more accurately. Absolute location information could be obtained, e.g., by installing active badges on radio or infrared basis. However, such a system would only work inside buildings. Outdoors, other solutions have to be found. One possibility would be to trilaterate a portable device like a cellular phone or a two-way pager. Although service providers may have this information already, it is currently not available to the user. Another option for outdoors would be the Global Positioning System GPS. This is a satellite-based radio-navigation system that is developed and operated by the U.S. Department of Defense.

A project that uses GPS is comMotion [17]. It is a context-aware communication system for a mobile or wearable computing platform. A behavior-learning agent automatically learns the salient locations in the

[42] http://www.ssl.com/
[43] http://www.imap.org/

user's life. Once these places have been determined, location specific to-do lists, electronic Post-it™ notes, or reminders and subscription requested information can be delivered to the user at the right place and time. The user interacts with the system through a visual and/or speech interface.

Because comMotion already knows the users location, Active Messenger could query this information and enhance with it the accuracy of the existing location detection.

4.5 Wireless Application Protocol (WAP)

WAP[44] is a joint effort between several key players of the mobile communication industry to further develop the ideas of intelligent messaging and other similar technologies. The goal is to develop an open protocol that can be adopted by any manufacturer and that enables easy delivery of relevant information and services to mobile users. WAP is HTML-compatible and supports GSM networks and CDMA. However, the aim is to support all other current and future digital cellular technologies.

WAP allows users of cellular phones supporting the protocol access to applications and functions such as:

- Unified messaging: the management of personal telephone profiles for handling voice, fax, and email.
- Information services such as stock trading, banking, directory services, exchange rates, etc.

Because Active Messenger addresses similar problems, it seems to be appropriate to keep in touch with this development, possibly integrating WAP capable devices once they will be available.

4.6 Thread recognition

Quite often, email communication between users can be clustered into threads. This means, a user refers to one or several prior messages concerning the same issue. It would be interesting to have the agent detect a thread and the messages that are part of this thread. If Active Messenger could detect a thread reliably, messages of the same thread could be grouped together and treated in a special way.

However, thread recognition is not trivial. A simple way would be to group messages that come from the same sender and have the same subject line. This approach works only if the message from a person is always a reply to the last message sent to this person. If user A sends two messages to user B, and user B replies to one of them, it is only possible to detect a thread if the subject line is left relatively unchanged. Additionally, recognizing threads that span more than one medium is difficult because it may require content understanding. E.g., if the user sends email to a person, and this person calls the user back and

[44] http://www.wapforum.org.

leaves a voice mail message, it requires reliable speech recognition and content understanding to group these two messages together.

4.7 Media conversion

Active Messenger is capable of converting email messages to another medium, e.g., email to fax, or email to voice mail. The agent performs simple conversions, e.g., it modifies email messages that have to be sent to a pager by deleting signatures and limiting the length of the messages. Furthermore, email messages that have to be read to the user on the phone or sent to the voice pager are processed by Phoneshell [26] to enhance the intelligibility of the synthesized audio. However, it would be interesting to modify the agent so that the device characteristics are taken into account more extensively when converting messages.

The output capabilities of each device and channel are different. It would be interesting to have an agent that is able to determine the nature of the device the user is using, and deliver content in a format most suitable for its output capabilities. E.g., if the user receives a message that contains a picture, the agent decides if the picture should be sent to the device separately, depending on the capabilities of the device to display graphics, perhaps adjusting the picture resolution and size according to the available bandwidth. Similarly, sound files embedded in email message could be transmitted as audio when the agent calls up the user on the phone.

4.8 Content understanding, filtering, and scaling

Similar to media conversion, but much harder to realize, would be content understanding. This means to modify the agent so that it is able to understand the actual content of a message. This would be useful to scale up or down the message content, or adjust the forwarding rules of the agent.

For example, if a device can only display short messages, the agent would summarize a long email message so that it fits the display without losing the essential information. This is only possible if the agent understands the content of a message.

On the other hand, if a short message contains abbreviations, Web links, or people’s names, the agent could enhance the message with information from the Internet and other resources.

Furthermore, if the email contains a table of data, e.g., stock quotes, and the user’s device has a graphic display, the agent could transform the table into a graphic and send this visual information instead.

Content understanding would also facilitate thread clustering. Additionally, if the agent would be able to understand the content of a message, it could determine the importance of a message more accurately.

However, it is clear that content understanding in general is a difficult problem. Obviously, such a system would require extensive use of artificial intelligence and is therefore far from being realized soon.

4.9 Learning capability

In the long run, Active Messenger could learn from the user’s behavior about her preferences to make the configuration of the agent simpler. For example, it would be comfortable to have an agent that does not have to be taught essential facts like not to call the user late at night—without the user specifying what “late at night” actually means. But this is a hard problem: it is difficult to extract a “latest time to call user at home” if the user doesn't have a regular daily schedule.

Additionally, the main problem with learning is that it has to happen fast enough to adapt to the user's behavior changes without being too sensitive about daily variations. Furthermore, some users may not be comfortable with preferences that were not set specifically by them.

5. Bibliography

[1] *AOL Instant Messenger* [WWW Document]. URL http://www.aol.com/aim/ (visited 1998, December 21).

[2] Appenzeller, G., Lai, K., Maniatis, P., Roussopoulos, M., Swierk, E., Zhao, X., Baker, M. (1999). *The Mobile People Architecture*. Technical Report CSL-TR-99-777, Computer Systems Laboratory, Stanford University, January 1999. Online at URL http://gunpowder.stanford.edu/~laik/projects/mpa/publications/TechReport/html/TechReport.html (visited 1999, August 8).

[3] *AT&T News Release: AT&T to shut down PersonaLink Services; shift to Internet* [WWW Document]. URL http://www.att.com/press/0796/960711.iaa.html (visited 1999, August 8).

[4] Ayad, K., Day, M., Foley, S., Gruen, D., Rohall, S., & Zondervan, Q. (1998). *Pagers, Pilots, and Prairie Dog: Recent Work with Mobile Devices at Lotus Research*. Proceedings of the Workshop at CSCW'98 on Handheld CSCW, Seattle, 14 November 1998. Online at URL http://www.teco.edu/hcscw/sub/111.Day/hcscw.html (visited 1999, August 8).

[5] Chesnais, P. R. (1997). Canard: A framework for community messaging. In *The First International Symposium on Wearable Computers*, Cambridge, Massachusetts, October 1997, p. 108 - 115.

[6] Chesnais, P. R. (1999). *A Framework for Designing Constructionist Approaches to Community-Centered Messaging*. Ph.D. thesis, Massachusetts Institute of Technology.

[7] Dorner, S. (1988). *Eudora: Bringing the P.O. where you live*. Qualcom, Inc. Copyright 1988-1992, University of Illinois Board of Trustees.

[8] GURPS: Trek, the unauthorized sourcebook, internet edition, section 2 f - “Trek Tech” - part 6 - miscellaneous. [WWW Document]. URL http://home.rica.net/CaptainNemo/trek/sttech06.htm (visited 1999, August 8).

[9] Guterl, F. (1995). CeBIT '95: 007 on the Internet. *Datamation*, March 15, 1995. Online at URL http://www.datamation.com/PlugIn/issues/1995/march15/03bint50.html (visited 1999, August 8).

[10] Harmer, J. (1998). *The OnTheMove project*. BT Laboratories, Martlesham Heath, Ipswich, England.

[11] *IBM Intelligent Agent Services* [WWW Document]. URL http://www.research.ibm.com/iagents/ (visited 1999, August 11).

[12] *ICQ - World’s Largest Internet Online Communication Network* [WWW Document]. URL http://www.mirabilis.com/ (visited 1999, August 8).

[13] *Iridium*™ [WWW Document]. URL http://www.iridium.com/ (visited 1999, August 8).

[14] Knothole homepage [WWW Document]. URL http://www.media.mit.edu/~stefanm/pager/ (visited 1998, August 8).

[15] Kriaras, I., Jarvis, A.W., Phillips, V.E., Richards, D.J. (1997). *Third-generation mobile network architectures for the Universal Mobile Telecommunications System (UMTS).* Bell Labs technical journal, vol.2, number 3, summer 1997.

[16] Manandhar, S. (1991). *Activity Server: A Model for Everyday Office Activities*. Master’s thesis, Massachusetts Institute of Technology, June 1991.

[17] Marmasse, N. (1999) *ComMotion: a context-aware communication system*. Master’s thesis, Massachusetts Institute of Technology, July 1999.

[18] Marti, S.J.W. (1993) *The Psychological Impact of Modern Communication Technologies on Users*. Master's thesis, University of Bern, Switzerland.

[19] Marx, M., & Schmandt, C. (1996). *CLUES: Dynamic Personalized Message Filtering*. Proceedings of CSCW '96, pp. 113-121.

[20] *Motorola Portable Answering Machine* [WWW Document]. URL http://www.mot.com/MIMS/MSPG/Products/Voice/tenor/ (visited 1999, August 8).

[21] *OnTheMove WWW homepage* [WWW Document]. URL http://www.sics.se/~onthemove (visited 1999, August 8).

[22] *PageWriter™ 2000* [WWW Document]. URL http://www.motorola.com/MIMS/MSPG/SmartPagers/pw2k/pw2000.html (visited 1999, August 8).

[23] Reinhardt, A. (1994). The Network with smarts. New agent-based WANs presage the future of connected computing. *Byte Magazine*, October 1994. Online at URL http://www.byte.com/art/9410/sec7/art1.htm (visited 1999, August 8).

[24] Reinhardt, A. (1995). IBM Plans Ambitious Network. *Byte Magazine*, August 1994. Online at URL http://www.byte.com/art/9408/sec4/art4.htm (visited 1999, August 8).

[25] Roussopoulos, M., Maniatis, P., Swierk, E., Lai, K., Appenzeller, G., Baker, M. (1999) *Person-Level Routing in the Mobile People Architecture.* To appear in Proceedings of the USENIX Symposium on Internet Technologies and Systems, October 1999. Online at URL http://mosquitonet.Stanford.edu/publications/USITS1999/USITS1999.html (visited 1999, August 8).

[26] Schmandt, C. (1993). Phoneshell: The Telephone as a Computer Terminal. *Proceedings of ACM Multimedia '93*, pp. 373-382, New York, August 1993.

[27] *SkyTel™* [WWW Document]. URL http://www.skytel.com/ (visited 1999, August 8).

[28] *Tango™ Two-Way pager* [WWW Document]. URL http://www.mot.com/MIMS/MSPG/Products/Two-way/tango/desc.html (visited 1999, August 8).

[29] *The integrated fax server opportunity* [WWW document]. URL http://www.moreton.com.au/MBWEB/whitepapers/fax.htm (visited 1999, August 8).

[30] *The Mobile People Architecture homepage* [WWW Document]. URL http://mpa.stanford.edu/ (visited August 8, 1999).

[31] *Treknology Encyclopedia* [WWW Document]. URL http://www.uni-siegen.de/dept/fb12/ihe/bs/startrek/treknology1.htm (visited August 8, 1999).

[32] van den Berg, S. R. (1994, October). *Procmail – Mail processing Package* [FTP archive]. RWTH-Aachen, Germany. Available FTP: Hostname: ftp.informatik.rwth-aachen.de/pub/packages/procmail/procmail.tar.gz.

Version 1.1, September 3, 1999.