Centralised alarm distribution

Anders Hallberg
Markus Helgesson

March 2006

LUND UNIVERSITY

Master's Thesis in Industrial Automation
Department of Industrial Electrical Engineering
and Automation

Supervisors:	Gustaf Olsson, Lund University
	Håkan Jeppsson, Beijer Electronics AB
Examiner:	Gunnar Lindstedt, Lund University

Department of
Industrial Electrical Engineering and Automation
Lund University
Box 118
SE-221 00 LUND
SWEDEN

http://www.iea.lth.se

CODEN:LUTEDX/(TEIE-5219)/1-45/(2006)

Abstract

This Master's Thesis deals with distribution of alarms. Alarms are generated by peripheral automation products and sent to a central computer using Simple Mail Transfer Protocol (SMTP). An application running on the computer then distributes the alarms to operators using Short Message Service (SMS).

A solution using Nimbus Alarm Server by TroSoft HB has been tried and evaluated. Thereafter a new application called Pheidippides has been developed, with a somewhat different focus. Both solutions use the same peripheral automation equipment – the operator terminal series E-1000 from Beijer Electronics.

Much emphasis has been put into making documentation for setting up the solutions. The documentation is easy to understand and provides step-by-step instructions for setting up the applications, the operator terminal and Windows XP. The instructions cover communication between the terminal and computer over both Local Area Network (LAN) and modem using Point to Point Protocol (PPP) connection.

Acknowledgements

First, we would like to thank our supervisor Gustaf Olsson and examiner Gunnar Lindstedt at Lund University, for their enthusiasm and helpful advice during the development of this thesis.

Many thanks go to the people at Department 34 at Beijer Electronics who have gone out of their way to make our stay here a welcome one indeed. This especially applies to our supervisor Håkan Jeppson and also Christer Sturestam, whom we during these months have come to consider as a kind of mentor. We would also like to mention Göran Håkansson for giving us the opportunity to perform this job.

We would also like to thank Tomas Rook at TroSoft HB for the fast response he has given us. It was very valuable in the first stages of our work.

Thanks go to Christian Rosen at Lund University for providing us with the LaTeX template document we have used when writing the report. Christina Rentschler also deserves thanks for the quick and helpful assistance she has provided.

Malmö, 2006-02-17
Anders Hallberg
Markus Helgesson

Contents

Chapter 1

Introduction

1.1 Background

Beijer Electronics are often approached by their customers with questions regarding communication with their automation equipment. This involves the possibility to send alarm reports by Short Message Service (SMS) to the technical operator on call or to be able to interact with decentralised equipment from a remote computer through internet or connection over Point to Point Protocol (PPP).

Many of the pieces to make this work already exist, but there is a lack of a general overview. Therefore all parts need to be examined and documented in a step-by-step fashion. Special attention is to be given to the alarm distribution software, which may need to be modified or developed.

1.2 Scenario

In order to begin an analysis of the requirements, we consider the following scenario:

Not far from Oarp there is a pump station that supplies the immediate area with fresh water. The pump station is equipped with a number of Programmable Logic Controllers (PLC) controlled by an operator terminal. The operator terminal is connected to a modem with the possibilities of connecting to the internet or establishing a PPP-connection.

If an unexpected event occurs at the pump station the terminal connects to the internet and transmits a report with information about the event through e-mail to the operator on call. The operators take turn being on call one week at a time using a rolling schedule. Since the reports are sent through e-mail the operator on call has to constantly monitor his or her e-mail client. If the event occurring at the pump station requires any sort of action from the operator he or she has to physically get to Oarp. The limitations with this concept are obvious and directly affect the well being and general happiness of the operator currently on call.

If the operator instead gets an SMS with the alarm report he or she would be able to move freely as long as his or her cell phone has network coverage. If the event at the pump station doesn't require a manual intervention but simply an adjustment of some parameter it would be preferable if the operator could connect to the pump station remotely and change the parameters. Or why not even allow the operator to use a handheld device with General Packet Radio Service (GPRS) capabilities?

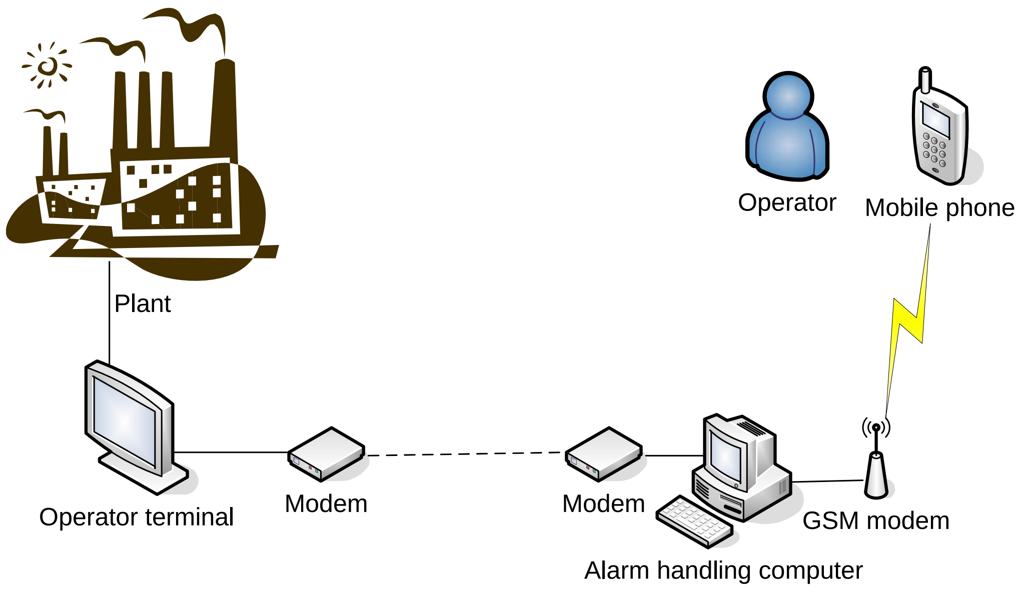


Figure 1.1: An unsupervised process is controlled using an operator terminal connected to a modem. When an alarm is generated the operator terminal dials up the alarm handling computer that distributes the alarm by SMS to the operator currently on call.

1.3 Assignment analysis

We have identified the following requirements in order to fulfil the scenario described in Section 1.2 in a fashion according to our and Beijer Electronics' expectations:

1. The operator terminal located in the pump station needs a way to communicate with the central computer. This can be done by modem, through either an internet connection or a PPP-connection.

2. It should be possible to use different communication alternatives including Local Area Network (LAN), Asymmetric Digital Subscriber Line (ADSL), GPRS, Global System for Mobile communications (GSM) and conventional modem.

3. The terminal must be capable of transmitting the alarms once a connection has been established with the central alarm handling computer.

4. An application capable of receiving the alarms and distributing these properly to the operators should be running on the central alarm handling computer. The application should distribute the alarms to the operator on call by SMS, according to a rotating schedule, using a GSM modem or an internet service. The operator must respond with an acknowledgement within a specified time or the alarm should be distributed to a wider group.

5. A remote user of the operator terminal should be able to make the same changes as a local user.

6. The communication between the remote operator and the terminal should be handled in a secure way, for example using tunnelling through a Virtual Private Network (VPN).

7. The central alarm handling computer should have a firewall, for increased security.

8. A document with detailed instructions of how to configure the hardware and software involved should be produced.

Since many of the raised issues are not new, some of the requirements are already fulfilled:

1. The operator terminal supports connection by Ethernet (the most common LAN type) or by using a modem connected to the serial interface for a PPP connection.

3. The operator terminal supports sending alarms by e-mail, using to the Simple Mail Transfer Protocol (SMTP).

5. It is possible to mirror the terminal with full functionality from a remote computer using the application Remote Access Viewer by Beijer Electronics AB.

Nimbus Alarm Server by TroSoft HB is an application that is able to handle distribution of alarms. It is mainly designed to be integrated with somewhat larger automation facilities using a Supervisory Control and Data Acquisition (SCADA) system. A small business typically wants a simpler application with a feature set designed for somewhat different needs, as well as a lower price range.

The assignment will therefore be divided into two parts. In the first we produce a solution based on Nimbus Alarm Server and in the second we develop a new application that Beijer Electronics AB will provide to their customers free of charge.

In the first part of the assignment we want to complete the following paragraphs with the use of Nimbus Alarm Server:

2. Evaluation of different communication alternatives including LAN, GSM modem and conventional modem.

6. Evaluation of security alternatives in hardware and software, mostly with the use of a VPN.

7. Creation of instructions for adapting the Windows XP firewall for running Nimbus Alarm Server.

8. Production of a document with complete instructions for setting up and maintaining the alarm distribution. The program RoboHelp will be used to produce the document.

In the second part we will develop a new application fulfilling the requirements stated in paragraph 4. Documentation should also be produced in the same manner as with the first part. The developed application should...

- Receive alarms by SMTP from Beijer Electronics operator terminal E-1000 family.
- Distribute alarms to operators by SMS.
- Use a rotating schedule to determine which operator should receive the SMS.
- Handle acknowledgements from the operators.
- Present information in a user friendly interface.
- Log all occurred events in a persistent way.

1.4 Outline of the report

In Chapter 1 we introduce the problem scenario with an analysis. In Chapter 2 we give a brief description of Bejier Electronics AB and the peripheral equipment utilised by our solution. A solution using Nimbus Alarm Server is described in Chapter 3. We have also made an alternative solution using our own-developed application Pheidippides, which is described in Chapter 4. Finally we present our results and conclusions in Chapter 5.

Chapter 2

Beijer Electronics AB

2.1 General

Beijer Electronics AB is a development- and agency corporation within the industrial automation area. The corporation currently has around 200 employees and is divided into two business areas - Automation and HMI Products. Automation offers agency products from leading international suppliers as well as a complete range of automation system solutions. Its focus is on the Nordic market, i.e. Sweden, Norway and Denmark. HMI Products develops and sells operator terminals world wide. For more information visit www.beijer.se.

2.2 Products

We now introduce a few products from Beijer Electronics that will be of relevance to our work.

2.2.1 E-1000 operator terminal family

Figure 2.1: The E-1000 operator terminal family from Beijer Electronics AB.

The E-1000 family is the latest addition to Beijer Electronics E-series. It has high performance and many communication alternatives, including hardware connections for Ethernet, RS-232 and USB.

The software, which runs on top of Microsoft Windows CE, supports Simple Mail Transfer Protocol (SMTP), Hyper Text Transfer Protocol (HTTP) and File Transfer Protocol (FTP).

2.2.2 GSM Modems

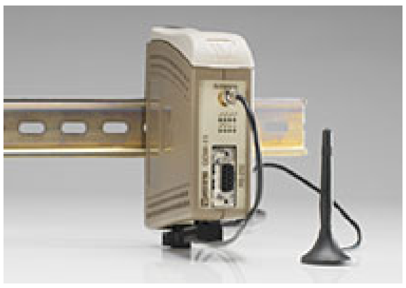


Figure 2.2: A GDW-11 GSM modem from Westermo.

GD-01 and GDW-11 are two Global System for Mobile communication (GSM) modems developed by Westermo for use in an industrial environment. A GSM-modem has the benefit of making it possible for equipment to communicate with the surrounding world as long as it is placed within the coverage of a GSM-operator. It can also be used for the purpose of sending and receiving Short Message Service (SMS) messages.

2.2.3 Conventional modems

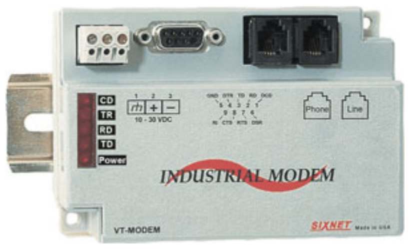


Figure 2.3: A VT-Modem from Sixnet.

Sixnet has a series of industry adapted modems called VT-Modems. They have a variety of features, but can all be attached to a E-1000 terminal or a typical computer by the RS-232 serial interface.

Chapter 3

Phase 1 - A solution using Nimbus Alarm Server

3.1 Work process

When we first began working on this Master's Thesis, much was still unclear regarding what was to be done. The first weeks we spent familiarising ourselves with Beijer Electronics products and trying to understand what was possible to do with them when it came to alarm management. We completed a course in programming the E-1000 terminal with E-Designer and also got an introduction to programming Programmable Logic Controllers (PLC) with GX IEC Developer, which eased this in some way.

We experimented with Nimbus Alarm Server and began setting up a few different scenarios using a E-1070 operator terminal and Nimbus Alarm Server communicating over Local Area Network (LAN). Tomas Rook at TroSoft HB were of great help to us, assisting us with license keys and overnight bug fixes.

When a working solution was achieved over LAN we started working with a modem connection over Point to Point Protocol (PPP) instead. This meant more experimenting with the E-1070 terminal and working to set up Windows XP as an incoming PPP-server. Some effort was spent getting the Global System for Mobile communications (GSM) modem up and running.

We also looked at the security part. The firewall that is integrated in Windows XP SP2 was set up and used. When it came securing the communication we noted that this can be achieved either by using a hardware firewall with Virtual Private Network (VPN) capabilities or by extending the E-1070 software to provide a software VPN. Looking further at either alternative would be the subject of a Master's Thesis in its own right and therefore out of the scope for our work.

The next step was to write the step-by-step documentation, which we did with the help of RoboHelp. We also started designing our own application, which we quite promptly named Pheidippides after the famous runner who carried the news of the Greek victory at the battle of Marathon (490 BC). The development process is described in Section 4.4

A little documentation work was done in parallel with the development, but the main effort with this report and the help documentation for Pheidippides was done after the development process was completed.

3.2 E-1070 operator terminal

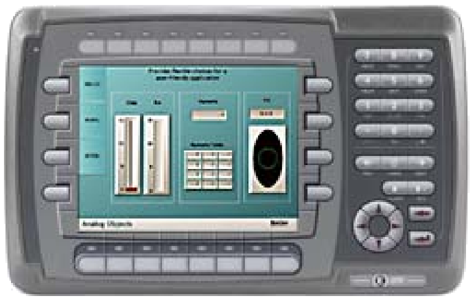


Figure 3.1: The E-1070 operator terminal from Beijer Electronics.

The E-1070 can be configured to trigger a user defined alarm in the case a digital signal gets set or an analogue signal goes above or below a certain level. An alarm has an alarm text and belongs to an alarm group. The groups can, for instance, be used to order the alarms in priority groups. The alarm also has a status which can be either active or inactive. It can also be manually acknowledged by an operator, which can be seen as a third status.

When an alarm is triggered it shows up in the alarm list on the terminal. There one can easily see the current alarms and whether they are active or inactive, and if they have been acknowledged. The operator can acknowledge alarms and remove old alarms from the list.

The terminal can also be set to send an e-mail when an alarm is triggered or changes status. It simply connects to a predefined Simple Mail Transfer Protocol (SMTP) server when it is time. If the SMTP server is unreachable for some reason the terminal has a buffer it uses to store the e-mail until they can be sent. All information is sent in the subject field of the e-mail, the body is not used. The subject is of the format "YY-MM-DD [h]h:mm:ss status alarm-group alarm-text", where status can be "*ALARM", "inact" or "ack". Two examples are "05-11-25 2:48:03 *ALARM A-LARM Something dreadful has happened to the shiny thingy" and "06-02-20 15:04:27 ack B-LARM Low level in secondary tank".

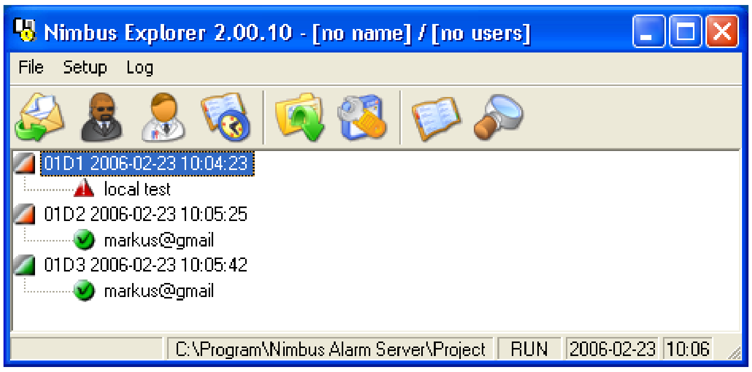
In most cases the terminal is not likely to have a permanent internet connection. Because of this there is an option to establish a PPP connection over a modem attached to the terminal. This can be done on demand each time there is an e-mail to send.

3.3 Windows XP

Windows XP can quite easily be set up to act as a server for incoming PPP dial ins simply by using the provided wizard. Since Service Pack 2, the integrated firewall in Windows XP is enabled by default and therefore has to be set up to allow for the SMTP communication. It is definitely recommended to use the firewall and allow only for the specific requested communication rather than simply disabling it. This gives quite a large improvement regarding the computer security on the server side.

3.4 Nimbus Alarm Server

Nimbus Alarm Server has been developed by TroSoft HB with integration towards Supervisory Control and Data Acquisition (SCADA) systems in mind. It has a large compatibility list and is able to receive alarms from many different sources in several formats. This includes the E-family operator series through SMTP, which is the only combination we have spent any time on.



01D1 2006-02-23 10:04:23
local test
01D2 2006-02-23 10:05:25
markus@gmail
01D3 2006-02-23 10:05:42
markus@gmail

C:\Program\Nimbus Alarm Server\Project RUN 2006-02-23 10:06

Figure 3.2: Nimbus Alarm Server is an advanced application for alarm distribution. The user interface may be somewhat lacking though.

When an alarm is received it is handled differently depending on what profiles are currently active. A profile is set to be active at certain times, for instance weekdays 7 am to 4 pm, and can include filters that only look for certain alarms, for instance from a specific alarm source. The profile then has connected receivers that the alarms are sent to. There are many possible types of receivers, including e-mail and SMS.

There are also more advanced features, including the possibility of synchronizing with a SCADA system, a valuable feature in a large facility.

Chapter 4

Phase 2 - Developing Pheidippides

4.1 Introduction

In the second part of the Master's Thesis an alarm distribution application named Pheidippides was developed. Compared to the Nimbus Alarm Server (see Section 3.4) Pheidippides is a less general alarm distribution program. Pheidippides is specialised at receiving alarms from the E-1000 operator terminal family and distributing by SMS (Short Message System) using a GSM-modem (Global System for Mobile communications).

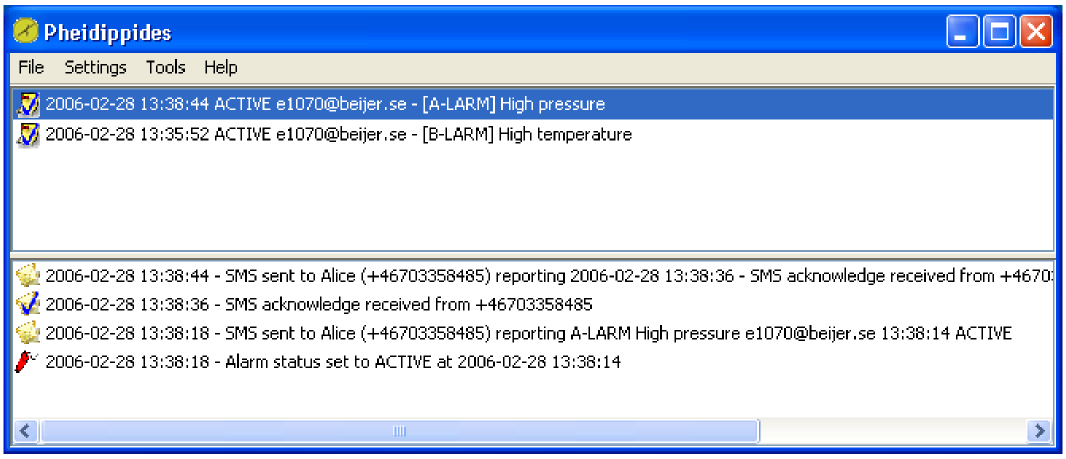
4.2 Requirements

The requirements for the program are the following:

- Receive alarms by SMTP (Simple Mail Transfer Protocol) from Beijer Electronics operator terminal E-1000 family.
- Distribute alarms to operators by SMS.
- Use a rotating schedule to determine which operator should receive the SMS.
- Handle acknowledgements from the operators.
- Present information in a user friendly interface.
- Log all occurred events in a persistent way.

4.2.1 The rules of distribution

The E-1000 terminal informs when an alarm is active, inactive and acknowledged. The alarm distribution program is required to keep distributing alarms to an increasing amount of recipients from a schedule, until the alarm is acknowledged. This acknowledgement may come from an operator at the E-1000 terminal or by SMS from the operator on call. The distribution is expanded if a certain amount of time has passed without an acknowledgement either by SMS or from the terminal. If the time passes without acknowledgement the next recipient in the schedule is sent the alarm.



2006-02-28 13:38:44 ACTIVE e1070@beijer.se - [A-LARM] High pressure
2006-02-28 13:35:52 ACTIVE e1070@beijer.se - [B-LARM] High temperature

2006-02-28 13:38:44 - SMS sent to Alice (+46703358485) reporting 2006-02-28 13:38:36 - SMS acknowledge received from +4670
2006-02-28 13:38:36 - SMS acknowledge received from +46703358485
2006-02-28 13:38:18 - SMS sent to Alice (+46703358485) reporting A-LARM High pressure e1070@beijer.se 13:38:14 ACTIVE
2006-02-28 13:38:18 - Alarm status set to ACTIVE at 2006-02-28 13:38:14

Figure 4.1: Pheidippides' main window shows the currently open alarm occasions.

When an acknowledgement is received a message is sent to all recipients involved, reporting who has acknowledged the alarm.

An alarm occurrence is considered completely finished when the alarm is inactive and acknowledged.

4.3 Structure overview

The development of Pheidippides resulted in a final iteration with an interaction between packages written especially for Pheidippides as well as third-party packages solving specific tasks. The packages are described in detail in Section 4.5 and the major components are displayed in Figure 4.2.

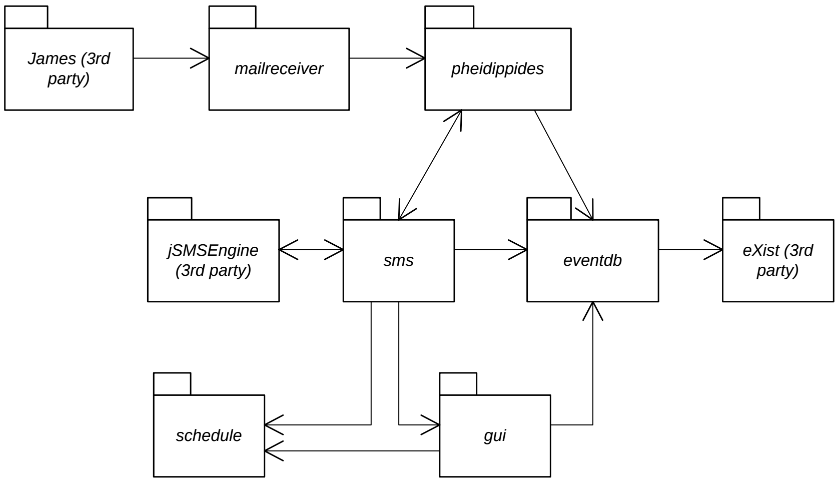


Figure 4.2: A diagram displaying the relations between the major packages in the structure of Pheidippides.

The core functions of the alarm distribution program are the receiving of alarms by SMTP and the distribution of alarm reports by SMS. For these tasks third-party packages have been utilised. To receive the e-mails the Apache project *James* is used as mail server. The distribution of SMS is handled by the *jSMSEngine* package.

To make the incoming e-mails available for processing in the application the *mailreceiver* package is used. The responsibility of the *mailreceiver* package is to adapt the incoming alarm to an object filled with the alarm information. The alarm objects are delivered to the main package *pheidippides*.

The major responsibility of the main package, *pheidippides*, is to initiate the other packages at the start up of the application Pheidippides. Objects with incoming alarms and incoming SMS are delivered to *pheidippides* who direct the objects to the *eventdb* package for storage. A notification is sent to the *sms* package informing there are new alarms available for distribution.

Objects representing all incoming alarms, outgoing SMS and incoming SMS are stored using the *eventdb* package. Alarm sequences that are closed are written to persistent storage using the third-party XML (Extensible Markup Language) database package *eXist*.

The sending and receiving of sms is handled by the *sms* package. The *sms* package checks *eventdb* for new alarms and alarms that need redistribution. The recipient of the SMS is requested from *schedule* package. Any incoming SMS is sent as an object to the *pheidippides* package. The actual communication with the GSM-modem is handled by the third-party *jSMSEngine* package.

The schedule for the operators on call are set in the *schedule* package. The *schedule* package uses a rotating schedule where operators are assigned a week respectively in a cyclic manner.

The *gui* package handles the interaction with the user. The contents of the *eventdb* package is displayed with the main view presenting the currently active or unacknowledged alarms (see Figure 4.1). A view of the history of passed alarms is also accessible. Settings for the *sms* package and the *schedule* of recipients are altered using the *gui* package.

The sequence diagrams in Figure 4.3 and Figure 4.4 display the flow of information and the interaction between packages in the event of an incoming alarm and an incoming SMS acknowledgement from the operator.

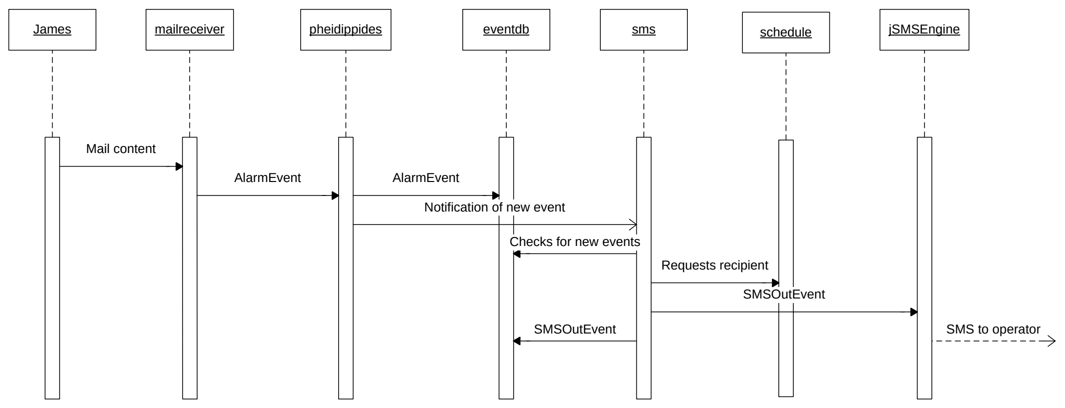


Figure 4.3: A sequence diagram displaying the interaction between the packages as an alarm is received.

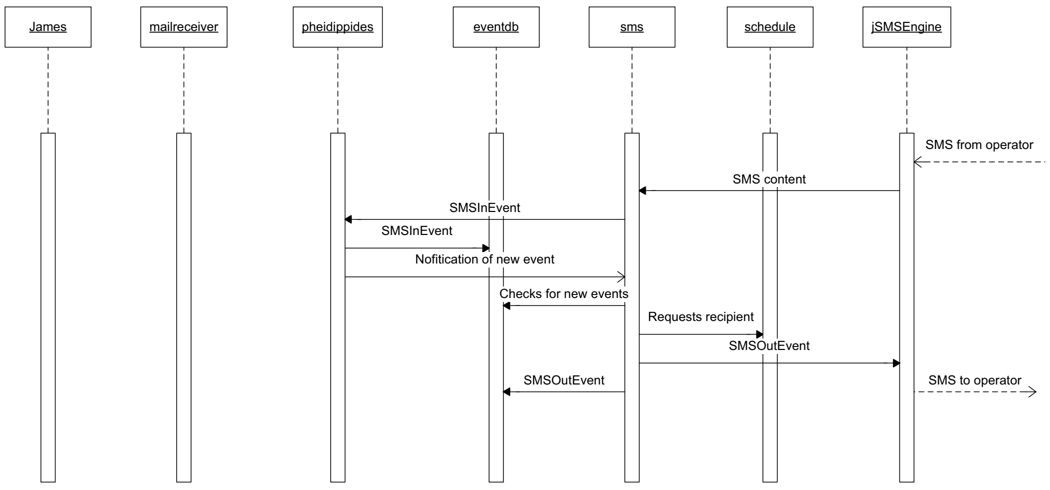


Figure 4.4: A sequence diagram displaying the interaction between the packages as an SMS is received.

4.4 Development process

4.4.1 Introduction

In the process of the development of Pheidippides we have used an approach inspired by the concept of Extreme Programming[1] (XP). Progress has been made in small iterations where continuous planning and design has been the key. The following steps describe the progress of the program.

4.4.2 Preparations

As the very first step in our development of Pheidippides we had to decide on a programming language. A request from our company supervisor was to make the program platform independent. This request led us in the direction of Java which has been the main language during our education at Lund University.

Before making a final decision regarding programming language we needed to test whether the basic requirements were able to be met using Java. The core functions of the application are to receive alarms distributed as e-mails and deliver these as SMS text messages. Therefore we must have some kind of mail server as well as some mean of communication with a GSM-modem.

For the mail server we decided to go for an existing solution instead of coding one ourselves. After looking around for different alternatives we settled for the Apache James server (see Section 4.6.1), an open source project written in Java. It supports interaction with external applications and is therefore perfect for our needs.

The GSM-modem we use is connected to the computer using a RS-232 serial port. We tested a Java extension package for communication over a serial port which got the job done. A slight disadvantage for the platform independency was discovered as the package is differently installed on

[1]See www.extremeprogramming.org for more information

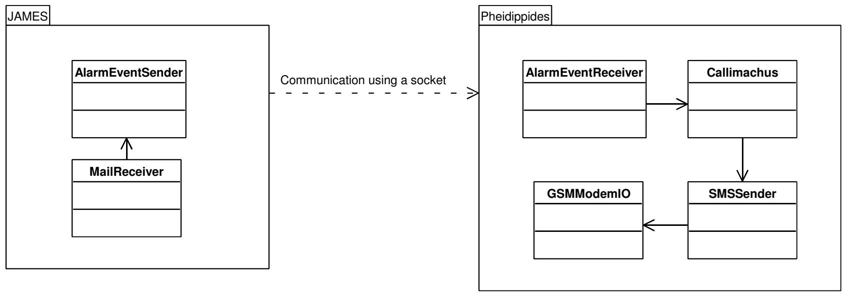


Figure 4.5: Structure of the first iteration. The mail is received by the James mail server and delivered to an instance of the MailReceiver class. The MailReceiver adapts the mail to an object which is serialized and transferred using sockets to our main program Pheidippides. The class Callimachus delivers the mail to the mailbox of the SMSSender. The SMSSender constantly checks the mailbox and delivers any incoming alarms as SMS to a predefined number.

different platforms, and therefore is not completely integrated with our application.

Too simplify development in Java, different integrated development environments (IDE) were tested. NetBeans[2] came out on top with the major advantage of a good GUI editor. Normally it takes a lot of effort to specify the location of GUI components. In NetBeans components are placed using a drag and drop environment which is a great asset.

4.4.3 First iteration

By now we were convinced that we would be able to create the application using Java. We made a minimalistic design which would be able to receive an e-mail and send an SMS to a static number. The design was a simple producer / consumer relation where we used mailboxes as buffers. We chose to represent the threads with their mailboxes using the Lund Java-based Real-Time (LJRT) library (see Section 4.6.4).

The mail server and our application ran in different virtual machines and therefore the communication between the programs was handled using sockets. For the sending of SMS our intention was to write that part ourselves. But when we stumbled upon the jSMSEngine package (see Section 4.6.2) which handles sending and receiving of SMS, we decided to go for that solution. At this stage we started logging to simplify debugging, using Log4j (see Section 4.6.5).

We were able to complete a program that would receive alarms and perform the distribution, without any fancy processing.

4.4.4 Second iteration

The distribution of alarms is dictated by a set of rules specified in Section 4.2. The distribution is depending on previous events and therefore we need to store information of all incoming events. We designed a database where the alarms would be stored and sorted according to which terminal

[2]See http://www.netbeans.org/ for more information

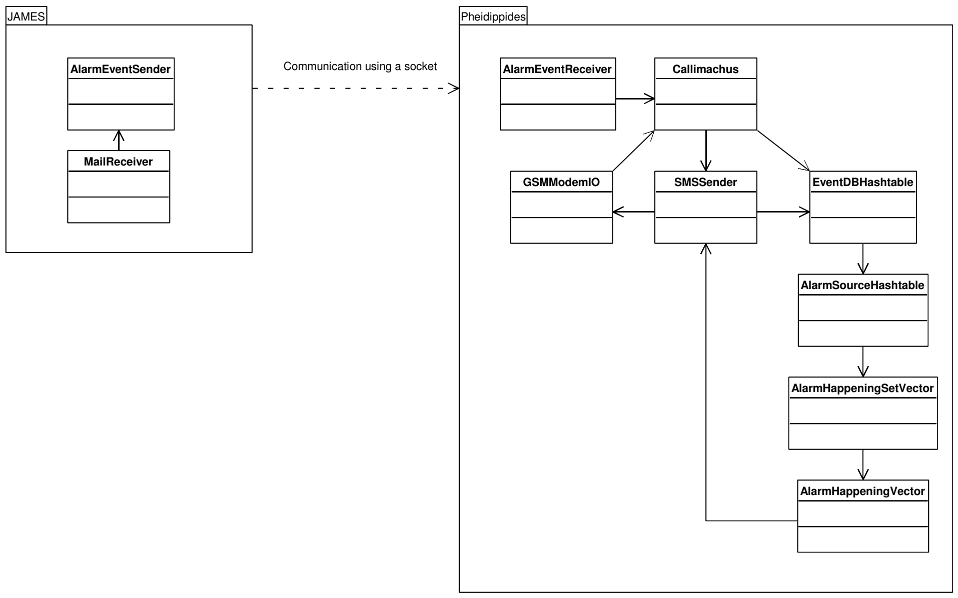


Figure 4.6: Structure of the second iteration. Each incoming event is added to the database. The database is constructed of a hash table with the different alarm sources as an entry level. Each alarm source has a hash table with the different types of alarms from that specific source. Each alarm type keeps track of the different occurrences of that specific alarm. Each alarm occurrence stores the events associated with that specific occurrence.

they were sent from, the text describing the alarm and the different occurrences of the alarms with the same source and text. A GUI was implemented displaying the incoming alarms, reports of sent SMS messages as well as incoming SMS messages, in a tree structure.

With the database structure in place it was time for more designing. The design for redistribution of unacknowledged alarms was set as well as the design for the schedule. For the schedule we aimed high with an implementation with great flexibility based on recurring events using the project ChronicJ[3]. The problem with this approach was making the schedule maintainable for the user. The GUI for the schedule would not be user friendly unless we would remove a magnitude of the features in the ChronicJ project. After discussion with our supervisor we settled for a rotating schedule where a list of operators were assigned a week each repeatedly, leaving ChronicJ out altogether. The schedule was also blessed with a GUI (Graphical User Interface).

4.4.5 The third and final iteration

The application was demonstrated for our supervisors from Lund University and Beijer Electronics AB. A list of possible improvements was made which were ranked by the company supervisor based on necessity as follows:

- Saving history to disk and the possibility of viewing the history in a structured manner.

[3]See http://chronicj.org/ for more information

- The possibility of easily disabling SMS-sending. Used at process start ups where many alarms may be triggered in a controlled environment.
- A function for verifying that Pheidippides is up and running. Remotely checked by sending an SMS to Pheidippides and getting a status message in response or perhaps a periodical status message is sent from Pheidippides to the operator on call.

We had the time to complete the first one. This request led to some refactoring of our design. The tree structure in the database was replaced with a vector storing the active alarm happenings. When the happening is closed it is passed on to the eXist database. Using the database we could display requested parts of the history in a tree, structured very much like our previous GUI. The main GUI was altered to only show active or unacknowledged alarms in a list.

In previous iterations the James mail server and Pheidippides have run in different Java Virtual Machines. The drawback with this approach is that we must have two separate applications running in our alarm program. The remedy for the final iteration was to have Pheidippides initiated by James using the `MailReceiver` class. When James is invoked it calls an initialisation method in `MailReceiver`, placing them both in the same JVM. Therefore we no longer need to serialise our `AlarmEvent` and transmit using sockets. The `MailReceiver` posts the `AlarmEvent` straight into the mailbox of `Challimachus`, making `AlarmEventSender` and `AlarmEventReceiver` obsolete.

To store settings we used Preferences which is native Java. A tool for simulating incoming alarms was developed. It will make it easier for the user to verify that Pheidippides is properly installed without having to cause any alarms in the process.

See Figure 4.7 for the complete class diagram of the final iteration.

4.5 Pheidippides packages

4.5.1 se.beijer.pheidippides

The main package which contains the core files.

`Pheidippides`
: Contains the main method whose only function is to inform a caller on how to properly initiate the program by invoking James.

`Callimachus`
: This class is responsible for initiating and terminating critical parts of the program, as the `GUI`, `SMSSender` and the `EventDB`. Incoming events from `MailReceiver` and the `GSMModemIO` are posted in Callimachus' mailbox. Callimachus redirects them to the `EventDB` and notifies the `SMSSender` that new events have arrived. Callimachus is a `JThread` from the Lund Java-based Real-Time package (see Section 4.6.4).

4.5.2 se.beijer.pheidippides.eventdb

This is the database package for storage of all events that are sent during the execution of Pheidippides. Communication in Pheidippides is mainly done by sending a `PEvent`. Different actions are then undertaken depending on what event is sent.

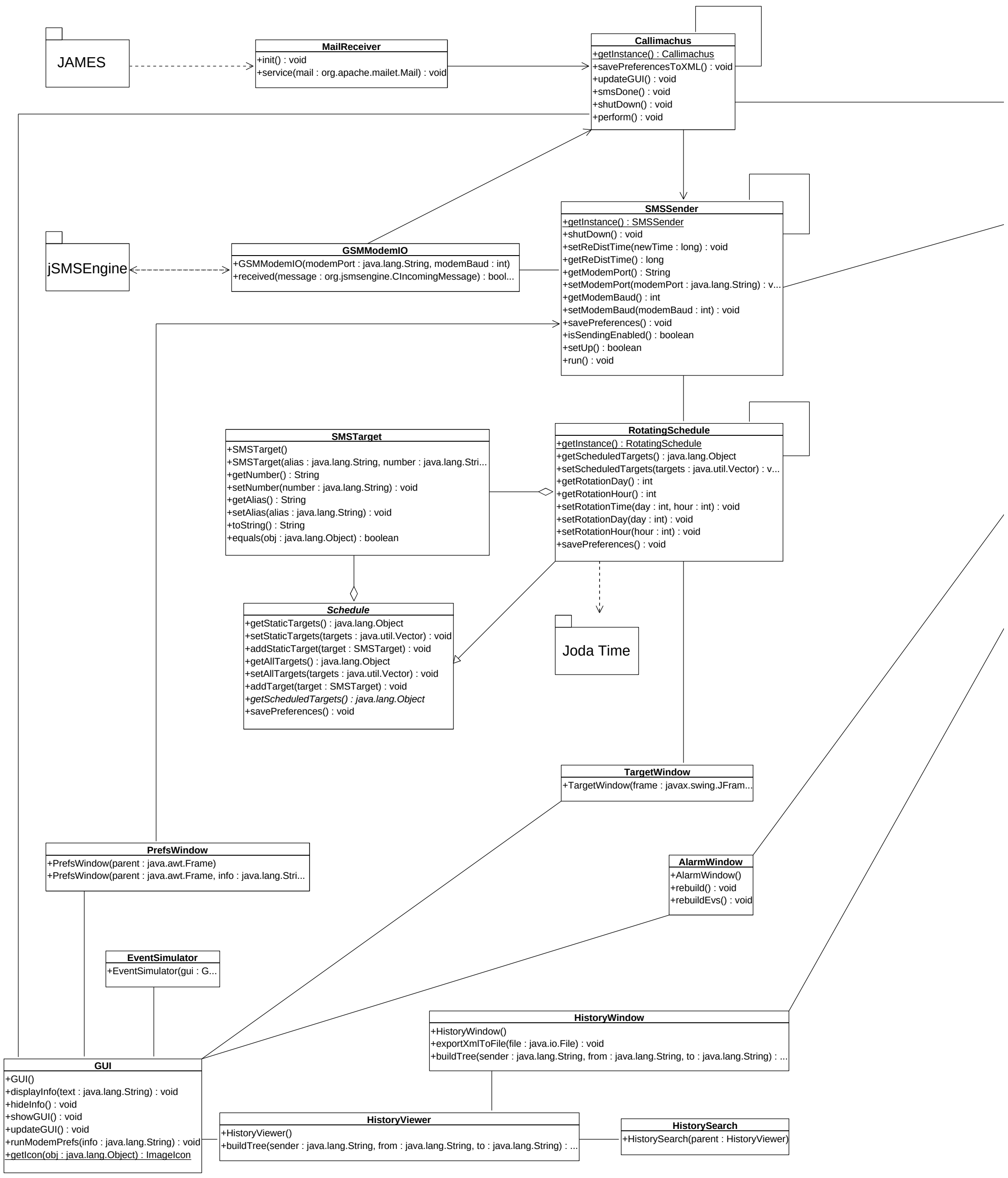
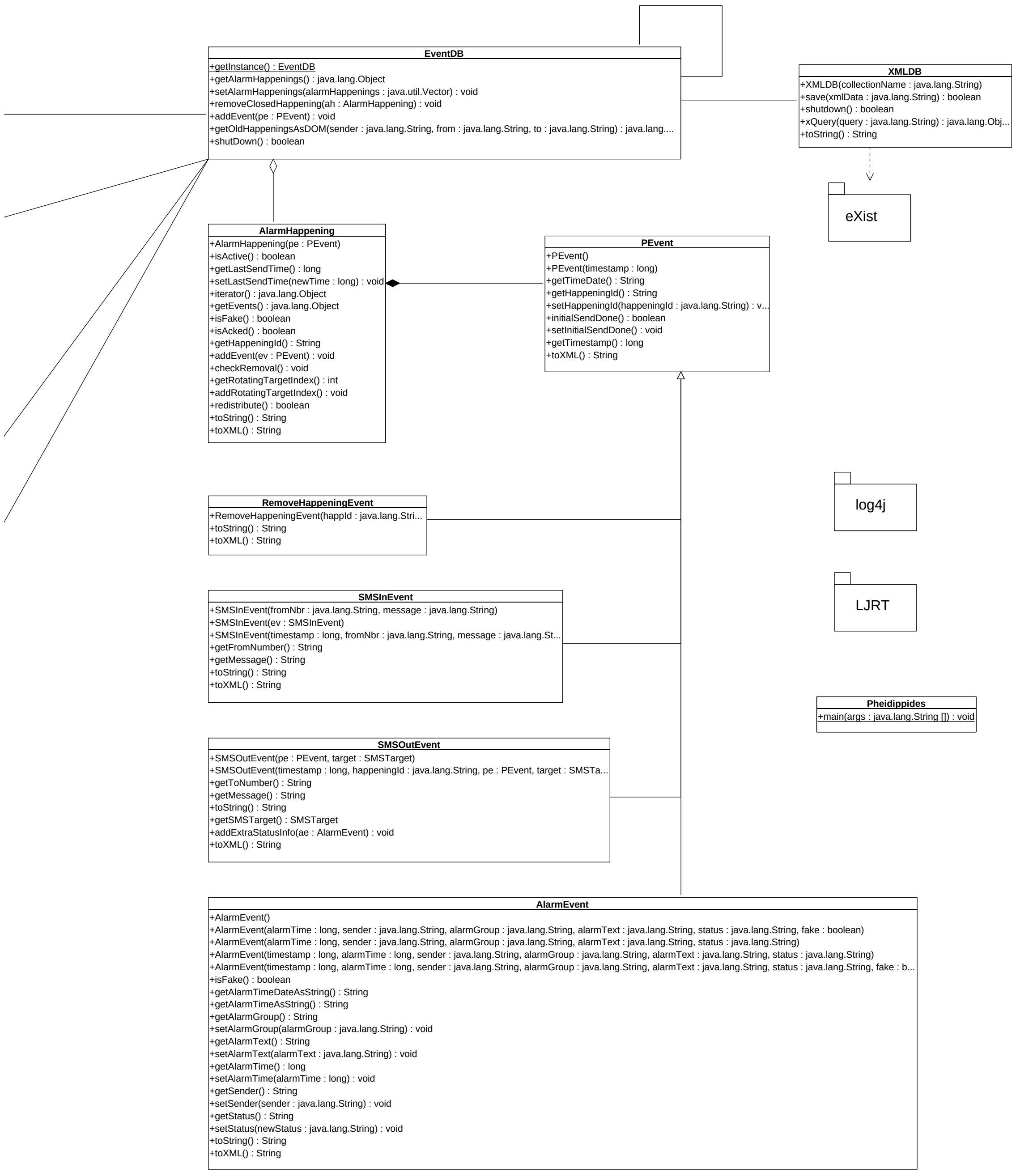

JAMES
MailReceiver
+init() : void
+service(mail : org.apache.mailet.Mail) : void
Callimachus
+getInstance() : Callimachus
+savePreferencesToXML() : void
+updateGUI() : void
+smsDone() : void
+shutDown() : void
+perform() : void
SMSSender
+getInstance() : SMSSender
+shutDown() : void
+setReDistTime(newTime : long) : void
+getReDistTime() : long
+getModemPort() : String
+setModemPort(modemPort : java.lang.String) : v...
+getModemBaud() : int
+setModemBaud(modemBaud : int) : void
+savePreferences() : void
+isSendingEnabled() : boolean
+setUp() : boolean
+run() : void
jSMSEngine
GSMModemIO
+GSMModemIO(modemPort : java.lang.String, modemBaud : int)
+received(message : org.jsmsengine.CIncomingMessage) : bool...
SMSTarget
+SMSTarget()
+SMSTarget(alias : java.lang.String, number : java.lang.Stri...
+getNumber() : String
+setNumber(number : java.lang.String) : void
+getAlias() : String
+setAlias(alias : java.lang.String) : void
+toString() : String
+equals(obj : java.lang.Object) : boolean
RotatingSchedule
+getInstance() : RotatingSchedule
+getScheduledTargets() : java.lang.Object
+setScheduledTargets(targets : java.util.Vector) : v...
+getRotationDay() : int
+getRotationHour() : int
+setRotationTime(day : int, hour : int) : void
+setRotationDay(day : int) : void
+setRotationHour(hour : int) : void
+savePreferences() : void
Schedule
+getStaticTargets() : java.lang.Object
+setStaticTargets(targets : java.util.Vector) : void
+addStaticTarget(target : SMSTarget) : void
+getAllTargets() : java.lang.Object
+setAllTargets(targets : java.util.Vector) : void
+addTarget(target : SMSTarget) : void
+getScheduledTargets() : java.lang.Object
+savePreferences() : void
Joda Time
TargetWindow
+TargetWindow(frame : javax.swing.JFram...
PrefsWindow
+PrefsWindow(parent : java.awt.Frame)
+PrefsWindow(parent : java.awt.Frame, info : java.lang.Stri...
AlarmWindow
+AlarmWindow()
+rebuild() : void
+rebuildEvs() : void
EventSimulator
+EventSimulator(gui : G...
HistoryWindow
+HistoryWindow()
+exportXmlToFile(file : java.io.File) : void
+buildTree(sender : java.lang.String, from : java.lang.String, to : java.lang.String) : ...
GUI
+GUI()
+displayInfo(text : java.lang.String) : void
+hideInfo() : void
+showGUI() : void
+updateGUI() : void
+runModemPrefs(info : java.lang.String) : void
+getIcon(obj : java.lang.Object) : ImageIcon
HistoryViewer
+HistoryViewer()
+buildTree(sender : java.lang.String, from : java.lang.String, to : java.lang.String) : ...
HistorySearch
+HistorySearch(parent : HistoryViewer)

Figure 4.7: UML class diagram for Pheidippides.


EventDB
+getInstance() : EventDB
+getAlarmHappenings() : java.lang.Object
+setAlarmHappenings(alarmHappenings : java.util.Vector) : void
+removeClosedHappening(ah : AlarmHappening) : void
+addEvent(pe : PEvent) : void
+getOldHappeningsAsDOM(sender : java.lang.String, from : java.lang.String, to : java.lang.String) : java.lang....
+shutDown() : boolean
XMLDB
+XMLDB(collectionName : java.lang.String)
+save(xmlData : java.lang.String) : boolean
+shutdown() : boolean
+xQuery(query : java.lang.String) : java.lang.Obj...
+toString() : String
eXist
AlarmHappening
+AlarmHappening(pe : PEvent)
+isActive() : boolean
+getLastSendTime() : long
+setLastSendTime(newTime : long) : void
+iterator() : java.lang.Object
+getEvents() : java.lang.Object
+isFake() : boolean
+isAcked() : boolean
+getHappeningId() : String
+addEvent(ev : PEvent) : void
+checkRemoval() : void
+getRotatingTargetIndex() : int
+addRotatingTargetIndex() : void
+redistribute() : boolean
+toString() : String
+toXML() : String
PEvent
+PEvent()
+PEvent(timestamp : long)
+getTimeDate() : String
+getHappeningId() : String
+setHappeningId(happeningId : java.lang.String) : v...
+initialSendDone() : boolean
+setInitialSendDone() : void
+getTimestamp() : long
+toXML() : String
log4j
RemoveHappeningEvent
+RemoveHappeningEvent(happId : java.lang.Stri...
+toString() : String
+toXML() : String
LJRT
SMSInEvent
+SMSInEvent(fromNbr : java.lang.String, message : java.lang.String)
+SMSInEvent(ev : SMSInEvent)
+SMSInEvent(timestamp : long, fromNbr : java.lang.String, message : java.lang.St...
+getFromNumber() : String
+getMessage() : String
+toString() : String
+toXML() : String
Pheidippides
+main(args : java.lang.String []) : void
SMSOutEvent
+SMSOutEvent(pe : PEvent, target : SMSTarget)
+SMSOutEvent(timestamp : long, happeningId : java.lang.String, pe : PEvent, target : SMSTa...
+getToNumber() : String
+getMessage() : String
+toString() : String
+getSMSTarget() : SMSTarget
+addExtraStatusInfo(ae : AlarmEvent) : void
+toXML() : String
AlarmEvent
+AlarmEvent()
+AlarmEvent(alarmTime : long, sender : java.lang.String, alarmGroup : java.lang.String, alarmText : java.lang.String, status : java.lang.String, fake : boolean)
+AlarmEvent(alarmTime : long, sender : java.lang.String, alarmGroup : java.lang.String, alarmText : java.lang.String, status : java.lang.String)
+AlarmEvent(timestamp : long, alarmTime : long, sender : java.lang.String, alarmGroup : java.lang.String, alarmText : java.lang.String, status : java.lang.String)
+AlarmEvent(timestamp : long, alarmTime : long, sender : java.lang.String, alarmGroup : java.lang.String, alarmText : java.lang.String, status : java.lang.String, fake : b...
+isFake() : boolean
+getAlarmTimeDateAsString() : String
+getAlarmTimeAsString() : String
+getAlarmGroup() : String
+setAlarmGroup(alarmGroup : java.lang.String) : void
+getAlarmText() : String
+setAlarmText(alarmText : java.lang.String) : void
+getAlarmTime() : long
+setAlarmTime(alarmTime : long) : void
+getSender() : String
+setSender(sender : java.lang.String) : void
+getStatus() : String
+setStatus(newStatus : java.lang.String) : void
+toString() : String
+toXML() : String

`EventDB`
: EventDB keeps track of the active and unacknowledged alarm occurrences in a vector of `AlarmHappening`. The EventDB is supplied with `PEvents` which are added to their corresponding `AlarmHappening`. When a happening is closed it is passed on to XMLDB. After each incoming event the vector of active happenings is serialised and stored to disk to prevent loss of information in a crash. When Pheidippides is started the vector from the previous run is loaded.

`AlarmHappening`
: This class handles `PEvents` associated with a certain occurrence of an alarm, ranging from the first activity of the alarm until the alarm is inactive and acknowledged. It keeps track of the state of the alarm and whether the alarm has been acknowledged by the SMS recipients or from the terminal. It keeps track of the last time the alarm information was sent to a recipient which is used for deciding if the alarm information is to be sent to a new recipient, required no acknowledgement has been received.

`XMLDB`
: The database of closed happenings is maintained using this wrapper class. It uses `eXist` which is an open source XML database. The closed happenings are stored to disk and can be accessed using the GUI for the history.

4.5.3 se.beijer.pheidippides.gui

The GUI package quite simply contains the Graphical User Interface for Pheidippides.

`GUI`
: This is the class in the package that displays the main frame with the upper panel of options as well as the `AlarmWindow`. From the upper panel we can access the other components of the package.

`AlarmWindow`
: A part of the main GUI. Displays the active alarms stored in the `EventDB`.

`TargetWindow`
: This view is used for adding recipients to the static and rotating schedule. There are fields to set the time to wait for acknowledgement before redistributing as well as the time of the week to rotate the schedule.

`PrefsWindow`
: If the program fails to connect to the modem at start up, this window will appear where the user can change the modem settings. It can also be accessed from the main GUI frame. It interacts with `SMSSender` by setting the modem connection port and the connection speed before attempting to connect to the modem.

`HistoryViewer`
: The main history frame displays the menu options as well as the `HistoryWindow`. The

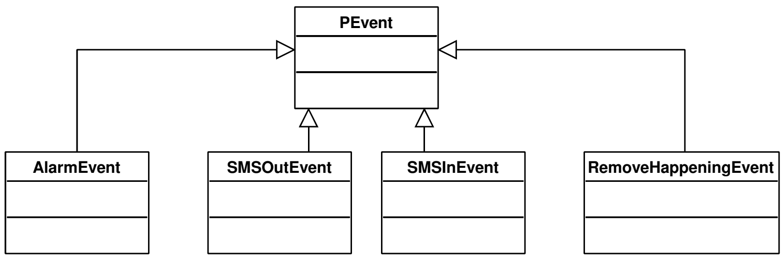


Figure 4.8: Structure of the package se.beijer.pheidippides.pevents

options we have are doing a search in our database and exporting the specific search to an XML file.

`HistorySearch`
When we wish to make a search in our history database we use the window HistorySearch to set our parameters. Searches can be made on the source of alarms or on which time period the alarms have occurred. As a search has been requested the parameters are passed on to the `HistoryViewer` who passes the information to the `HistoryWindow`.

`HistoryWindow`
HistoryWindow requests the search from `EventDB`, with the parameters from the `HistorySearch`. The result is built into a JTree with the different alarm occurrences sorted on alarm source and alarm description. HistoryWindow can also store the current search in an XML file.

`EventSimulator`
This is a feature for testing the installation of Pheidippides. Here we can fake alarms to make sure the distribution works as intended.

4.5.4 se.beijer.pheidippides.mailreceiver

The mailreceiver package is used by the third-party package James when it receives a mail.

`MailReceiver`
During normal operation MailReceiver is invoked by `James` when a mail arrives. MailReceiver then processes the mail, makes a corresponding `AlarmEvent` which is sent to `Callimachus` in se.beijer.pheidippides. MailReceiver has another important task at the start up of Pheidippides. The whole application is started by initiating James who in turn calls the init-method in MailReceiver where `Callimachus` is initiated.

4.5.5 se.beijer.pheidippides.pevents

Contains all events used for communication and data handling in Pheidippides.

`PEvent`
: This is the super class used for the event-based communication in Pheidippides. It is not meant to be used directly.

`AlarmEvent`
: Describes an event related to an alarm occurring, e.g. on a terminal. It indicates either the triggering, inactivation or the acknowledgement of an alarm. From the AlarmEvent we can extract a description of the alarm, the time the event has occurred as well as the source of the alarm.

`SMSInEvent`
: Describes a received SMS. The only current scenario for such an event is from an operator as confirmation of received alarms.

`SMSOutEvent`
: Describes an outgoing SMS sent from Pheidippides to an operator.

`RemoveHappeningEvent`
: This event is used when the user wishes to close an open `AlarmHappening` manually.

4.5.6 se.beijer.pheidippides.schedule

The alarms are distributed to operators on call. In this package the schedule for the operators is implemented. An operator is represented by a `SMSTarget` from the package se.beijer.pheidippides.sms.

`RotatingSchedule`
: Describes a schedule with a simple weekly rotation. The default rotation occurs Monday 00:00, that is once a week at midnight between Sunday and Monday. The implementation of the schedule utilises the third-party package Joda-Time (see Section 4.6.6).

`Schedule`
: Abstract super class to be used for implementing new schedules.

4.5.7 se.beijer.pheidippides.sms

Takes care of the sending and receiving of SMS. It utilises the third-party package jSMSEngine (see Section 4.6.2) for the communication with the modem.

`GSMModemIO`
: Handles the communication with the GSM modem by implementing an abstract class in jSMSEngine. It listens for incoming SMS and sends them to `Callimachus`.

`SMSTarget`
: A simple class that contains the name and number of the receiver of the SMS. They are stored in `Schedule` and requested by `SMSSender`.

`SMSSender`

This class checks for new alarms in the EventDB and for alarms that need redistribution since they lack acknowledgement from the previous receiver. SMSSender creates `SMSOutEvent` that will be added to the database as the messages are sent. It interacts with the schedule to retrieve the proper recipients. SMSSender is a `JThread` from the Lund Java-based Real-Time package.

4.6 Third-party packages

When developing Pheidippides we have incorporated third-party packages to solve specific tasks. Their licenses allow us to use and even modify the packages as long as original copyright notices are included (see Appendix C).

4.6.1 James

A vital part of our solution is the mail server which receives the alarms from the terminals. We have chosen the Java Apache Mail Enterprise Server known as James. It is an open source project completely written in Java by the Apache software foundation. What attracted us was the function of a mail application platform, where an API is available for constructing applications who wishes to access the incoming mails. This is where we have incorporated the `MailReceiver`.

More information on http://james.apache.org/

4.6.2 jSMSEngine

The sending and receiving of SMS is handled by jSMSEngine. It is compatible with a wide range of GSM-modems and mobile phones using the serial port for communication.

For the serial interface the Java Communication API is used, which unfortunately is quite picky with the installation. Normally an external package can be placed anywhere the application may access it, but in this case it had to be placed at a certain spot in the JRE installation. Furthermore the installation differs between platforms.

The strategy jSMSEngine uses for checking incoming messages is based on polling. Every ten seconds jSMSEngine requests a listing of all unread messages from the modem. An alternative approach might have been to listen for the new message indication, which can be sent out from the modem as soon as a new message has arrived. The problem with this approach is that the new message indication may be lost if it is occurring while we are trying to send a message.

More information on http://jsmsengine.sourceforge.net/

4.6.3 eXist

The `XMLDB` stores and retrieves information of old `AlarmHappening` in an eXist database. eXist is an XML database where we can make searches with the help of XQuery.

More information on http://exist.sourceforge.net/

4.6.4 Lund Java-based Real-Time

The classes `Callimachus` and `SMSSender` are threads based on the Lund Java-based Real-Time package. We use `JThread` which is equipped with a mailbox. The mailbox is a communication buffer where other classes may put event objects intended for the owner of the mailbox. The mailboxes are used in the internal communication between `MailReceiver` and `Callimachus` and the events used in the communication are sub classing `PEvent`.

More information on http://www.cs.lth.se/EDA040/common/java.shtml

4.6.5 log4j

During the execution of Pheidippides the activity is logged using the log4j package. While developing the program we label information that we might wish to record on the scale DEBUG - INFO - WARN - ERROR - FATAL. Without recompiling our project we can set the level of logging depending on our intentions. If we are trying to locate a bug we set the logging level to DEBUG to record all information to see what might have caused the problem. During normal execution we might only be interested in that an ERROR has occurred, not wasting space on recording all debug information.

More information on http://logging.apache.org/log4j/

4.6.6 Joda Time

We have replaced the date and time classes in Java with the Joda Time package. Joda Time has a many useful functions which are put to use in the `RotatingSchedule` implementation. It also fixes many annoying issues such as the inconsistent numbering of days and months in the date and time classes in Java.

More information on http://joda-time.sourceforge.net/

4.7 Comments

In retrospect there are a few things we would like to have done differently in the development of Pheidippides.

First off we would like to have separated the third-party packages from our own implementations to a greater extent. The interaction could be handled with a wrapper class for each of the third-party packages we use. This would make it easier to maintain the program if we for example would like to replace a package. If there were changes in the Application Program Interface (API) of the packages, we would only have to change the implementation of the wrapper. As of now we would have to alter classes with other responsibilities which might introduce unexpected errors. The main goal by introducing wrappers would be to make the implementation of our classes general while the wrappers of the packages would be adapted to the specific package API.

All classes in our implementation are not strictly keeping to their respective responsibilities. The `SMSSender` class is involved in checking the database for new alarms to transmit, as well as checking if alarms are to be resent. `SMSSender` could be limited to buffer and transmit outgoing messages while a new separate class is responsible of determining which alarms that are to be delivered. This would make the program easier to maintain and would simplify extending the program with other ways of transmitting alarms.

4.8 History

So why did we name our alarm distribution program Pheidippides?

At the battle of Marathon (490 BC), where the Athenians crushed the Persians, the legend claims that a man called Pheidippides was sent to Athens as a messenger to report of the victory. He completed his task but unfortunately died in the process (his death ought to disqualify the name from being used for a stable alarm distribution program). The Athenian commander at Marathon was Callimachus who plays a vital part in our program (he died as well).

Chapter 5

Results

5.1 Results

We now look back to the analysis we did in Section 1.3 and see how we have fulfilled the requirements we found. For convenience we first repeat the requirements:

1. The operator terminal located in the pump station needs a way to communicate with the central computer. This can be done by modem, through either an internet connection or a connection over Point to Point Protocol (PPP).

2. It should be possible to use different communication alternatives including Local Area Network (LAN), Asymmetric Digital Subscriber Line (ADSL), GPRS, Global System for Mobile communications (GSM) and conventional modem.

3. The terminal must be capable of transmitting the alarms once a connection has been established with the central alarm handling computer.

4. An application capable of receiving the alarms and distributing these properly to the operators should be running on the central alarm handling computer. The application should distribute the alarms to the operator on call by Short Message Service (SMS), according to a rotating schedule, using a GSM modem or an internet service. The operator must respond with an acknowledgement within a specified time or the alarm should be distributed to a wider group.

5. A remote user of the operator terminal should be able to make the same changes as a local user.

6. The communication between the remote operator and the terminal should be handled in a secure way, for example using tunnelling through a Virtual Private Network (VPN).

7. The central alarm handling computer should have a firewall, for increased security.

8. A document with detailed instructions of how to configure the hardware and software involved should be produced.

Now we look at each requirement and how we have dealt with it:

1. This requirement was already fulfilled. The operator terminal supports connection by Ethernet (the most common LAN type) or by using a modem connected to the serial interface for a PPP connection.

2. In phase 1 we wrote documentation for setting up the communication over LAN, conventional modem and GSM modem. It should be straight forward to expand to ADSL and GPRS with the same solution. See Section 3.1

3. This requirement was already fulfilled. The operator terminal supports sending alarms by e-mail, using to the Simple Mail Transfer Protocol (SMTP).

4. In phase 1 we used Nimbus Alarm Server, which more or less fulfils the requirements. In phase 2 we developed our own application - Pheidippides. In Section 1.3 we gave several further requirements we wanted our application to fulfil, out of which we have achieved all. Thus, Pheidippides...

   - Receives alarms by SMTP from Beijer Electronics operator terminal E-1000 family.
   - Distributes alarms to operators by SMS.
   - Uses a rotating schedule to determine which operator should receive the SMS.
   - Handles acknowledgements from the operators.
   - Presents information in a user friendly interface.
   - Logs all occurred events in a persistent way.

5. This requirement was already fulfilled. It is possible to mirror the terminal with full functionality from a remote computer using the application Remote Access Viewer by Beijer Electronics AB.

6. We looked briefly at this in phase 1 but found it to be a too large subject to fit into our Master's Thesis. See Section 3.1.

7. In phase 1 we made detailed step-by-step instructions for configuring the Windows XP SP2 firewall for our solution. See Section 3.3 and Appendix B.

8. Detailed step-by-step documentation for setting up our solution with Nimbus Alarm Server was written in phase 1. This was then expanded to cover Pheidippides in phase 2. See Appendix B.

5.2 Conclusions

We have constructed a solution for receiving alarms from an E-1070 operator terminal and distribute them to operators on call by SMS with the use of a GSM modem. First we used an existing application, Nimbus Alarm Server by TroSoft HB, and then we developed our own, Pheidippides.

Nimbus Alarm Server is an advanced application, the usages of which far outreache the scope of this Master's Thesis. For the uses we have had in mind it works but not very well. It is probably most in larger facilities with integration towards Supervisory Control and Data Acquisition (SCADA) systems that Nimbus Alarm Server shines. For smaller businesses Pheidippides is probably a better alternative, as it provides a user interface that is easy to understand and is also provided for free. For more specialised needs one could also modify to the source code since it is freely provided.

In developing Pheidippides we have used several third party solutions in an object oriented manner, which has given us benefits but also difficulties. A large amount of documentation had to be read in order to understand what most packages actually provided, and comparing different alternatives was a very time consuming process. Often it would probably have been as time efficient to develop a more limited version of a package ourselves. However, a big advantage with using an available solution that is publicly offered and has widespread use is that it is stable and thoroughly tested.

An issue we have had to deal with on several occasions is the balance between feature configuration and usability. Even if we could introduce more features in the program and make these user definable, it would make the user interface cluttered and hard to understand. Often it is preferable to somewhat limit the settings and provide a clean user interface that is easy to understand. The few users that really need to change a behaviour can modify the source code themselves.

We have encountered the problem of falling into a thinking pattern that might be well adjusted to our way of thinking but not a good solution to the given problem. The best example of this is the database structure we developed (see Section 4.4), where we first had a very hierarchical structure but ended up with a simple vector fulfilling our needs.

5.3 Further work

For future development of Pheidippides there are a few areas we suggest concentrating on.

During the later part of our development we let our company supervisor rank some features based on urgency. On that list there are two items which have not been implemented.

The first was a function which lets us remotely check that Pheidippides is up and running. It can be done by the operator sending an SMS to Pheidippides who responds with a status report. Alternatively a status report can be sent from Pheidippides with a static interval.

The second function would let us pause the distribution of SMS. This is useful when the monitored process is in the start up phase and many alarms may be triggered in a controlled environment.

An other area where improvements can be made is regarding the schedule. As of now, one operator is assigned to all incoming alarms. A system where operators were assigned to alarms depending on the alarm source would be desirable. If the server would use different schedules depending on alarm source, it would allow larger or perhaps even multiple plants to use the same server.

Intelligent alarm handling would be a nice feature, especially in larger facilities where a single fault could trigger a domino effect and cause a large amount of alarms to be generated, making it very hard to find the actual problem in all the information. The logical place to put this intelligence would be in the terminal rather than in the alarm handling application.

A deeper look into security issues regarding the communication between the operator terminal and central alarm handling computer would be good.

References

[1] Per Holm, *Objektorienterad programmering och Java*, Studentlitteratur, 2000

[2] Stevens Pooley, *Using UML - Software engineering with objects and components*, Addison Wesley, 2000

[3] Behrouz Forouzan, *Data communications and networking*, McGraw Hill, 2001

[4] Karl-Erik Årzen, *Real-time control systems*, Lund University, 2003

[5] Gustaf Olsson Christian Rosen, *Industrial automation*, Lund University, 2003

For more information on Java please visit java.sun.com

Appendix A

Acronyms

ADSL (Asymmetric Digital Subscriber Line) - High speed internet access using the telephone line.

API (Application Program Interface) - Software interface to system services or software libraries.

FTP (File Transfer Protocol) - Standard protocol for transferring files from one computer to another.

GPRS (General Packet Radio Service) - Packet switching system enabling enhanced data rate over GSM networks

GSM (Global System for Mobile communications) - The most widely used digital mobile phone system and the de facto wireless telephone standard in Europe.

HTTP (Hyper Text Transfer Protocol) - Standard protocol for transmitting Web pages on the internet.

JDK (Java Development Kit) - Software development package from Sun Microsystems that implements the basic set of tools needed to develop Java applications.

JRE (Java Runtime Environment) - Package of software that must be installed on a machine in order to run Java applications.

LAN (Local Area Network) - Computer network covering a local area, like a home, office or small group of buildings.

PLC (Programmable Logic Controller) - Highly reliable special-purpose computer used in industrial monitoring and control applications.

PPP (Point to Point Protocol) - Standard protocol for connecting to internet through a modem.

RS-232 (Recommended Standard 232) - Standard interface for serial binary data interconnection between devices.

SCADA (Supervisory Control and Data Acquisition) - Generic architecture for industrial monitoring and control systems.

SMS (Short Message Service) - Service for sending short text messages to mobile phones.

SMTP (Simple Mail Transfer Protocol) - Standard protocol for transferring e-mail.

SQL (Structured Query Language) - Structured query language for accessing relational database systems.

UML (Unified Modelling Language) - Standard notation for expressing object-oriented analysis and design decisions.

VPN (Virtual Private Network) - Technology that establishes a private or secure network connection within a public network, such as the internet.

XML (Extensible Markup Language) - Markup language for documents containing structured information.

Appendix B

User manuals

We have constructed detailed step-by-step instructions for both the solution we constructed in phase 1, using Nimbus Alarm Server, and phase 2, using Pheidippides. These have been written in RoboHelp and are available only in electronic format. They are distributed with Pheidippides and also available at Beijer Electronics' webpage http://www.beijer.se.

Appendix C

Software licenses

In Pheidippides we make use of several third party libraries which are distributed under either the Apache License Version 2.0 or the GNU Lesser General Public License. Since these licenses take up quite a few pages, and that not many are likely to actually read them, we have decided not to include them in the report. They are available online at http://www.apache.org/licenses/ and http://www.gnu.org/licenses/licenses.html respectively.