Email Meets Issue-Tracking: a Prototype Implementation

A thesis submitted in fulfilment of the

requirements for the degree of

MASTER OF SCIENCE

of

RHODES UNIVERSITY

by

Zukhanye N Kwinana

December 2005

Abstract

The use of electronic mail (email) has evolved from sending simple messages to task delegation and management. Most mail clients, however, have not kept up with the evolution and as a result have limited task management features available. On the other hand, while issue tracking systems offer useful task management functionality, they are not as widespread as emails and also have a few drawbacks.

This thesis reports on the exploration of the integration of the ubiquitous nature of email with the task management features of issue-tracking systems. We explore this using simple ad-hoc as well as semi-automated tasks. With these two working together, tasks can be delegated from email clients without needing to switch between the two environments. It brings some of the benefits of issue tracking systems closer to our email users. The system is developed using Microsoft Visual Studio .NET, with the code written in C#.

The eXtreme Programming (XP) methodology was used during the development of the proof-of-concept prototype that demonstrates the integration of the two environments, as we were faced at first with vague requirements bound to change, as we better understood the problem domain through our development. XP allowed us to skip an extended and comprehensive initial design process and incrementally develop the system, making refinements and extensions as we encountered the need for them. This alleviated the need to make upfront decisions that were based on minimal knowledge of what to expect during development.

This thesis describes the implementation of the prototype and the decisions made with each step taken towards developing an email-based issue tracking system. With the two environments working together, we can now easily track issues from our email clients without needing to switch to another system.

Table of contents

List of figures

List of tables

List of code segments

List of acronyms

API	Application Programming Interface
APOP	Authenticated Post Office Protocol
ASP	Active Server Page
COE	Centre of Excellence
COM	Component Object Model
CSCW	Computer Supported Collaborative Work
DLL	Dynamic Link Library
DNS	Domain Name Server
EAI	Enterprise Application Integration
FSM	Finite State Machine
FTP	File Transfer Protocol
HCI	Human Computer Interaction
HP	Hewlett-Packard
HTML	Hyper Text Markup Language
HTTP	HyperText Transfer Protocol
IBM	International Business Machines
IIS	Internet Information Server
IMAP4	Internet Message Access Protocol 4
LDAP	Lightweight Directory Access Protocol
LOB	Line Of Business
MAPI	Messaging Application Programming Interface
MIME	Multipurpose Internet Mail Extension
MTA	Mail Transfer Agent
MUD	Multi-User Dungeon
NDS	Netware Directory Service
PEA	Personal Email Assistant
PIA	Personal Information Assistant
PIM	Personal Information Management
POA	Post Office Agents
POP3	Post Office Protocol 3

RFC	Request For Comment
S/MIME	Secure Multipurpose Internet Mail Extension
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
TDD	Test-Driven Development
WfMC	Workflow Management Coalition
WFMS	Workflow Management System
XML	Extensible Markup Language
XP	eXtreme Programming

Acknowledgements

A special thank you to the One who has guided and blessed me, especially when I felt as if I had done all that I could.
A heart-felt thank you to my family for being there through the good and the not so good, joy shared is joy doubled, sorrow shared is sorrow halved.
Thanks to my "extended family", which had to put up with a lot of my drama, why you guys are still around is beyond me ;)
And last but certainly not least to my favourite niece in the whole wide world, Kwakhanya you are a star!

Thank you to my supervisors for their guidance throughout the past two years.

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, with financial support from Telkom SA Ltd.

Chapter 1

Introduction

1. Introduction

Individuals both at home and at work are needing to deal with increasing amounts of information, which they need to develop ways of managing so that they can find and use later. The act of storing and organizing information so that it may later be retrieved from the personal information store is known as personal information management, or PIM [Spurgin 2003]. This information can be stored in different places, depending on the person, but a lot of the information is stored in electronic mail clients. This is logical because electronic mail (email) has become one of the most widespread methods of communication in today's society, and one of the most popular applications on the Internet [Lyman and Varian 2000] in [Takkinen 2002]. Email is ubiquitous and plays a central role in the workday and it is being used for spontaneous conversation, as well as business communication. A large number of people keep an email client open in the background, visiting it as regularly as when new email arrives. The success of email has encouraged users to employ computer networks for accomplishing and delegating tasks in their daily life at work and so email is used for much more than receiving and sending text from one person to another; its use has evolved to support contact, document and task management [Spurgin 2003].

Managing tasks to completion within an organization is crucial in order for a company to function in a coordinated manner. More often than not tasks are delegated using email. Due to human error, these tasks can be forgotten or the deadline mistaken. While email is a useful tool, it was not originally designed to support the management of tasks, and does not provide facilities for monitoring task status, nor does it allow adequate access to task-related information.

Researchers have investigated and developed systems that aid in task management. These systems fall under the broad category of Computer Supported Collaborative Work (CSCW). CSCW is a designation that focuses on work from a broad perspective – the tasks that people carry out, their workplaces, and technology that supports that work [Neurauter 2002]. Groupware and workflow (management) systems (WFMS) fall under CSCW. Groupware is software that functions to provide a means for human collaboration [Lococo and Yen 1998]. WFMS is a networked control system that assists in analysing, coordinating and executing business processes by automatically defining, creating, and managing the execution of

workflow models. This is done by using one or more coordinated engines, which are responsible for interpreting process definitions, interacting with agents, and when required, invoking the use of the other information systems involved in the work [Ellis and Wainer 1999].

There have been a few systems that have been developed that have tried to aid task management, for example [Liberum 2000, Rhett 2004]. Some of these have done so by editing the email client's user interface to help make the task-centric attributes more visible and less likely to be forgotten [Bellotti, Ducheneaut, Howard and Smith 2003, Baecker, Grudin, Buxton and Greenberg 1995]. [Bergman, Griss and Staelin 2002] uses a method of filtering the emails into different folders. Each of these systems has, in their own way, aided task management in email. These systems have a few dilemma-causing drawbacks:

- Individuals are isolated from the rest of the organization if they are not using the system.
- Removing the user from a well-known environment (their email inboxes) and forcing them into an unknown interface (a specialized tracking task management system) that they need to learn encounters resistance. Bellotti, Ducheneaut et al. [2000] noted that people are not keen to leave their "comfortable" email inboxes and move to new software. Short of starting from top management down, people would much rather use the email clients they are familiar with.
- Workflow and task tracking systems usually have a limited scope: using them is generally limited to within one's organization or department. This is a problem because a large number of tasks are delegated to people outside an organization, or outside the department.
- Email is just used to send emails by these systems but they do not play a central role – for example delegating the task – in the steps that need to occur in the tracking system.

The above-mentioned points show some of the drawbacks of current systems that manage tasks. We will use these points to direct us towards building a system that caters for and removes these drawbacks. Our research is based on the observation that the features in an issue-tracking system could greatly assist task management if the main steps of task initiation and delegation could be easily accomplished from our email agent. As stated in [Kwinana,

Wentworth and Terzoli 2004], we want to integrate the stateless emails with the stateful tasks in an issue tracking system to come up with a system that takes the best of both and puts them together. Email is, in some sense, a lowest common denominator service used by everyone, within and across organizational boundaries. The aim of this project is to enhance the coupling between our email systems and an issue tracking system, so that tracked tasks can be initiated, delegated, and monitored via email, by users both inside and outside of an organization.

This system will have all the functionality of an issue tracker without its drawbacks. It will be email based so the user will be kept inside their inbox, their *habitat* [Ducheneaut and Bellotti 2001], which they are familiar with. From their habitat they will delegate a task and the issue-tracker will take over in tracking the task and assist in it being completed. The issue tracker will send reminder emails as well as escalating emails when necessary. Due to the fact that it is email-based, it will be able to track tasks delegated inside and outside an organization; also no one will be isolated by the organization if they choose not to use it. Our system will not have a list of users stored somewhere and so anyone can make use of it and not just registered users. This system goes a long way in information management in terms of tasks so as to ensure that the work that needs to be done is completed on time.

1.1. Approach to the project

The first step in this project is to review the existing literature. We will begin with what has been written on CSCW, WFMS and groupware. This will help in understanding the discipline we will be working under and any requisites to which we need to adhere. This will also help us to correctly classify our system under the correct "label", with correct components. After this we will extend the review of the literature to look into systems that have been developed, under CSCW, that assist in managing tasks. These systems will range from academic systems all the way to commercial systems that attempt to manage tasks in whatever degree.

To implement this system, we employ the eXtreme Programming (XP) methodology. "XP is a light-weight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements" [Beck 2000]. According to XP, we approach the project using *user stories* (desirable system features specified by the customer as stories). As

XP works an *iteration* (valuable stories selected by the customer for the programmers to work on for the next few weeks) at a time, user stories will be selected that best suit the first iteration, after which the programmers will start discussing how they think the story should be implemented. These stories do not just increase the features of the system; they help in pushing the design and implementation forward. It is with their help that we refine the system specification incrementally one story at a time. As time goes by and more information is gathered the system will fine-tune itself, one iteration at a time, answering questions as it goes along that improve the design of the system.

1.2. Structure of the thesis

The rest of this project write-up is structured as follows:

Chapter 2 introduces and categorises task management in terms of CSCW, groupware, workflow and WFMS. This provides a common understanding of the terminology, and allows us to accurately determine where our system belongs.

Chapter 3 examines specific task management research systems and products that have been developed. This ranges from systems developed at educational institutions all the way to commercial products that support task management. This chapter also has an introduction to XP.

Chapter 4 is the first implementation chapter where we concentrate on our first iteration. The main user story is based on a simple task that is delegated via email, with no preset steps. The system intercepts the email before it is delivered, and if it is marked for tracking, details of the email are recorded. The main user story helps us make a number of decisions, which are covered in this chapter, for example, how to approach client side and mail side interception and tracking and how to use the Outlook object model.

Chapter 5 concentrates on the second iteration where we start refining the basic story, incrementally adding more workflow features. The main user story is based on commonly occurring tasks within our organization, whose steps are known and thus can be automated, such as reserving accommodation or a car. This user story was subsequently expanded to

include choice, parallelism and looping, typical control structures needed to express more complex workflows.

Chapter 6 is the final chapter where we discuss the resulting system and how well it performs the tasks it was developed for. Decisions made and their impacts are discussed as well as any possible future extensions.

Chapter 2

Literature review

2. Introduction

Some terms are used extensively in the literature. We present a discussion and explanation for these important concepts. This serves to clarify the terms and meanings we adhere to in this write-up.

The term Computer Supported Collaborative Work (CSCW) originates from a workshop in 1984 that was organised by Paul Cashman and Irene Grief [Grudin 1994]. The reason for the workshop was to bring together people from different disciplines that shared an interest in how people work so that they might understand how technology could support them. The idea of people working together using technology even if they were spatially separate started off as a game. The Multi-User Dungeon (MUD) is a complex variant of a common style of computer game known as the "adventure game" [Dourish 1998]. Richard Bartle and Roy Trubshaw at the University of Essex developed MUD in 1979. The players of the game participated in a fictional world moving and acting in a textual virtual environment recorded in a software database. This game was quite similar to the many existing games except for one thing – interaction with other players. Many people could play the game all at once over a network. They could occupy the same dungeons at the same time. The players interacted with each other as well as the world created by the programmer.

Even though CSCW was coined in 1984, it was Curtis's 1992 paper that examined patterns of social interaction in LambdaMOO, which looked at these systems as valid areas of mainstream research [Dourish 1998]. The paper served to "crystallise and legitimatise an area of research which had, perhaps, been somewhat clandestine before, at least within the HCI and CSCW communities... The subsequent work conducted by Curtis and colleagues at Xerox PARC, and by Bruckman at the MIT Media Lab spread the idea of collaborative virtual realities as places for collaboration" [Dourish 1998]. The collaboration of people in different places over a common goal then grew to an area of research that contains labels like groupware, workflow management systems (WFMS), and the like.

The sections to follow are a more detailed discussion of the different aspects of CSCW, starting with CSCW itself, after which groupware, workflow and workflow management systems (WFMS) are discussed. The purpose of the discussion is to define each term and its

rationale. This is so that we understand the previous research that has been undertaken and the reasons behind it so as to better understand the context of our system. We then introduce the research that has been undertaken to facilitate task management in email.

2.1. Computer supported collaborative work

A broad definition of CSCW is provided by Neurauter [2002]: Computer-supported cooperative work (CSCW) is a designation that focuses on work from a broad perspective – the tasks that people carry out, their workplaces, and technology that supports that work. CSCW thus reaches from sociological analysis and anthropological descriptions of work to the technological foundations of systems [Neurauter 2002]. In the acronym CSCW, the second C can be replaced by either collaborative or cooperative. In our discussions we use collaborative, but we do not edit other writer's definitions when we refer to them. A few definitions of CSCW blur the difference between groupware and CSCW and assume that they are almost identical if not synonymous. An example is the definition that "Computer-supported cooperative work or CSCW is computer-assisted coordinated activity carried out by groups of collaborating individuals" [Hazemi, Hailes, Wilbur, Pitsika and Wilbur 1996]. The most descriptive definition and, I feel, the most appropriate for our needs is by Terzis and Nixon [1999]: "CSCW is the scientific discipline that motivates and validates groupware design. It is the study and theory of how people work together, and how computer and related technologies affect group behaviour. CSCW is an umbrella collecting researchers from a variety of specialisations – computer science, cognitive science, psychology, sociology, anthropology, ethnography, management, information systems – each contributing a different perspective and methodology for acquiring knowledge of groups and for suggesting how the group's work could be supported." In general, CSCW is a term mainly used by the research community while groupware is a term used by the business sector to refer to the numerous products available for workgroup support [Takkinen 2002].

We now narrow down CSCW and introduce Figure 2.1 below from Grudin and Poltrock [1997], which represents a focus of computer systems development and the principal "customer" or "user" of the resulting technology. Until recently, all activity was in the shaded outer and inner rings. The outer shaded ring represents major systems and applications, that is large mini computer systems and mainframes designed to serve organizational goals. The

organizational goals were transaction processing, order and inventory control, computer integrated manufacturing and so on. The inner shaded ring represents applications designed primarily for individual users of PCs and workstations, that is, word processors, debuggers, spreadsheets, games and so forth. The two rings between these represent large projects and small groups. Large project support included electronic meeting rooms and workflow automation systems, which are most useful for groups of 6 or more. Development in each of the middle rings is groupware. On the horizontal axis of the left half of Figure 2.1 are software development contexts that dominate development of systems and applications of different scope. The two unshaded central rings represent groupware development:

- Government contracts have stimulated project-level software support.
- Small group support has been a new focus for commercial product developers and telecommunications companies who are strongly interested in technologies such as video.

The emergence of CSCW in the 1980s included both of these but is more strongly tied to the second, the shift of attention to small-networked groups. On the right of the axis are major research areas associated with the development and use of systems linked to each development context and a date by which each was firmly established.

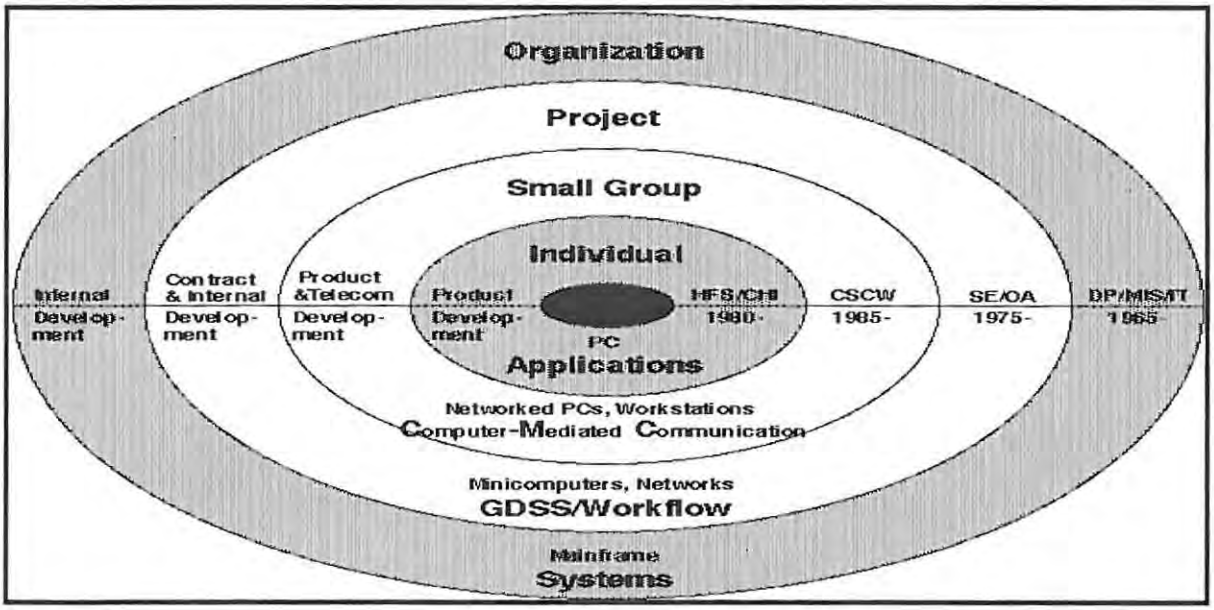


Figure 2.1: Computer system development focus [Grudin and Poltrock 1997]

As per the definition above, CSCW is an umbrella that covers groupware. Groupware is of importance to this research and thus the section below discusses it in more detail.

2.1.1. Groupware

There are three primary ways in which people work together in groups:

- They communicate with each other by sending information, requests or instructions.
- They collaborate with each other by working together on joint projects.
- They coordinate with each other as participants in structured or semi-structured sequence of tasks, or business processes.

Communication, collaboration and coordination are the "3 Cs" of groupware [Quixa Solutions 2005].

Roughly speaking, groupware is computer software that functions to provide a means for human collaboration [Lococo and Yen 1998] but more specifically "Groupware is a generic name for computer systems and applications that support collaborative work. There is a large variety of systems that can be categorized as groupware, including video conferencing systems, collaborative authoring systems, and group decision support systems, to name a few that have little in common. In contrast with workflow, which aims to make work more efficient by implementing work processes, groupware does not prescribe how the work is to be done" [Podgorny, Walczak, Warner and Fox 1999]. A large number of groupware systems are "applications for allowing a workgroup to share information, with updates by each user being made available to others, either by automated replication or concurrent sharing" [Olap Report 2005]. Groupware can be seen as network software that enables two or more users to work collaboratively in order to meet a particular goal [Ministry of Finance 2000]. There are many different definitions for groupware, but as can be seen from the above-mentioned definitions, all of them agree that groupware is productivity-ware that enables human interaction and allows for collaboration across time and place [Lococo and Yen 1998]. Another trait that appears in definitions is the obvious fact that groupware is software based and provides a shared environment where users work together. Below is a discussion about groupware products that fall under the 3 Cs of groupware as referred to in [Quixa Solutions 2005].

Communication products

These software products enable users to quickly and easily communicate with each other. Communications in workgroups have the following general attributes that are addressed by groupware products in this category:

- Communication is mostly ad hoc or random; there is no structure or process.
- Communication products must be quick and easy to use.
- Communication products must be low cost.
- They must be as widely deployed as possible to be useful.

Examples of communication products include email, fax, computer telephony, and video conferencing and chat programs.

Collaboration products

This is "software that enables a group of users to collaborate on a project by means of network communications" [Liebert Coporation 2005]. Collaboration in workgroups involves "knowledge workers" who work together as teams on projects such as producing a report, creating marketing collateral, designing a complex product, or participating in research. Therefore collaboration solutions enable individuals to work together on joint projects by providing the following:

- A "document" or repository where the collective work of the team is stored and easily accessible to all participants. The "document" is the key since it is the repository of the work, and the form in which the work is saved and displayed.
- A means for "knowledge workers" to access the document with good control over who has rights to do what.
- An interface that is easy to use and non-intrusive, so that it does not impede creativity that is essential for the success of knowledge workers.

Examples of collaborative solutions include Lotus Notes, document management systems and other multi-user applications.

Coordination products

In addition to communicating and collaborating, individuals also work together by participating in structured or semi-structured processes. This is workflow. Groupware solutions designed for coordination (workflow) cater to certain needs:

- The "process" is the essence of workflow, or coordination, therefore the solution should enable an organization to effectively implement its business processes.
- Processes are structured or semi-structured, they are never purely ad-hoc.
- Coordination is "pro-active." Its purpose is to push towards reaching a goal or outcome.

Every organization has a large number of business processes, therefore workflow is prevalent in every organization in some way, shape, or form. Examples of workflow include purchase orders, claims processing, reviews, expense reports, change orders, order processing and numerous others.

The 3 Cs of groupware have provided a way to understand groupware better and with the understanding it becomes easier to classify groupware [Dix, Finlay, Abowd and Beale 1998]:

- By where and when the individual participants perform the cooperative work – summarised in a time/space matrix (as in Table 2.1)
- By the function of the system – e.g., collaborative design, group authoring, meeting support, etc
- By the structural support function of the software
    - Computer-mediated communication – where direct communication between participants is supported
    - Meeting and decision support systems – where common understandings are captured
    - Shared applications and artefacts – where the participants' interaction with shared work objects (the artefacts of work) are supported

Figure 2.2 illustrates the resulting triangle when arranging the types of groupware-applications according to their degree of supporting functionality. This leads to different system classes [Neurauter 2002]:

- Communication: Supports space and time independent communication of different partners by use of electronic mail systems, video conference systems and others.
- Shared Information spaces: Store information for defined or undefined periods of time for implicit access e.g. hypertext-systems, databases or bulletin-board-systems (BBS)
- Workflow Management: Software-based support for the tasks that have to be carried out within a workflow (a workflow defines a sequence of activities).
- Workgroup Computing: Supports intra-personal coordination within groups or teams that were established to solve simple or infrequent tasks.

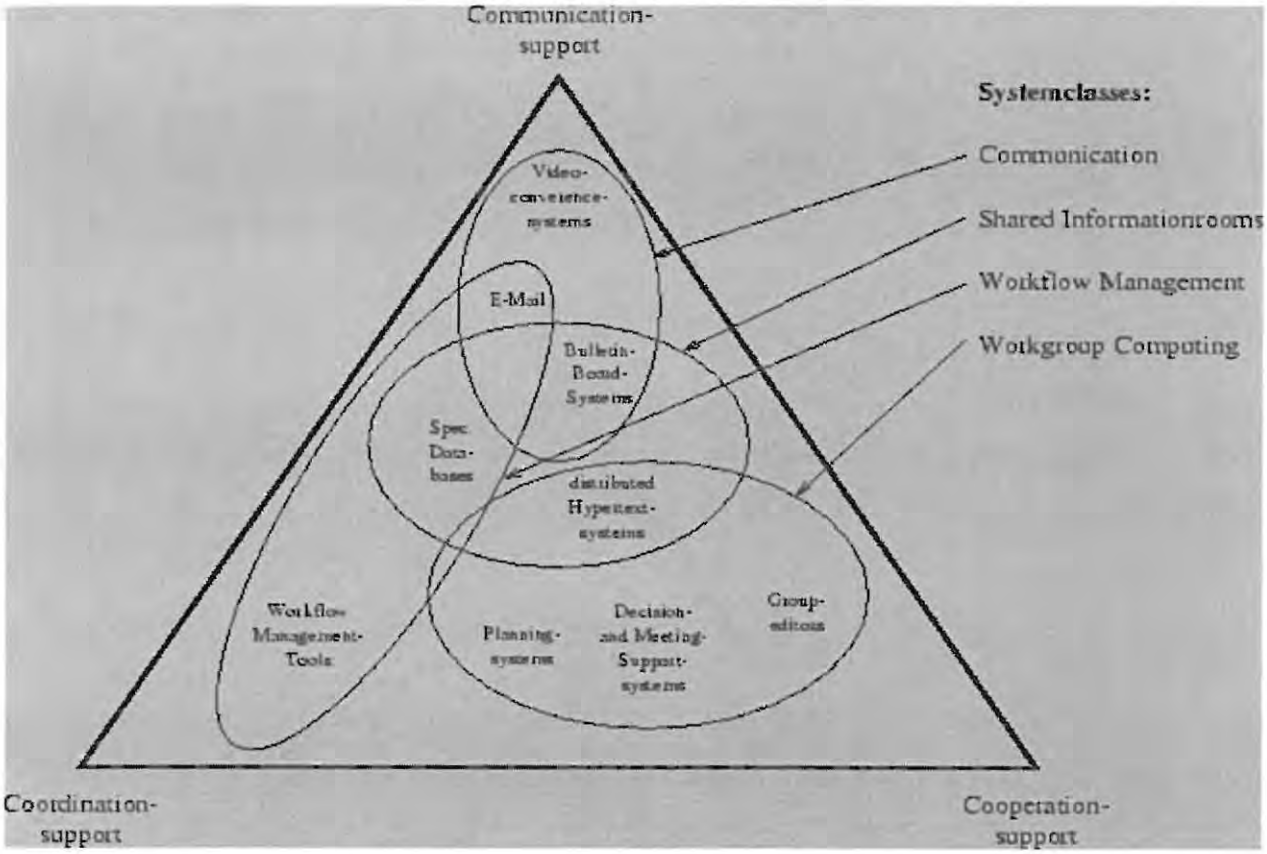


Figure 2.2: Classification corresponding to function of support [Neurauter 2002]

Groupware is as much technology as it is a human process. The two are merged to create an environment supporting collaborative work. There are three basic components of groupware [Podgorny, Walczak et al. 1999]:

- Knowledge base – technically, a data repository of any kind
- Workflow – a set of rules describing the activity in which a group of people participates
- Collaboration – a process of exchanging messages between group members

The three components are intertwined in a groupware package. Workflow definition can be stored in the knowledge base, and messages created by the collaboration process may become part of the knowledge base. Workflow outlines the scope of the collaboration process, which can also be affected by the contents of the knowledge base. Traditionally, groupware supports asynchronous collaboration: the messages are being forwarded and stored in the knowledge base [Podgorny, Walczak et al. 1999].

It was Podgorny, Walczak et al.'s [1999] opinion that "the internet will breed a completely new way for people to interact via their desktop machines. This technology will soon affect all aspects of life of the techno-societies, including a profound change in business practices."

Based on whether users are working asynchronously or synchronously, at the same place or different places, the domain of groupware or CSCW can be separated into four quadrants as in Table 2.1, the time/space matrix below [Takkinen 2002]. Our focus in the thesis is groupware that does not require members to work together at the same time. "Asynchronous groupware supports communication and problem solving among groups of individuals who contribute at different times, and typically also are geographically dispersed" [Baecker, Grudin et al. 1995].

	Same place	**Different places**
Same time ***Synchronous***	Electronic meetings Team rooms Group decision support systems Electronic whiteboards	Video conferencing Teleconferencing Screen saving Document sharing Electronic whiteboards
Different times ***Asynchronous***	Shared container Mailboxes Electronic bulletin boards Virtual rooms, kiosks Document management systems	Electronic mail Workflow Form flow Messaging Routing and notification

Table 2.1: Time/space matrix [Takkinen 2002]

"Groupware is distinguished from normal software by the basic assumption it makes: groupware makes the user aware that he is part of a group, while most other software seeks to hide and protect users from each other ... Groupware ... is software that accentuates the multiple user environment, coordinating and orchestrating things so that users can "see" each other, yet do not conflict with each other" [Hazemi, Hailes et al. 1996].

Below are some expectations and advantages of groupware [Linktionary 2001]:

- Groupware stimulates cooperation within an organization and helps people communicate and collaborate on joint projects.
- Groupware coordinates people and processes.
- Groupware helps define the flow of documents and then defines the work that must be done to complete a project.
- Groupware provides a unique way for users to share information by building it into structured, compound documents. The document becomes the central place where shared information is stored.

Ideally, groupware should be able to help each person in a collaborative project perform his or her specific job in a more efficient way. Groupware simply defines ways of using existing applications to share information and help users collaborate.

2.1.1.1. Electronic mail

The first category of groupware applications are Electronic Mail Systems, which is relatively mature after many years of use and widespread acceptance. Many believe that these systems are the only really successful groupware application [Terzis and Nixon 1999]. This is not too great a surprise as email takes after the "writing a letter" metaphor. People are immediately attracted to email because how to use it comes naturally and is thus comfortable. It comes also as no wonder that a lot of people "live" in their inboxes. Through the years Electronic Mail Systems evolved towards many different directions employing all the developments in computer technology. So, the contents of the message evolved from systems for the exchange of simple text messages, to systems for the exchange of compound documents (documents that include images, graphs, etc), to even multimedia documents (e.g. voice mail systems). In parallel they evolved from systems that maintained a flat collection of messages to hypertext

systems (e.g. Hypermail) or even complex systems for message handling like CLUES that allows dynamic personalised message filtering, prioritising voice and text messages using personal information from the user's workspace [Terzis and Nixon 1999].

As users continue to use it, their expectation of email changes and so email evolves to meet the capabilities of computers and users' changing expectations. Improvements in email include intelligent agents that use the structure of the message, standard message representations, a greater range of content, and more reliable, scalable architectures. Email is fundamentally structured. Messages consist of a series of field labels (To, From, Subject, etc.) and field values, ending with a body field containing the content of the message. An important step in the evolution of email was to provide a capability for creating additional fields. The Information Lens [Malone, Grant, Lai, Rao and Rosenblitt 1989] demonstrated how these fields, combined with agent technology, could help users process and handle their mail. Today many groupware products, including most email systems, contain tools for constructing such agents, and improved human-computer interfaces that make them more usable [Grudin and Poltrock 1997]. Because of its maturity, other categories of groupware rely on electronic mail to deliver messages. Message-based groupware includes applications like Lotus Notes, Novel GroupWise, and Microsoft Exchange [Terzis and Nixon 1999].

Even though the main purpose of email is communication among people, its structure, reliability, and ubiquitous nature has encouraged its use as a means of delivering messages between processes and people or among processes. In this way, email supports coordination as well as communication. For example, many Lotus Notes applications, workflow management products, and calendar systems use email to alert a person of events or of tasks to be performed. Some workflow management systems use email as the mechanism for routing and presenting work to users.

2.1.2. Workflow and workflow management systems

Workflow combines electronic messaging with document management and imaging. The messaging system is used as a transport for document flow sequentially through different processes. Accounting and procurement systems can use workflow management, for example. With the above information we can define workflows as composite tasks that comprise coordinated human and computational subtasks [Lei and Singh 1997]. A document moves

through various stages of processing by being sent to appropriate people who work on the documents, authorize the documents, and validate them. According to the Workflow Management Coalition "workflow" is defined as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to set of procedural rules" [Caro, Guevara, Aguayo and Gálvez 2000] in order to arrive at a common objective. Workflow enables software to emulate the functional processes of a department or an entire organization [Linktionary 2001]. A workflow is a combination of transitions and states that make up a process. Each workflow consists of configurable transitions and states that must be followed from the time that an issue or feature is opened to the time that it is closed [Christlinks 1994]. [Document Management Avenue], a glossary of terms, added that workflow uses the metaphor of a production line to model, manage and monitor clerical, administrative, and document-based tasks. On the other hand Phoenix [2005] pointed out that the participant in a workflow can be either a human resource or a computer application.

Workflow is made up of different concepts whose relationships are discussed below and are supported by Figure 2.3 below [Lei and Singh 1997].

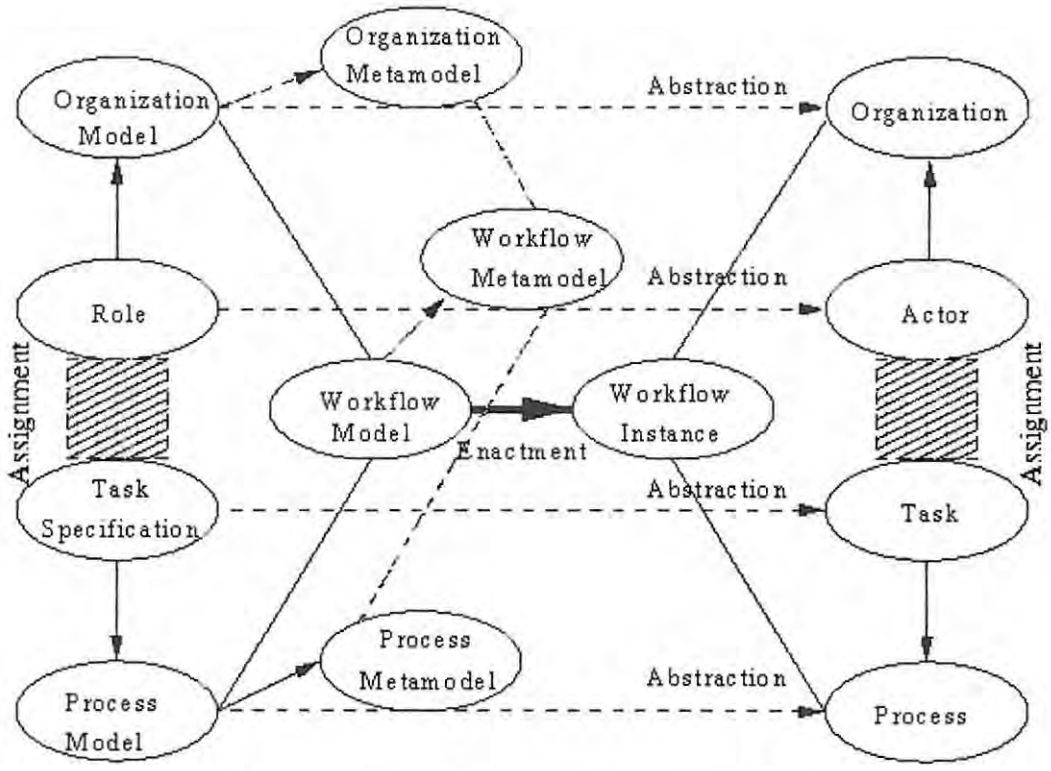


Figure 2.3: Workflow concepts [Lei and Singh 1997]

- A task is a definite piece of work.
- An actor is a human being or a machine that can perform a task.
- A role is a logical abstraction of one or more physical actors, usually in terms of functionality.
- A process is a business process, which is composed of tasks structured in an appropriate manner.
- A process model is an abstraction of business processes. It emphasizes the coordination of tasks by highlighting their interdependence.
- An organization aggregates actors into groups, which are structured in some way.
- An organizational model is an abstraction of organizations.
- A workflow (instance) is a process combined with an organization, and assigns the tasks in the process to actors in the organization.
- A workflow model combines a process model and an organizational model.

The metamodels provide the mechanisms for defining the models of a specific organization, its workflows and its processes.

Workflow has three main areas that are discussed below together with a diagram (Figure 2.4) [Zhao, Kumar and Stohr 2000]:

- Process/Case Management: In process management, business rules incorporate processes, roles and work allocation. These rules are defined and maintained by the business users. Case management exists when a work-item relates to a particular customer. A single individual completes the entire process including all its steps.
- Work Automation: Work automation implies the entire business process is automated through workflow tool sets. With fully automated business processes there is no human intervention, for example Internet banking.
- Collaborative Commerce: Collaborative commerce involves automated collaboration with external organizations. Processes communicate with each other, often using the Internet or the web as the infrastructure, for example Supply Chain Management systems.

Figure 2.4: Three main areas of workflow [Zhao, Kumar et al. 2000]

The workflow technology is used for [Caro, Guevara et al. 2000]:

- Routing work processes within a single organization and between organizations (e.g., travel agency–airline company).
- Reducing costs by means of automation.
- Improving work processes and obtaining faster services and more quality by the use of reengineering.
- Monitoring and managing work processes.
- Accelerating product development cycles.

Employing workflow affords an organisation many benefits. Automation of many business processed results in the elimination of many unnecessary steps and improved efficiency. Improved management of business processes achieved through standardizing working methods and the availability of audit trails leads to better process control. Consistency in the processes leads to greater predictability in levels of response to customers, which improves customer service. Software control over processes enables their re-design in line with changing business needs, providing flexibility. Business processes improve because there is now a greater focus on them, which leads to their streamlining and simplification [e-workflow 2005]

With the presence of workflow, we need a system that is going to manage the business processes that the workflow is involved in. A workflow management system (WFMS) is a networked control system that assists in analysing, coordinating and executing business process by automatically defining, creating, and managing the execution of workflow models by using one or more coordinated engines. These engines are responsible for interpreting process definitions, interacting with agents, and when required, invoking the use of the other information systems involved in the work [Ellis and Wainer 1999]. A WFMS typically has two sub-systems:

- A modelling subsystem that allows organizational administrators and analysts to construct a procedural model of the flow of work among people and tasks.
- An enactment subsystem, which uses the model to coordinate task executions by various participants at various workstations, connected to a network.

Workflow systems are mainly constructed for large workgroups.

A WFMS is made of a large set of software modules, seen in the Figure 2.5 below of the WfMC workflow reference model, which can be classified in the following way [Caro, Guevara et al. 2000]:

- Tools for workflow modelling: These are used for explicit and formal modelling of tasks to be done, the organizational structure of the company, agents, and so on.
- Tools for workflow construction: These are used to build workflow applications, and they are based on the models obtained with the definition tools. Among these software tools, we find a process definition tool, an organizational structure design and agents' characteristic tool, and reengineering and simulation workflow modules.
- Workflow engine: This is the main component for executing a workflow application. This tool is responsible for orchestrating, executing, monitoring, and managing every single task, agent, information system, instance of work processes, and so forth, that comes into play in the WFMS. We could say that it is the true heart of the system.
- Client applications: These are computer applications used by agents of the WFMS to carry out their task.
- Inherited applications: These are applications already existing in the infrastructure of the company that should be integrated in the WFMS.

The implementation of a WFMS provides the following advantages [Caro, Guevara et al. 2000]:

- Formal modelling of processes
- Proper framework for business process re-engineering
- Better planning and possibility of simulation
- Adaptability
- Increase in satisfaction
- Better control of processes
- Restoration of the system after failures and/or exceptions

Commercial workflow systems include systems like Lotus Notes and IBM FlowMark. Research systems include systems like DartFlow, METEOR, WebFlow, and WWWorkflow [Terzis and Nixon 1999].

Workflow systems have emerged as one solution to the problem of coordinating events, artefacts, and people. The approach adopted by workflow systems tries to reduce the complexity of coordination in three basic steps. First, the work to be done is reduced to a basic form through a process of categorization. The categorization breaks up work into elements, for example activities, documents, and user roles. Once categorized, relations between the different components of work can be defined. For example, temporal sequences of events can be put together, or object dependencies can be generated to ensure that entities such as documents may not continue to another state before they have been completed. These sequences or dependencies can be precisely described by the use of a formalism that defines the relations among specified components. Finally, workflow systems may use the formalism to automate some aspects of the work entirely. It is these three steps that give technologies their workflow functionality [Grinter 2000].

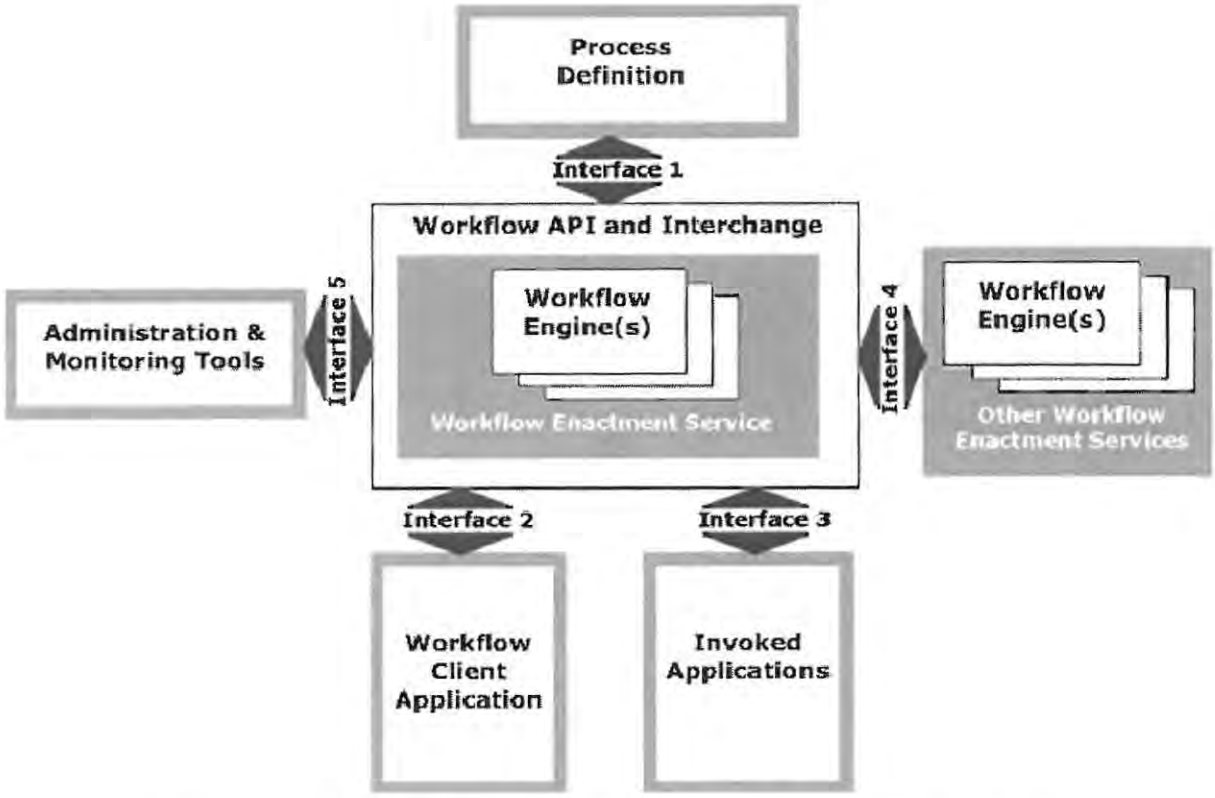


Figure 2.5: Workflow reference model by WfMC [Caro, Guevara et al. 2000]

The above section is necessary in order to explain the beginning of CSCW, its growth to include groupware, workflow and WFMS. The next section moves forward by introducing the research that has gone into managing pending tasks using email. We start by addressing the state of affairs of task management using emails. This ranges from keeping emails in the inbox as reminders or as a to-do list to using external tools, for example issue-trackers.

2.2. Uses of email

Email has become more like a habitat than an application. It is used for a wide range of tasks such as information management and for coordination and collaboration in organizations. Research has been done that shows that email is the place in which a great deal of work is received and delegated and is a growing portal for access to online publications and information services. Email is used throughout the day by many people and is a major means of non face-to-face communication. It is now the main means of document exchange. It is co-opted by its users for much information management functions, such as to-dos (by marking-up or re-sending oneself reminder messages) and contact management (by sorting by name

and filtering). As a result the primary core purpose of email becomes overloaded, providing inadequate support for certain tasks it is routinely used to accomplish [Ducheneaut and Bellotti 2001].

"Management of messages referring to the future is inadequately supported by email tools. Handling this type of messages involves human prospective memory, that is, it involves remembering to remember to perform an action in the future, be it a delayed response to a message, a project task, or attending a future meeting. These are called prospective messages and observe that although previous studies of email use pointed out some prospective uses of email, they did not focus on them. In addition, previous experimental email systems typically dealt either with handling of incoming email traffic or with message organization and retrieval. They dealt, thus, with current and archived messages, but not with prospective messages" [Gwizdka 2001]. Our research is complementary to those efforts, in that we want to support future actions in email.

To show that email is commonly used to handle prospective information, research carried out by [Gwizdka 2001] showed that 15 out of 19 participants kept prospective messages in their inbox. The research also showed that the email environment was also commonly used as a reminder, not only for handling email messages, but also for other actions and about future events (again in 15 out of 19 cases). This is quite similar to a person's work area. The arrangement and position of documents on the work area has significance and the documents are there to act as reminders of tasks that need to be attended to.

A finding from Gwizdka's [2001] study indicates that email messages are used to carry and hold prospective information, but that there is insufficient support for handling this type of information in email. Details of future actions and the context in which they are to be executed are not recorded. Users need to engage in the process of monitoring what tasks are to be performed and when [Gwizdka 2001]. A system is necessary that will remind the individual about a particular task at reasonable intervals while also providing updates to the task when they become available.

Email users are heavily invested in their existing tool. They are thus unlikely to adopt an entirely new tool that requires them to move their legacy email archives and learn new software without some top-down organizational imperative (such as organizational edicts or

moving to a new company with different supported software) [Bellotti, Ducheneaut, Howard, Neuwirth and Smith 2000]. It is thus very important to leave employees in their comfortable environment where they are already efficient – their habitat – and to increase the functionality of their existing email client so that extra functionality is available to them to make their lives easier. If users were to adopt a second or third tool, there is a possibility that their work and attention becomes divided between the tools. More than one tool makes it almost difficult to get a single view of all the users tasks, calendar appointments, to-dos and email. That is why systems like Microsoft Outlook have the feature "Outlook Today", which tries to provide a composite overview of pending tasks, unread emails and the like.

One of the most commonly performed activities in email is management of pending tasks. This research focuses on how to support this activity in email. But first we need to explore other researchers' attempts at task management using emails so as to see where they fall short and thus where we can improve matters. This is done in the next chapter.

2.3. Summary

In this chapter we have introduced, defined and discussed CSCW, groupware, WFMS and workflow – all of which are related to each other and, more importantly, related to our research. The discussion above has shown that technology has played a supporting role when a group of people are working towards a common goal whether it is by aiding in communication, collaboration or coordination. We also discussed how email is widespread and used every day by a large number of people within an organization, and that one of its main functions recently is to delegate tasks within and outside an organization. Email has been overloaded in terms of its functions and so the functionality proposed by our system seems to promise what is needed.

Chapter 3

Related work

3. Introduction

In this chapter we introduce both commercial and non-commercial systems that have been developed and are efficient in solving a particular problem in task management. This is to show that researchers (both academic and in business) recognise the need for a facility that allows management and tracking of tasks in emails. The next section discusses an issue-tracking system developed within our institution that plays a crucial supporting role as the issue tracker used during this project. After this we introduce the methodology we will be employing in order to develop our email-based issue tracking system. XP and test-driven development (TDD) are introduced briefly.

3.1. Groupware

3.1.1. Taskmaster

"An increasing body of literature points to the importance of email as a task management resource. Mackay detailed how it supports a variety of time and task management activities. Whittaker and Sidner extended her findings to show how the email inbox is a repository of "to-dos", "to-reads" items of "indeterminate status" and "ongoing correspondence" that can be difficult to deal with. More recently, we discussed how email is transforming into a habitat, the central place from which work is received, managed, and delegated in organizations" [Bellotti, Ducheneaut et al. 2003]. With this knowledge, researchers from the Palo Alto Research Centre have come up with the Taskmaster.

The Taskmaster system "recasts email as task management and embeds task-centric resources directly in the client" [Bellotti, Ducheneaut et al. 2003]. Taskmaster is an email system entirely redesigned for task and project management. Taskmaster offers a new solution to the often-decried "pain of email" by recognizing upfront that this technology is not simply concerned with messaging, but that dealing with email and managing tasks and projects are indistinguishable. They accomplish this goal purely through a redesign of email's user experience without changing its fundamental technical infrastructure. The researchers of Taskmaster decided that the main element of interest to them was the task not the message. Individual messages can represent tasks, but interdependent tasks comprise threads of

message files, links and drafts. Any related incoming messages (replies in a thread, with any attendant files or links) are grouped together based upon analysing message data. Taskmaster reflects findings that when managing a task, one's own messages are often as important to keep track of as those of others (often representing to-dos for others). So, in a break from the standard "email-as-messaging-system" model (with inboxes and outboxes), incoming and outgoing messages are viewed together.

Taskmaster is implemented in Visual Basic as an add-on to an Outlook client configured for the Exchange Mail Server. It duplicates incoming messages and passes outgoing messages back to Outlook; so all traffic appears in the user's Outlook client in the in- and out-boxes. In the conclusion of the research, the researchers commented "Our research shows that it is possible to significantly and positively affect email users' experience by embedding task management resources directly in the inbox, where they are most needed, as well as breaking down the barriers between the various components of contemporary email applications. The small set of features we have built into our prototype and tested appears to be a strong foundation for a radical (and long overdue) overhaul of email's user interface. It is also a clear indication that life in the email habitat should be rethought not in terms of messaging, but rather in terms of the various activities users are trying to accomplish through that activity" [Bellotti, Ducheneaut et al. 2003].

Taskmaster is a well-designed system that performs its duties well, providing easy access to emails and as a result its development added to the research body that is looking to improve task management in email. Instead of trying to improve Taskmaster, we developed a system that looks at another dimension. Our email-based issue tracking system is not trying to improve accessibility; rather, it is acting like the useful personal assistant that brings information to your attention when needed and when necessary. As stated in Kwinana, Wentworth and Terzoli [2005] the emails are in your face when you need to act on them and will not get lost in many screen views of email messages. The user is informed, via a popular medium, of a pending task, of an escalation to higher authority or any other useful information regarding a task.

3.1.2. HP's Personal Email Assistant

The PEA provides a customisable, machine learning-based environment to support the activities of a major time sink of our daily lives – the processing of email [Bergman, Griss et al. 2002]. A PEA is an application or suite of applications that proactively monitor and manage a user's email. The system has been designed to be usable either with or without an agent-based infrastructure, and to be useful with a variety of email systems. In its current form, it leverages and augments the capabilities provided by Exchange and Outlook. It provides capabilities for:

- Smart vacation responder
- Junk mail filter
- Efficient email indexing and searching
- Deleting
- Forwarding
- Re-filing
- Prioritising of email.

HP's vision of the PEA is as a key piece of a larger Personal Information Assistant (PIA) that indexes and manages content from several personal information sources, such as email, local files, and bookmarks/favourites, and provides a unified search over the personal, local, and global information. In turn, they envision the PIA as a component of a Personal Assistant (PA) that assists users in handling many tasks, involving email, calendars, context, and personal information. In addition to general information indexing and search, the toolkits and services associated with the PIA can be used to respond to specific task-oriented queries, trigger actions based on the discovery of specific information or context changes (such as receipt of email, change in a web page or calendar, or completion of tasks), or construct task-oriented summaries of salient information for meetings, etc.

The PEA supports these key email-related tasks [Bergman, Griss et al. 2002]:

- Prioritise – the PEA uses classification and rules to prioritise incoming messages. The user may then view email sorted by priority in Outlook.
- Filter – the PEA uses classification and rules to filter unwanted mail, such as "junk" mail.

- Index/retrieve – the PEA maintains an inverted index of all current and archived mail that the user may search with a Google-like interface.
- Re-file – the PEA uses classification and rules to move messages to appropriate folders.
- Vacation response – a Vacation agent uses the contact and calendar agents to respond to incoming messages with appropriate vacation responses.

PEA is solving a different problem to ours as it concentrates on managing emails and, to an extent, the email environment, using different techniques amongst which are re-filing and prioritising emails. The makers of PEA understand that a variety of functionalities is needed in emails to support the user who is doing more than just sending and receiving few text messages a day.

Venolia, Dabbish, Cadiz and Gupta [2001] built a threaded email client that could help people process a large amount of unread messages. The prototype's thread list is categorized by day. The prototype groups emails that referred to one idea together and thus it makes referring back simple; an important feature for users who have large amounts of emails all containing different topics that need to be attended to as swiftly as possible.

3.1.3. TimeStore-TaskView

The TaskView research focuses on how to support management of pending tasks in email and explores alternative solutions that use different external representations of messages and associated tasks [Baecker, Grudin et al. 1995]. Its design was a follow up prototype to TimeStore. TimeStore, a novel, time-based email interface proposed and developed by Baecker, is an example of the first approach. Messages are automatically organized by time and by sender on a two-dimensional grid (as in Figure 3.1 and 3.2 below). The two-dimensional representation allows locating messages by using when the message was received and who sent it. The second type of timeline, the temporal reference of messages, is used to facilitate the management of pending tasks embedded in messages in TimeStore-TaskView. The TimeStore-TaskView interface is based on TimeStore and uses the same graphical representation. In TimeStore-TaskView, small icons on a two-dimensional grid represent tasks embedded in messages with temporal and other task information shown on the

horizontal and vertical axis, respectively. Other task attributes include sender, subject, or keywords extracted from message body (user selectable). Navigation back and forward in time is provided.

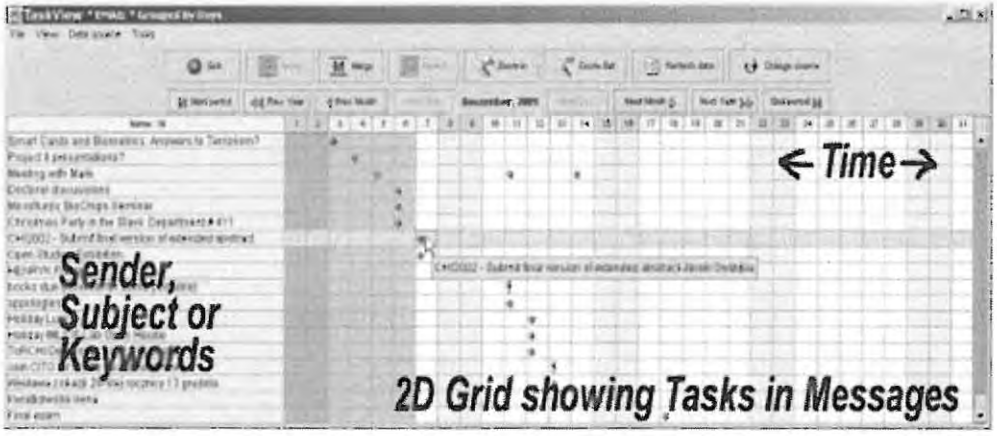


Figure 3.1: TaskView user interface [Gwizdka 2002a]

The main focus of this research is on the presentation of pending task information. TimeStore-TaskView presents active messages; that is, messages with future reference containing pending tasks. The study by Gwizdka [2002a] demonstrated the value of the TaskView interface and the benefit of task visualization on efficiency of finding information in messages related to pending tasks.

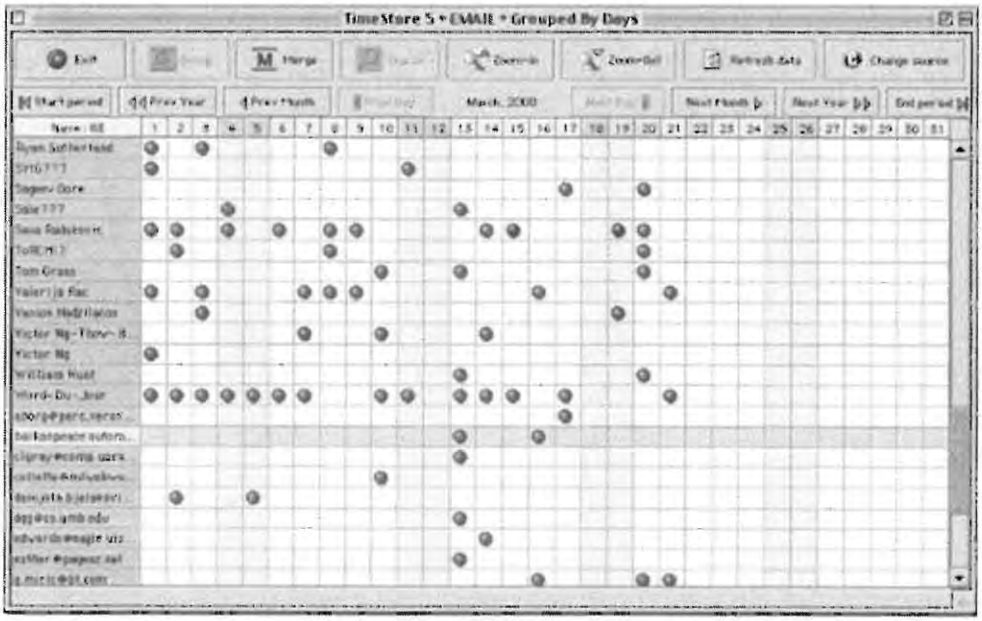


Figure 3.2: TimeStore interface – month view [Gwizdka 2002a]

3.1.4. Request v3: a modular, extensible task tracking tool

Request v3 is a task management tool that provides a selection of user interfaces, support for multiple database back-ends, flexible security controls, and extensive reporting capabilities [Rhett 2004]. It tracks tasks for administrators, provides information about the task to the users, supports the existing environment, works with the existing systems, and makes it easier for administrators to perform their jobs without adding to existing burdens.

3.1.5. TaskVista

A cognitive system was proposed that could support busy professionals in government or military roles in managing and even performing office and military tasks. This system would be capable of reasoning and learning and would be aware of and be able to explain its own behaviour as well as accepting direction from users. A possible embodiment of this type of system is a task list manager (TLM) called TaskVista that helps users manage and execute their to-dos [Bellotti, Dalal, Good and Bobrow 2004]. Users can easily create a new to-do by typing one in, or, dragging an item (e.g. a file or email) into the list. The (editable) title defaults to the subject or title of a dragged-in item to reduce the user's need to type. Additional items such as notes, documents, etc., can be dragged in to a to-do list, so the to-do list becomes a resource for saving content and launching activity on the task, like a pile or folder. But unlike a pile or folder, a to-do has computational properties that support task management, e.g. priority, time constraints. To-dos can have other properties that, for example, show location, task or participant dependencies, and whether they are part of a bigger project. Properties do not need to be specified up-front, and can change over time [Bellotti, Dalal et al. 2004]. TaskVista is a good tool to help people manage their to-dos, but its emphasis covers a different scope because our system wants the management to be based in emails.

3.1.6. ThinkDoc

ThinkDoc is a server-based service interposed between email senders and recipients in the email channel that is the network or Internet [Bellotti, Ducheneaut et al. 2000]. ThinkDoc intercepts email from a subscriber (using a variety of possible re-addressing mechanisms) and

invokes a service call that is embedded in the message (perhaps in an attachment). The recipients of the email receive a message with a reply-to address containing information to identify the respondent and the original message. Messages and attachments passing through ThinkDoc can be processed in various ways based on information embedded in messages, perhaps stripping out an attachment and replacing it with a URL, or perhaps creating an archived copy of the email, or sending a notification to someone. The following are examples of services that are possible [Bellotti, Ducheneaut et al. 2000]:

- Users can send all their email through ThinkDoc and use a ThinkDoc proxy email address for all contact with other users. In this mode, all their email comes through the service. ThinkDoc keeps archives of their email.
- Important emails and replies can be logged on the server providing a disinterested and trusted record of communications.
- Attached documents can be replaced with a link. Senders might update the document on the ThinkDoc server. Thus recipients of documents can always be sure to access the most recent version of the document.
- Users can forward messages to ThinkDoc to create new contact records for the sender perhaps returning a simple alias for future use.
- Users can forward messages to a ThinkDoc address to create a reminder that sends the message back in a specified amount of time (e.g., in 7 days).

ThinkDoc has advantages, which include:

- Infrastructure independent: it does not involve a single specialized software infrastructure across organizations (i.e. it uses common resources such as email and the web).
- Low cost: Does not involve large initial investment and extensive maintenance.
- Simple: Does not require extensive training and customization.

Below are some commercial names in business groupware, email and basic office management. The summaries below show what the different systems are capable of. Some of the systems below have made an attempt to support task management in email while others concentrate mostly on sending and receiving of emails.

3.2. Business groupware

3.2.1. Lotus Notes/Domino

Lotus Notes/Domino is a client/server-based email/groupware system from IBM/Lotus Software. The Notes part refers to the client, and Domino refers to the server in the system [Takkinen 2002]. Domino is an applications and messaging server with an integrated set of services that enables easy creation of secure, interactive business applications for the Internet and corporate intranets. The basic architecture of Notes was developed before the appearance of web browsers. Lotus Notes/Domino is based on the concept of shared document databases where documents can be stored, viewed, updated, replicated (explained below), and routed as required. Domino is a collection of server processes, including database replication, email routing, and indexing. There are two basic categories of databases. The first category is a shared database, which resides on one or more servers, accessible to many users. Databases can be copied to additional servers for easier access. The databases are synchronised across the network through replication. The second category is a local database, which resides on a workstation. Local databases are often personal databases, such as diaries or prototypes of new databases—or local replicas of remote databases.

Notes/Domino is characterised by the ubiquitous access to both messages and business data from any location, and via any device. Also, one common universal (unified) inbox is used for all forms of messages (email, fax, web pages, etc.) [Terzis and Nixon 1999 and Takkinen 2002]. Notes/Domino can be scaled to any size of organization. Its graphical user interface has a similar look and feel across all platforms (Windows 95, 98, NT, 2000, and Macintosh). Notes/Domino includes its own development environment for creating customised applications. All of the major Internet standards are supported, including security protocols such as SSL. The messaging system is based on client/server architecture and supports standard protocols like SMTP/MIME, x.400. Any kind of mail clients (e.g. POP3, IMAP4, MAPI, etc) can be used with Domino. Domino includes a workflow engine that allows distribution; routing and tracking of documents according to application defined processes. Through the workflow engine Domino enables co-ordination and streamlining of critical activities. Domino also enables automation of frequently performed processes with the use of agents that can be triggered by time or events [Terzis and Nixon 1999 and Takkinen 2002].

3.2.2. Microsoft Outlook/Exchange

Microsoft Exchange/Outlook is a client/server-based messaging system. The Microsoft Exchange server runs on a Windows Server system and provides functionality to the Exchange client, named Microsoft Outlook [Takkinen 2002]. Microsoft Exchange/Outlook is more focused on messaging than on task management. Nevertheless, Outlook provides some groupware functionality. Exchange includes capabilities for information-sharing (public folders) and calendar/scheduling functionality. Microsoft Outlook supports shared documents, forms, threaded discussions, and calendar/scheduling. It also offers basic personal information management (PIM) functions, i.e., the user can keep a calendar, add events and to-do items, schedule reminders, and jot down POST-IT-like notes. Any information being written down must always be tagged to a particular person or a task, i.e., no random information is possible. Standards supported by Microsoft Outlook include SMTP, POP3, IMAP4, and LDAP, in addition to MAPI (Messaging Application Programming Interface), which is Microsoft's own architecture for messaging applications (email, scheduling, PIMs, etc) [Takkinen 2002].

3.2.3. Novell GroupWise

Novell GroupWise is a client/server-based groupware product from Novell and one of the pioneers of the groupware field. It combines document management, email, group calendaring and scheduling, task management, imaging, and workflow in one integrated package [Takkinen 2002]. The architecture of GroupWise is divided into domains, run by Message Transfer Agents (MTA), with Post Offices contained within, and each serviced by Post Office Agents (POA). A web-monitoring interface allows an administrator to connect a browser to a POA or MTA to collect statistics and change configurations. The agents are server-based processes performing the tasks of dealing with the message flow through the system as well as the critical task of providing full text and profile indexing services for documents in GroupWise. The documents are encrypted, compressed, and stored as Binary Large Objects (BLOBs). GroupWise also has so-called Dynamic References, which are the automatic attaching of a referenced document to email messages sent outside the GroupWise environment. Approved users can search and browse another user's calendar and make suggestions for meeting times. So-called search folders can be created and saved so as to locate frequently used documents and email messages [Takkinen 2002].

The GroupWise client is supported on most platforms (Windows 98, 2000, and NT, as well as Unix, Linux and Solaris), but the server is only supported in Windows NT and 2000. Both can run on Novell's own platform Netware. A wide variety of protocols are supported (POP3, IMAP4, LDAP, and Novell's NDS directory services), in addition to Secure Sockets layer (SSL) and Secure Multipurpose Internet Mail Extension (S/MIME).

3.2.4. Memo

Memo from Nexus began as a corporate-based email system in the mid-1980s, originally developed by Volvodata. Memo was previously marketed as Verimation. The IBM OS/390 operating system is today used as the server. The client is available for a number of platforms (Windows 95, 98, NT, and 2000). Memo has calendar/scheduling functions, and document sharing. Furthermore, it uses electronic forms, called Memo forms, to manage tasks. A "workflow item" arrives in the Memo mailbox just like ordinary email messages, forms and calendar invitations. Memo enables customised applications that can include workflow, data integration with legacy systems, and transaction integration via the intelligent forms capability and its published API. Memo is classified as a hybrid system, providing the advantages of client/server email in functionality and the advantages of a host-based Managing Tasks Asynchronously system in scalability, manageability, speed of upgrading system, and lower administrative cost. It also includes a Java email client. A new generation called Memo Open Client supports key Internet protocols, including POP3, IMAP4, and LDAP. The Memo Open Client is supported on Windows 95, 98, NT, and 2000 [Takkinen 2002].

"Other systems that were employed were the Object Lens System, which was the first system to allow users to filter their email. The Coordinator system attempted to increase the meaningfulness of email messages by requiring senders to classify the messages they sent in terms of a number of *speech acts*" [Zhao, Kumar et al. 2000]. Motiwalla and Nunamaker [1992] studied the use of knowledge-based email systems as a tool for supporting managerial decisions. Several studies have been conducted on information filtering methods in the context of email for document filtering and for document sharing. These studies focused on full text matching based on keywords with various weighting schemes. Motiwalla proposed an intelligent system for prioritizing email received based on personal preferences (or profiles).

3.2.5. Microsoft BizTalk

BizTalk does not fall strictly into the category of task management as the rest of the systems in the section of related work; nonetheless it is worth mentioning because its core function is similar to our email-based issue tracking system. BizTalk Server 2004, a Windows Server System product, helps customers efficiently and effectively integrate systems, employees, and trading partners. BizTalk is a middleware product that enables integration between line of business (LOB) applications and systems in an Enterprise Application Integration (EAI) scenario and trading partners in a business-to-business scenario. As part of this broad area of responsibility BizTalk is able to execute business processes that implement complex integration or workflow scenarios, transform data from one format into another, route messages based on message metadata or actual content, perform security duties such as encryption and signing, capture tracking information for administrators or business analysts, communicate with a variety of communication protocols and mechanisms to name but a few of its capabilities [Arch Hacker 2004]. Because BizTalk enables companies to integrate and manage business processes by exchanging business documents, it has a similar function to the email-based issue tracking system, especially when we use workflows and tracked emails to coordinate reservations of cars and / or accommodation.

The first part of this chapter has drawn a picture of the state of task management research in email. The scope ranges from personal research, academic research, to research that is commercial and that made a profit. As we have discussed the different research aspects of task management, we turn our attention again to our methodology, eXtreme Programming. We discuss how we made use of XP in developing our email-based issue tracking system and more. But before that we discuss RhoTrax, the issue tracking system that we use during our implementation. We place it here because it also falls under systems that have been developed to assist task management.

3.3. Liberum and RhoTrax

RhoTrax (the Rhodes Tracking System) is a system built around an open-source web-based issue tracker called Liberum by a group of students from Rhodes University with the author of this thesis being part of that group. The author currently plays the role of the administrator

of RhoTrax, correcting any unforeseen errors ever since the development project and ensuring that the system is working throughout. The aim of the project was to add features to the existing Liberum helpdesk and extend the system by providing a web service, so that the functionality could be easily accessed by other software systems. Liberum is an open-source help desk that provides a simple, easy to use web interface for managing and tracking technical support problems. The help desk is written in HTML and ASP and is easily modified and customised. The helpdesk is run using Windows NT, 2000 or XP running IIS [Liberum 2000]. In the RhoTrax extension project, a web service layer was added so that all the core functionality of Liberum is exposed as a web-service for use by other software. The functionality exposed is then used to support the rich clients for the users and the reps. Figure 3.3 below shows how the components fit together.

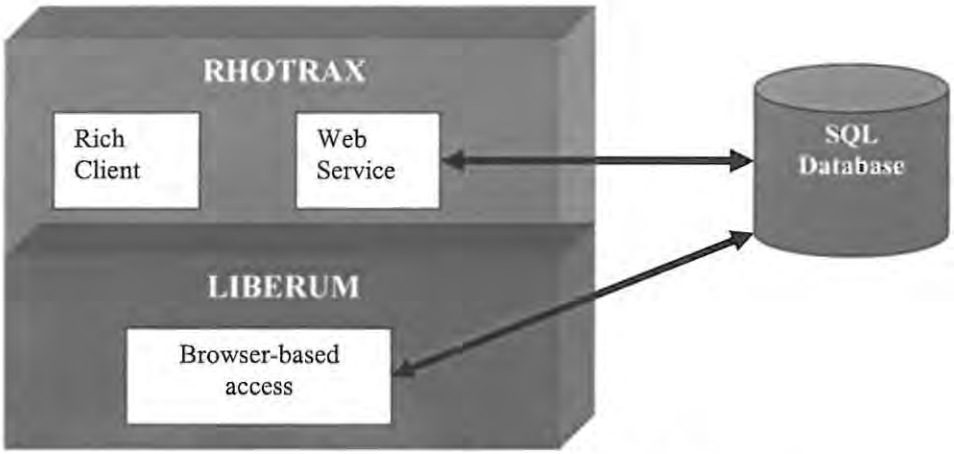


Figure 3.3: How Liberum and RhoTrax are related

The RhoTrax web service, Figure 3.4 below, provides access to the Liberum issue tracker. For our purposes, its web-service component contains the methods that will be called on when an email is intercepted and tagged for tracking.

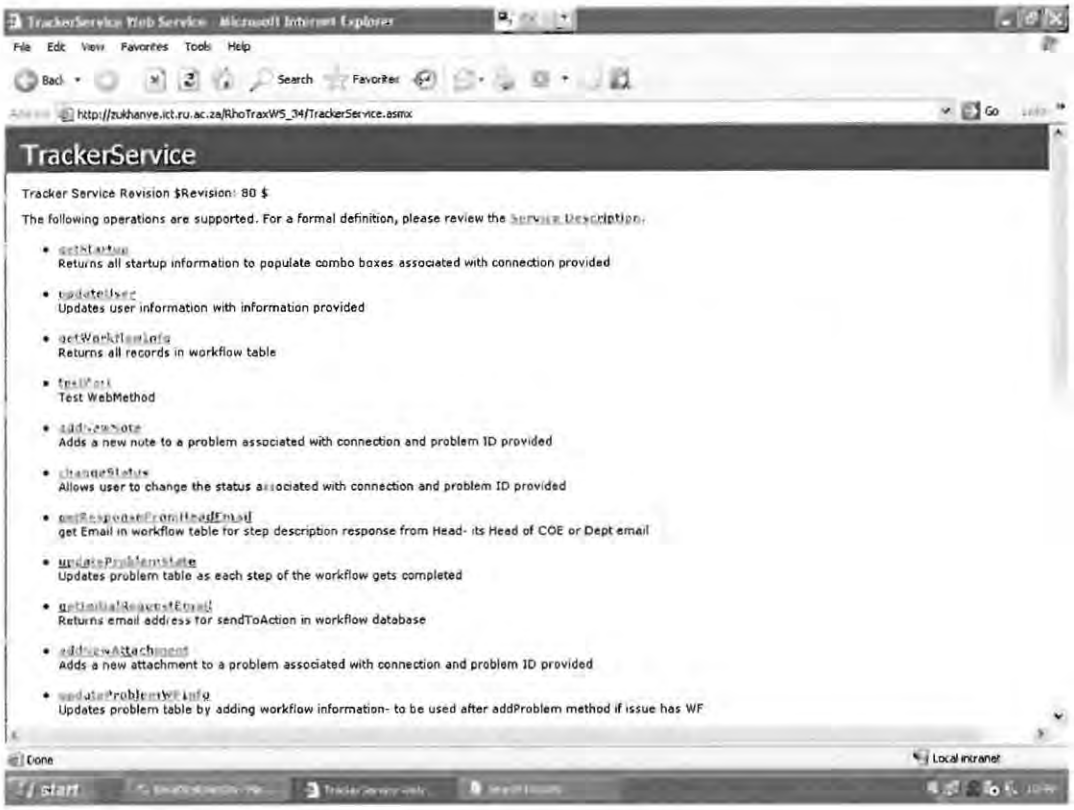


Figure 3.4: The web service called TrackerService

Liberum was used as a basis for RhoTrax because it contained much of the basic functionality needed for issue tracking. The availability of source code meant that we were also able to add additional features, such as allowing attachments, reminders and escalation, to the basic Liberum issue-tracking functionality. Within our organization we already had experience with Liberum, with our technicians using it to administrate the organization's technical requests and trouble tickets. RhoTrax is similar to the helpdesk systems available on the market that are used by system administrators and technicians to manage problems and their resolutions in an organised manner [Gwizdka 2002, Liberum 2000].

The system has *reps* (attend to issues submitted by users) and users. The user will submit an issue and the rep will attend to it. Below is a list of the features that the issue tracker has:

- The user can submit a new issue.
- The user can get the list and details of the issues associated with them.
- The user can see and update their personal information.
- The user can change and use a different database of issues whenever necessary.
- The user can drag and drop a file or files that they associate with a particular issue.

- The user can add notes to an issue they are associated with.
- The user can close the issue once it is completed.

RhoTrax has been designed to have a user interface for both reps and users. The rep's user interface shows the issues related to him, but he can also see the issues related to other reps by using the drop down list on the top centre of the screen. A user can only see the issues related to them. Figure 3.5 below shows the user interface. Also in the picture, along the top you can change which database you are using by simply clicking on the one you want to use.

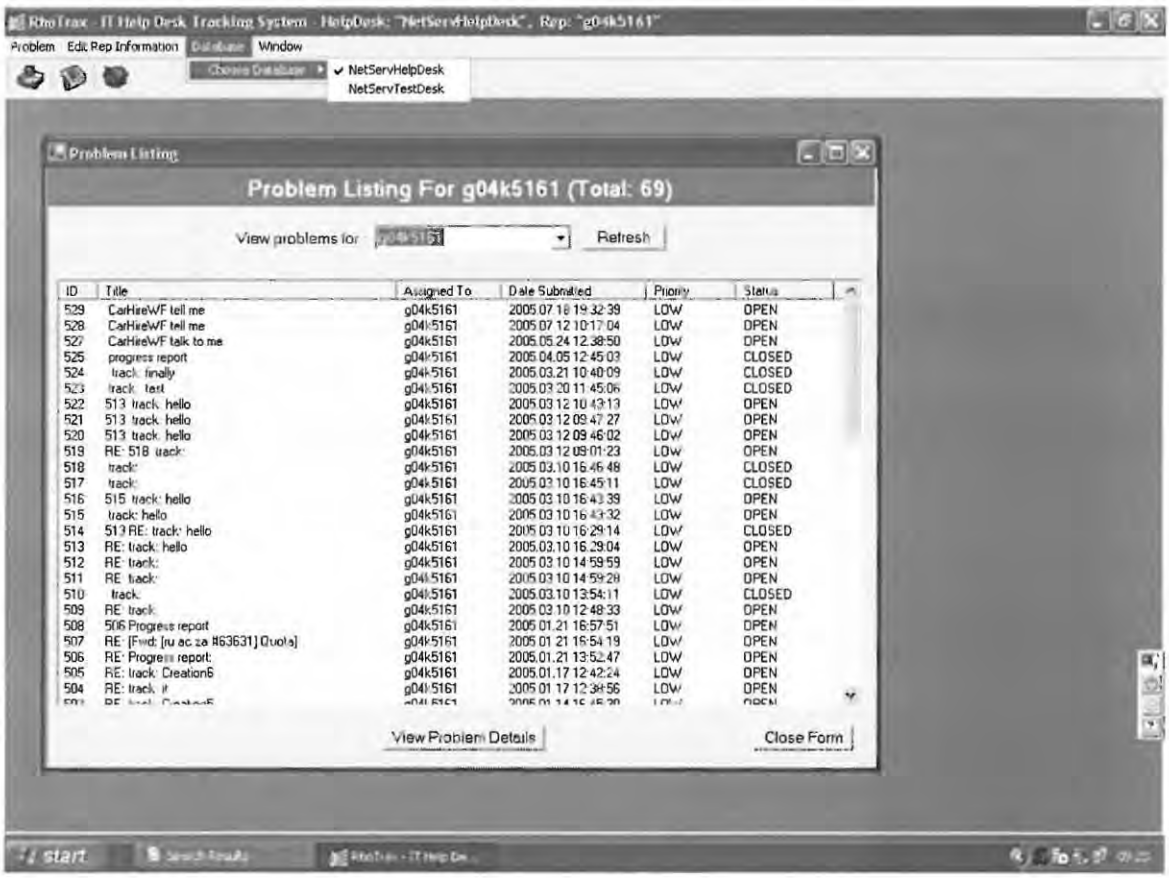


Figure 3.5: RhoTrax user interface for all problems associated with a particular rep

Double clicking on each issue shows its details. Each issue in RhoTrax has the following variables recorded and its problem details user interface (Figure 3.6) is shown below:

- Issue description
- Issue detail
- Issue number (ID)
- Attachment

- Note
- Category
- Date open
- Due date
- Who entered the issue
- The department they work in
- Their location
- Priority
- Solution
- Status

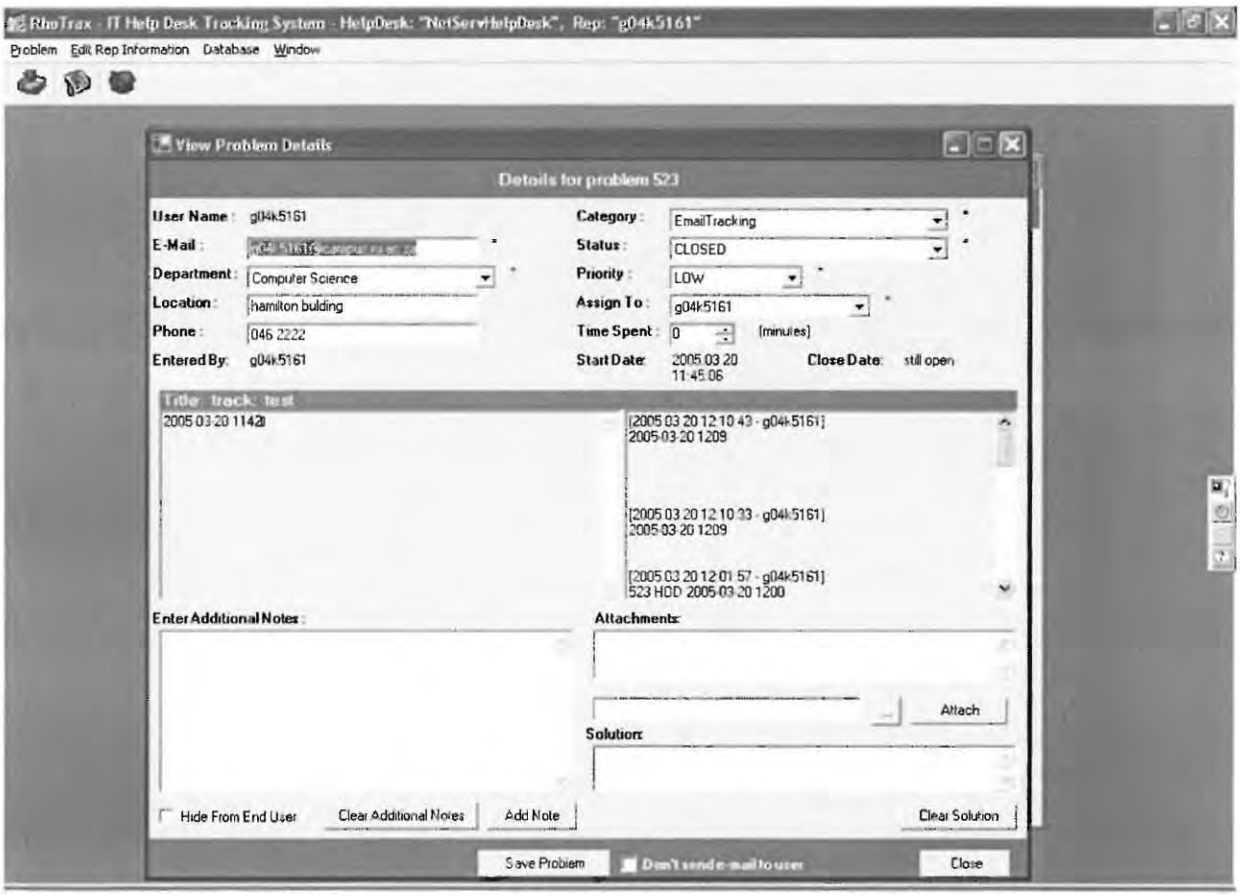


Figure 3.6: Details of a particular problem

The user can add notes (or comments) about an issue either via the user interface above or by using the web form accessible when clicking one of the links appended to the tracked email. You can also attach a file to an issue that it is related to.

The rep and users can update their personal information using the user interface (Figure 3.7) below.

Figure 3.7: Rep details

Figure 3.8 below illustrates the tables RhoTrax has in its database that are used by RhoTrax and MailTrax. MailTrax is the name used to describe the issue-tracking functionality that intercepts emails and if they are tagged, tracks them. The most important table is the `problems` table (Table 3.1) because everything is related to it. The rest of the tables are small and play a supporting role to the `problems` table.

Problems		
Field	Data Type	Description
id	int	Unique ID of a problem
uid	varchar	User ID
uemail	varchar	User email
ulocation	varchar	User location
uphone	varchar	User location
rep	int	Rep ID
status	int	Problem status
Time_spent	int	Time spent on problem
category	int	Problem category
priority	int	Priority of problem
department	int	User's department
title	varchar	Problem title
description	text	Problem description

solution	text	Problem solution
Start_date	datetime	Problem start date
End_date	datetime	Problem end date
Entered_by	int	Who entered problem
kb	int	Should it be added to knowledge base
workflow	char	Type of workflow attached to
lastExecutedState	int	If workflow, last executed state
dueDateWF	datetime	Due date of workflow

Table 3.1: Problems table

The other tables consist of the following columns (with the primary key underlined):

- tblNotes (id, note, addDate, uid, private)
- categories (category_id, cname, rep_id)
- status (status_id, sname)
- priority (priority_id, pname)
- departments (department_id, dname)
- tblConfig_Email(ID, type)
- tblConfig_Auth(ID, Type)
- tblConfig (SiteName, baseURL, AdminPass, EmailType, SMTPServer, HDName, HDReply, BaseEmail, EnablePager, NotifyUser, EnableKB, DefaultPriority, DefaultStatus, CloseStatus, AuthType, Version, UseSelectuser, UseInOutBoard, KBFreeText, DefaultLanguage, AllowImageUpload, MaxImageSize)
- tblUsers (sid, uid, password, fname, email1, email2, phone, location2, location2, department, IsRep, dtCreated, dtLastAccess, ListOnInoutBoard, firstname, lastname, inoutadmin, phone_home, phone_mobile, jobfunction, userresume, statustext, statuscode, statusdate, language, RepAccess)

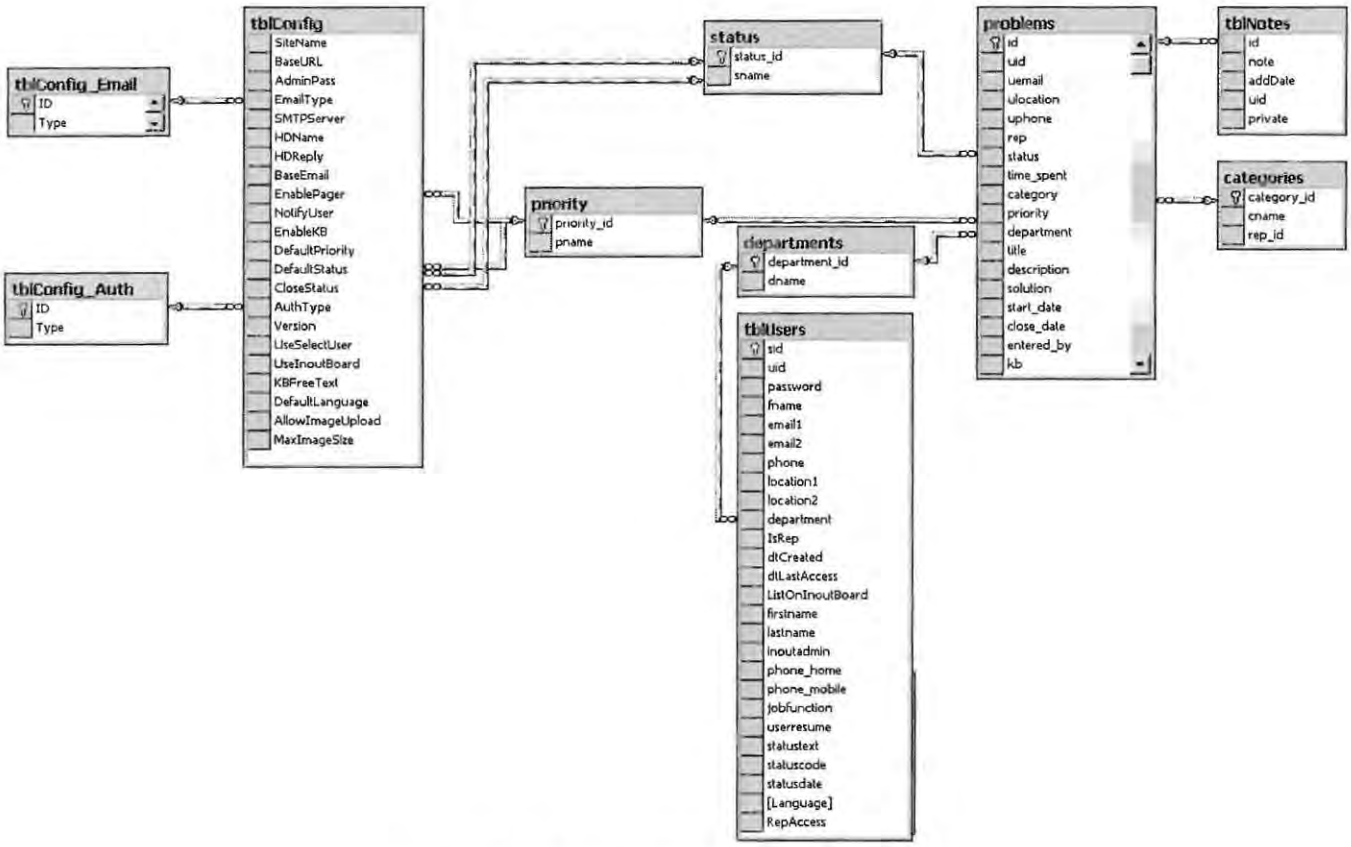


Figure 3.8: Tables used for RhoTrax and MailTrax

We could foresee early in the project that we would need to eventually add new data into the existing RhoTrax tables, and new methods to the web service, so the advantages of being able to build on an open platform that was already known to us were significant.

With RhoTrax we can now call on its web service's method to record an issue that has been tagged for tracking using the code:

```
public static string makeNewIssue(string subject, string body)
{
   try
   {

      string url= "http://zukhanye.ict.ru.ac.za/RhoTraxWS_34/
                   TrackerService.asmx";
      IssueTracker it = new Issue Tracker ("NetServHelpDesk","g04k5161",
                        url);
      Problem p = it.AddProblem
                  ("g04k5161","g04k5161@campus.ru.ac.za",
                   "hamilton bulding","046 2222","EmailTracking","LOW",
                   "Computer Science", subject, body);
      return p.problemid.ToString();
   }
   catch (System.Exception d)
   {  MessageBox.Show(d.StackTrace, d.Message);
      return d.Message;
   }
}
```

Code segment 3.1: MakeNewIssue method

3.4. eXtreme Programming

We developed the email-based issue-tracking system using eXtreme Programming (XP)'s methodology of software development. "XP is a light weight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements" [Beck 2000].

The fundamentals of XP include [Beck 2000]:

- Distinguishing between decisions that must be made by business interests and those to be made by project stakeholders.
- Writing unit tests before programming and keeping all of the tests running at all times
- Integrating and testing the whole system – several times a day.
- Producing all software in pairs, two programmers to one screen.
- Starting projects with a simple design that constantly evolves to add needed flexibility and remove unneeded complexity.
- Putting a minimum system into production quickly and growing it in whatever directions prove most valuable.

XP practices, in Figure 3.9, are used during a software development project and are listed below [Wells 1999]:

- User Stories – The customer has specified all the desirable features of the system as stories.
- Iteration and Iteration Planning – Within a release, the customer picks the most valuable stories for the programmers to work on for the next few weeks.
- Release Plan and Release Scope – The customer has to choose the smallest scope with the most immediate Business Value for each Release of the system, and the programmers will put it into production as quickly as possible.
- Story Estimate and Load Factor – The programmers will estimate the time they need to implement each story, and at the end of an iteration will compare the estimates against calendar days. This tracks their progress in implementing stories, and gives feedback and improves the ability to estimate over time, so that both the team and the customer develop a sensible idea of how many stories the team can implement per week.
- New Stories – If the customer's requirements change, new stories can be written, estimated, and added to the collection of unfinished stories. At the next release planning iteration, these can be given appropriate priority.
- Functional Tests – For each story in an iteration, the team will help the customer write automated tests that demonstrate to the customer's satisfaction the code is implementing the features in the user stories.
- The Seed – At the end of the first iteration the team will have a system that is recognizable as the final system. Not everything will work, but some stories will.
- Continuous Integration – At the end of each iteration the team will produce a system where the features that are implemented are ready for production.
- Unit Test – The team will write tests as they write the code, so tests can crosscheck with the customers.
- Refactoring – The team will continually evolve the design of the system: adding flexibility where it is needed, removing complexity where it doesn't help, and unifying duplicated code. This will help the team to continue delivering changes at reasonable cost for the life of the systems.

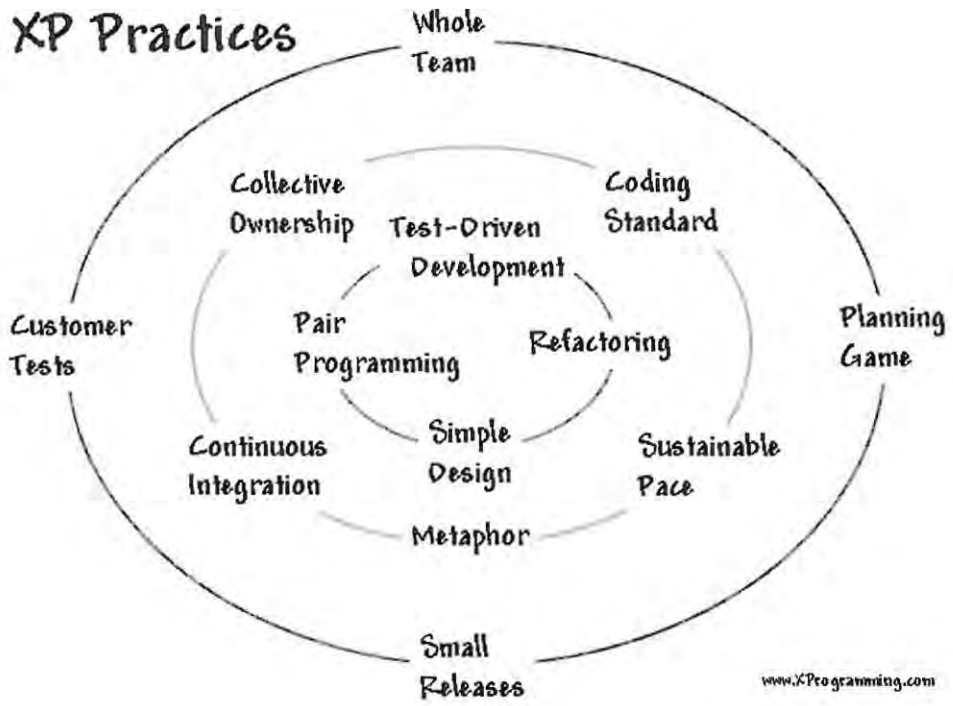


Figure 3.9: XP Practices [XProgramming 1999]

XP is sometimes described, as in the picture above, as an onion with the inner layer representing programming style (simple design, testing, refactoring coding standards), the middle layer represents team-oriented practices (collective ownership, continuous integration, metaphor, coding standards, 40 hour week, pair programming, small releases) and finally the outer layer contains the elements for how the team interacts with the customer (onsite customer, testing, small release, planning game) [Wake 2000].

Our system is incremental in its design so as to closely follow the design ideas of XP. In addition, there is an emphasis on trying to use existing high-level components and services, so that the key activity in the development centre is on assembling and interfacing together existing components, rather than building them from the ground. XP is different from other methodologies because [Beck 2000]:

- Short cycles can be achieved due to early, concrete and continuous feedback.
- An overall plan quickly comes up, due to its incremental planning approach, which is expected to evolve through the life of the project.
- It has the ability to schedule the implementation of functionality, responding to changing business needs in a flexible way.

- In order to monitor the progress of development, to allow the system to evolve, and to catch defects early it relies on automated tests written by programmers and customers.
- To communicate system structure and intent, it relies on oral communication, tests and source code.
- It relies on an evolutionary design process that lasts as long as the system lasts.
- It relies on the close collaboration of programmers with ordinary skills.
- It relies on practices that work with both the short-term instincts of programmers and the long-term interests of the project.

Developing a system needs planning and the design of its architecture. XP does not focus on architecture, but there are a few of its practices that address architecture. Spike solutions are created to figure out answers to tough technical or design problems so as to reduce the risk induced by a technical problem and increase the reliability of a user story's estimate. An overall system metaphor helps in discussions and decision-making about features and implementation; a metaphor for RhoTrax is an issue-tracking system. With iterations and small releases, we have a releasable system that has some features working but not all. Removing redundancy and cleaning up code as we progress does have a positive effect on the architecture. Finally the behaviour of the team and its generally unwritten rule also assist in shaping the architecture of the system.

3.4.1. Why eXtreme Programming was chosen

As described above, XP was conceived and developed to address the specific needs of software development by small teams facing vague and changing requirements. This is one of the reasons why XP was chosen for our developments. We are building and developing something that has not been thoroughly done before. XP is being used as a safety net to ensure that we progress without fear but also having the confidence to make changes at any point in the development – after all this is what a Masters researcher needs.

Ideally XP is appropriate for a small group, but this time an individual is using XP and thus XP's pair programming practice, will not be used. Because there is no extra pair of eyes, and mistakes are to be expected, test-driven development (TDD) will help ensure that the code remains as free from coding defects as possible. TDD will ensure that no code is written

unless it is necessary and there is a test that fails which the new code caters for. TDD will ensure lean code that is necessary. Together with refactoring, the code will be just what is needed for that particular functionality. The user stories will tell us which direction to follow for how long, as we know they can change and new stories added. This is typical of researchers and so it is a welcome change to see that XP embraces this and has ways of working with it.

Functional tests are there to keep the customer happy. It is of no use to give a customer code and to tell him that "it works". They are also good for the developer because it gives her a sense of accomplishment and achievement. Another thing that will boost morale is the seed. This is the system that after each iteration is recognised as the final system where not everything works but a little of everything does to show that progress is being made. At the end of each iteration, a system will be produced where the features that are implemented are ready for production.

The researcher, together with the supervisors, is the customer of the end product. It is however the responsibility of the researcher to provide both unit tests for the code and acceptance tests. The researcher and the supervisors come together and discuss the functionality to be added for a particular production cycle, and the researcher obliges (after discussing the effects for the rest of the system). The tests help the researcher know that all the software is running, at any time. This security is priceless for a researcher whose train of thought constantly changes, and functionality to the final system is constantly under scrutiny and usually constantly changes. This ensures the integrity of the system. It is good for a researcher to quickly put the system into production as soon as possible. This helps us see what is worth growing, and in what direction, which proves to be valuable. This is the flexibility XP provides us.

3.5. Test-driven development

Kent Beck defines test-driven development using the following rules [Newkirk and Vorontsov 2004]:

- Never write a single line of code unless you have a failing automated test
- Eliminate the duplication

"TDD is a best practice that involves producing automated unit tests for production code, before you write production code" [Adaptation Software]. TDD is the partner practice of refactoring. Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behaviour. Test-driven development (TDD) is a style of development where [Coad 2002]:

- You maintain an exhaustive suite of unit tests
- No code goes into production unless it has associated tests
- Tests are written first
- The tests determine what code you need to write

Unit tests are tests that are concerned with a small part of implementation-orientated functionality. They are written to test that the classes written exhibit the proper behaviour. Developers who write the code being tested write unit tests. Acceptance tests however test that features operate correctly. Acceptance tests are concrete scenarios that exercise the system in a typical fashion. They are independent of the developer and are written by whoever defines the features (the customer). XP strongly believes that a feature does not exist until there is a suite of tests to go with it. Everything in the system has to be testable as part of the safety net that gives confidence and courage. "Confidence that all the code tests clean gives you the courage (not to mention the simple ability) to refactor and integrate" [Coad 2002]. When you have a task to do (i.e. some bit of functionality to implement) you write code that will test that the functionality works as required before you implement the functionality itself. Furthermore, you write a little bit of test, followed by just enough code to make that test pass, then a bit more test, and a bit more code etc. "By writing only the code required to pass the latest test, you are putting a limit on the code you will write. You write only enough to pass the test, no more. That means that you do the simplest thing that could possibly work" [Coad 2002].

3.6. Summary

In this chapter we introduced research that has been done for task management using email. This ranged from commercial to non-commercial systems. These showed us that there have been attempts to improve the user interface that individuals are used to, to facilitate task management. This also showed that our direction in trying to achieve task management in

email has not been attempted before. We introduced RhoTrax, the issue tracker we are going to use as a foundation to our email-based issue tracking system. We introduced XP, the software development methodology that we will be using in implementing our email-based issue tracking system. A very close associate of XP, test-driven development, was then discussed briefly because there is minimal theory surrounding it – it is mainly implementation and thus will be discussed in chapters to follow.

Chapter 4

Iteration 1: simple email-based tracking

4. Introduction

With a large number of software development theses, a chapter on the design of the system is expected, after which there is the implementation chapter. XP is different because, for adaptable projects with fast-changing requirements, XP steers away from a thorough design of a system before implementation. As a result in this chapter and the following one, we will discuss the little design we do for a feature, and the resulting implementation. We will be discussing an XP iteration as it happens, starting with a discussion about how XP handles a new project, including architecture when developing a system. User stories are introduced after which programmed *spike solutions* (or quick prototypes) are discussed. A spike solution is a small throwaway program that solves a very specific issue, e.g. "how we can intercept the email, and find the tag." Spike solutions are used to reduce the risk of a technical problem and increase the reliability of a user story's estimate because of the increase in understanding the technical problem. Issue tracking is discussed first at the mail server side and then at the mail client.

4.1. How XP handles a new project

"XP is a relatively new agile approach to developing software. An XP team uses an on-site customer, a particular planning approach, and constant testing to provide rapid feedback and high bandwidth communication" [Beck 2000]. In an XP project, the customer contacts the development team to discuss a particular project. Quite often it is expected that the customer is not sure at this stage what features will best solve the problem, so the needs are expected to be refined over the lifetime of the project. When the system that the customer wants has been discussed, the features are described via user stories. The customer decides which stories will be attended to and as a result which features are going to be included in the next iteration, and sits in during development. The customer can decide to change the order in which the stories are going to be implemented or to replace the stories altogether. It is the responsibility of the programmer to explain the effects that each decision will have on the scope, the quality of the system, the date of the release and so forth. The team focuses on exploring the difficult sections of the system using spike solutions and programming as well as the release plan (the smallest scope with the most immediate business value for each release of the system). The

customer writes tests and answers questions while programmers continue developing the system. This continues until the system is finished, each iteration providing about two weeks of code and a releasable system that has some features working but not all of them. Table 1 below shows the different decisions the customer and the programmer make.

Customer decisions	**Programmer decisions**
Scope	How long it takes to add a feature
Priority	Technical consequence of decision about features
Composition of release	How team works
Dates of release	Detailed schedule

Table 4.1: Decisions made by the customer vs. the programmer

A release represents perhaps one to three weeks' work. During release planning the programmer writes and estimates a story, and the customer splits a story to manageable sub-stories so that the programmer can do a quick prototype for the story. In the end the customer has a list of the desired features and the programmers breaks the features to stories and provides each story with an implementation cost. The customer sorts the stories by assigning values to them, programmers estimate how long each will take and the customer chooses the stories with the most immediate business value. Once the customer and the programming team have agreed on the scope of the release and the stories are firm, the implementation can proceed.

4.2. The user story

To continue with the system development, we present user stories to show what we want from the system:

- *Sally emails Jack asking him to please reserve a venue for a particular meeting and includes all the necessary information. She also tags the email's subject with a keyword. The email is delivered to Jack with two links added at the bottom of the email body. When Jack receives the email and reserves the venue, he will click the corresponding link to close the issue. An email will be automatically sent to Sally telling her that the task has been accomplished.*

- *Jeff asks Jenny to please pick up a parcel from the department's secretary for him. He tags the email's subject to ensure the request gets tracked. The email is delivered to Jenny with two links added at the bottom of the email body. Jenny goes to pick up the parcel, but finds out from the secretary that the delivery company will delay for 24 hours. Jenny will now go to her email, click on the corresponding link, open a web form and add the delay information to the web form, and set a reminder. An email will be automatically sent to Jeff informing him of the delay and when the parcel is expected. After 24 hours Jenny gets a reminder email from the system to go check up on the parcel and Jeff will be copied on the email to let him know that Jenny was reminded. Jenny will pick up the parcel and return to her emails and click the corresponding link and close the issue.*

Figure 4.1 below is a diagram of the Finite State Machine (FSM) of the user stories described above showing all the states the issue can be in mainly OPEN (when the email has been sent to the recipient), DELAY (if task cannot be accomplished immediately) and CLOSED.

This is the time where the development team estimates how much effort each story will take, and how much effort the team can produce in a given time interval (iteration). The first iteration is a difficult one to estimate because there are so many unknowns. A factor to consider is that the development team consists of a single developer, and she will be developing all the user stories and quick prototypes. As a result, the usual time of an iteration taking two to three weeks meant having less deliverables.

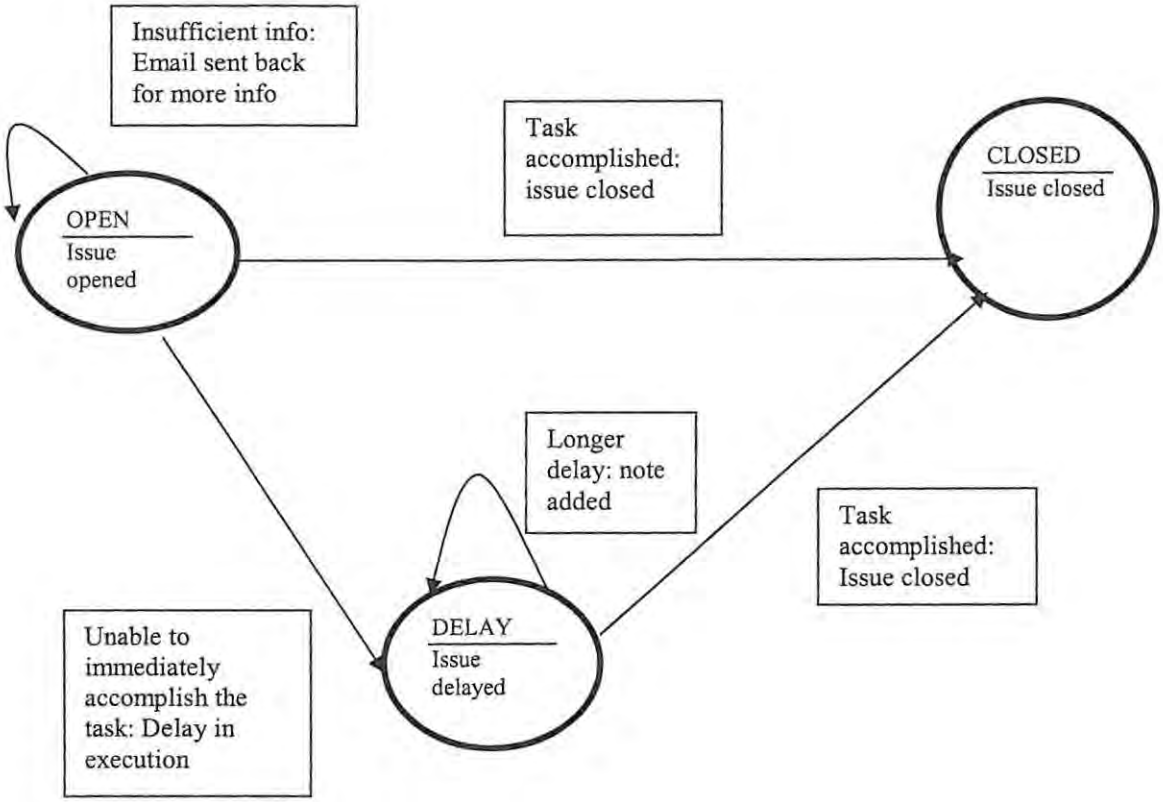


Figure 4.1: Finite state machine for simple tracking user story

With these user stories it is now possible to move forward and make decisions about our system. In order to assign the right "price tag" to the user stories we first have to break up the story into manageable bits. The feasibility of some aspects of the system may not be obvious, and if they pose a high risk to the outcome of the project, we should first develop quick prototypes. The user stories above do not help clarify where the tracking functionality is going to be, whether at the mail client or at the mail server and so this needs further investigation. After the discussion, a quick prototype that accesses emails within Microsoft Outlook is discussed. This seems to be a significant milestone presently and if we manage it, we will be closer to making more definite decisions.

4.3. Mail client side vs. mail server side interception and tracking

Interception of emails can either occur at the mail server or at the mail client. When this project started, we decided to explore both routes and make a decision once we had more information and experience with both of them. The main reason behind mail client interception and tracking was to open the functionality to people outside our organization. We chose Microsoft Outlook because it is a commonly used mail client within our organization.

Another reason for choosing Outlook is that it has the functionality to allow us to intercept emails at the client side. The MailTrax add-in to Microsoft Outlook can be accessed and installed by anyone who wants to use our tracking system. This meant that we could track issues delegated to other people in other organizations as well as our own. With this thought we move to our first spike solution.

Interception and tracking at the mail server has the advantage of the whole functionality being centralised. If there are any changes to the email-based issue tracking system, the update will occur once and be taken care of by an administrator and not the ordinary user. This will save time as well as prevent the user from making a mistake that will need an administrator to correct. There are a lot of impersonators all over the Internet that maliciously attack the network. With server side interception and tracking, ordinary users will be less inclined to believe a fake email from an "administrator" telling them to run a particular hyperlink on their machines. On the other hand, with mail server tracking, if a person wants to use the feature, they will have to change their configurations to use our specific mail server where the tracking functionality resides and this is not ideal.

4.3.1. Spike solution 1: accessing items in Outlook using the Outlook object model

Microsoft allows us the opportunity to create custom Microsoft Outlook objects and manipulate those objects from within Outlook or from another application. The Outlook object model exposes Outlook objects, which we can use to gain programmatic access to Outlook functionality [Microsoft 2001]. With the use of the object model we accessed the different folders within Outlook, that is, the public folders and all mailbox folders. Mailbox folders contain all Outlook built-in and custom folders. Each folder and sub-folder contains Outlook item objects, for example MailItem objects, ContactItem objects, JournalItem objects and so on.

Our first spike solution was to access the different items, more specifically the MailItem, within Outlook and display information about them like the sender, receiver, the date and time sent and so forth. This was successfully implemented and we gained confidence that we could successfully manipulate the contents of emails. We were a step closer to assigning realistic deadlines to our user stories. It was estimated that the spike would need two days to be

accomplished. After two days we had a working spike that was a small but significant piece in our first iteration.

4.3.2. Spike solution 2: mail server interception

Server side tracking ensures that the interception and resulting tracking is central and therefore easy to control. We wanted a mail server but could not use the department's mail server for two main reasons:

1. We cannot experiment using other people's mail, especially when it involved recording the contents of the email.
2. The mail server used in the department is quite large (needs a machine dedicated to it) and needs time to learn and administer.

There are a large number of mail servers available but our target mail server has to be easy to use and very lightweight and reliable. Microsoft Exchange Server and MailEnable Mail Server were investigated to see which has the better benefits for our scenario.

Exchange Server, the Microsoft messaging and collaboration server, is software that enables the user to send and receive electronic mail and other forms of interactive communication through computer networks [Microsoft 2005]. The recommended system requirements needed to support the installation and use of Exchange are [Microsoft 2005]:

- 500 MHz or higher processor
- Windows Server 2003 operating system
- 512 MB of RAM or more
- 500 MB on the hard disk where Exchange Server 2003 is installed
- 200MB on the system drive

"MailEnable is a lean and robust mail-server, designed as an enterprise level mail solution, with only modest system requirements" [MailEnable 2004]. MailEnable requires that the operating system can be any of the following [MailEnable 2004b]:

- Windows 2003 Standard Edition, Web Edition or Enterprise Edition
- Windows 2000 Advanced Server, Server or Professional
- Windows XP Professional
- NT 4 Server or Workstation

This meant that we could use the already existing Windows XP Professional and did not have to install a new operating system or have a machine dedicated just to being a mail server. Other requirements include:

- Intel Compatible Hardware
- 128MB RAM, 100MB hard disk space
- Microsoft IIS v5.0 or greater for Web Mail and Administration capabilities

Other than the advantages listed above, MailEnable Standard Edition server was chosen because it offered:

- Robust SMTP and POP3 services for Windows NT/2000/XP/2003 systems
- POP3 service supports RFC 1939, APOP secure authentication and grant or deny access for IP address ranges
- SMTP service supports RFC 821, 974, 1869, 1870, 2554, 2821
- Pickup events so you can run scripts or executables on an email
- Redirection
- Extensive logging
- Configuring mail services through easy to use Microsoft Management Console application
- Diagnostic utility to determine faults
- Configure redirection for email addresses and domains

After the server was successfully set up and functioning we investigated a way of extending its features. When an email is sent through the server, it goes via a number of modules, called connectors in MailEnable terminology, which facilitate its proper delivery. One of these connectors is the Mail Transfer Agent (MTA). The MTA Service is responsible for the routing of email in and out of MailEnable. When an email is accepted by the SMTP or POP retrieval service, the MTA service determines which connector is used to handle the email. For instance, if the email is destined for a mailbox, the MTA service will send the email to the PostOffice Connector in order for it to deliver the mail to the correct mailbox. If the email is destined for a list, the MTA service will send the email to the List service.

When the MTA moves a message between connectors, an optional executable file can be attached to run on the so-called pickup event. The MTA pickup event passes the mail message

filename to the external application [MailEnable 2004a]. This pickup event is used to intercept the emails, and retrieve some information from the email and store it for use during tracking of the task in the email. The external application has an opportunity to modify the email. Figure 4.2 below shows the window that contains the MTA properties.

Figure 4.2: Mail transfer agent properties

The emails that need to be tracked will be tagged by particular text in the subject of the email. If the email contains the text *track:* then our interceptor will open a new issue in the issue tracker, as well as adding two URLs to the bottom of the email. These links will refer back to a web service, and will contain a unique ID that will identify the issue in the tracking system. The first link will open up details of the recorded issue and the recipient can add more information about the issue. The second link will be used to close the issue when the recipient has accomplished the task. The main advantage of mail server side interception is that it is mail client independent and does not force the users to, for example, use Microsoft Outlook. Below is a piece of the code used by the MTA for intercepting an email and, if it is tagged, recording its content in the tracking system.

```
static void Main(string [] args)
{
if (args.Length < 2) return;
string MailFileArg = args[0];
tring ConnectorArg = args[1];
RegistryKey reg =  Registry.LocalMachine.OpenSubKey("SOFTWARE\\Mail
                   Enable\\Mail Enable");
string MailDataPath = (string) reg.GetValue("Data Directory");
string MessageFilePath= MailDataPath + "\\Queues\\" + ConnectorArg +
                        "\\Inbound\\Messages\\" + MailFileArg;
 CDO.Message msg = LoadMessageFromFile(MessageFilePath);
 string people= msg.To + msg.BCC + msg.CC;
 if (msg.Subject.IndexOf("track:") > 0  || msg.Subject.IndexOf("Progress
                                          report:") >0  )
 {
   fromSenderDB= msg.From;
   fromSenderDB = fromSenderDB.Remove(0,(fromSenderDB.IndexOf("\"") +
                  1));
   fromSenderDB = fromSenderDB.Remove(0,(fromSenderDB.IndexOf("\"") +
                  1));
   toReceiverDB= msg.To;
   toReceiverDB = toReceiverDB.Remove(0,(toReceiverDB.IndexOf("\"") +
                  2));
   toReceiverDB = toReceiverDB.Remove(0,(toReceiverDB.IndexOf("\"") +
                  2));
   subjectDB= msg.Subject;
   bodyDB= msg.TextBody;
   statusDB= "open";
   dateOpenDB= msg.SentOn;
   if (msg.Subject.IndexOf("track:") > 0)
   {
   string url=
         "http://zukhanye.ict.ru.ac.za/RhoTraxWS_34/TrackerService.asmx";
   string helpdesk = "NetServHelpDesk";
   IssueTracker it = new IssueTracker(helpdesk,"g04k5161",url);
   try
      {
       string str= msg.Subject.Substring(0,3);
       p = it.GetAnyProblemById(Convert.ToInt32(str));
       string userID= p.userid;
       Note note= new Clientlib.Note(msg.TextBody , DateTime.Now,
       userID, 0);
       p.addNote(note);
      }
   catch (System.Exception )
     {
      if ( msg.To.IndexOf(people)<0 && msg.BCC.IndexOf(people) < 0 &&
                                  msg.CC.IndexOf(people) < 0)
         {
            problemID = makeNewIssue(subjectDB, bodyDB);
         }
      msg.TextBody= msg.TextBody +
                   "http://zukhanye.ict.ru.ac.za/RhoTraxWebClient_30/
                    WebForm1.aspx?problemID="+ problemID + "\n
                    http://zukhanye.ict.ru.ac.za/RhoTraxWebClient_30/
                    CloseProblem.aspx?problemID=" + problemID;
      msg.Subject= problemID +" "+ msg.Subject;
     }
```

```
        }
    }
}
```

Code segment 4.1: MTA script executed on email

After deciding on MailEnable mail server, it took a while to set it up and to make it work. We estimated that it would take two weeks to get to know the mail server as well as its features. Because we had read its documentation before installing it, we were aware that it had a feature that allowed access to an email after it had been sent but before it was received. Getting this feature to work was also incorporated in the two-week estimate. This spike took much longer than we had anticipated and went well past the estimate: it took four weeks to make the quick prototype. This made us cautious about our estimates because even though it was just a quick prototype, it did need a lot more time than anticipated.

4.4. Assembling everything

After we had written spike solutions for parts of the system that seemed difficult, the next step was to connect them in a simple way. To do this we created a small database with minimal information just to meet our goal. The database was used to store the details contained in the email, that is subject, body, sender, receiver and the date. We used the MTA from the MailEnable mail server to intercept emails as they were sent. Emails were sent from one account to another using the mail server, and were intercepted using the MTA. When they were intercepted, and tagged for tracking, information from the email was recorded into a database table. The email had two hyperlinks added to the end of its body and was then delivered to the receiver, as in Figure 4.3 below. One link could be clicked when the task delegated via the email had been completed, the other could be clicked if there was a delay in accomplishing the task so as to record the delay and set a reminder. When these two functions were operational, the refactoring step required minimal cleaning up of the code of the MTA and the code to record the email contents.

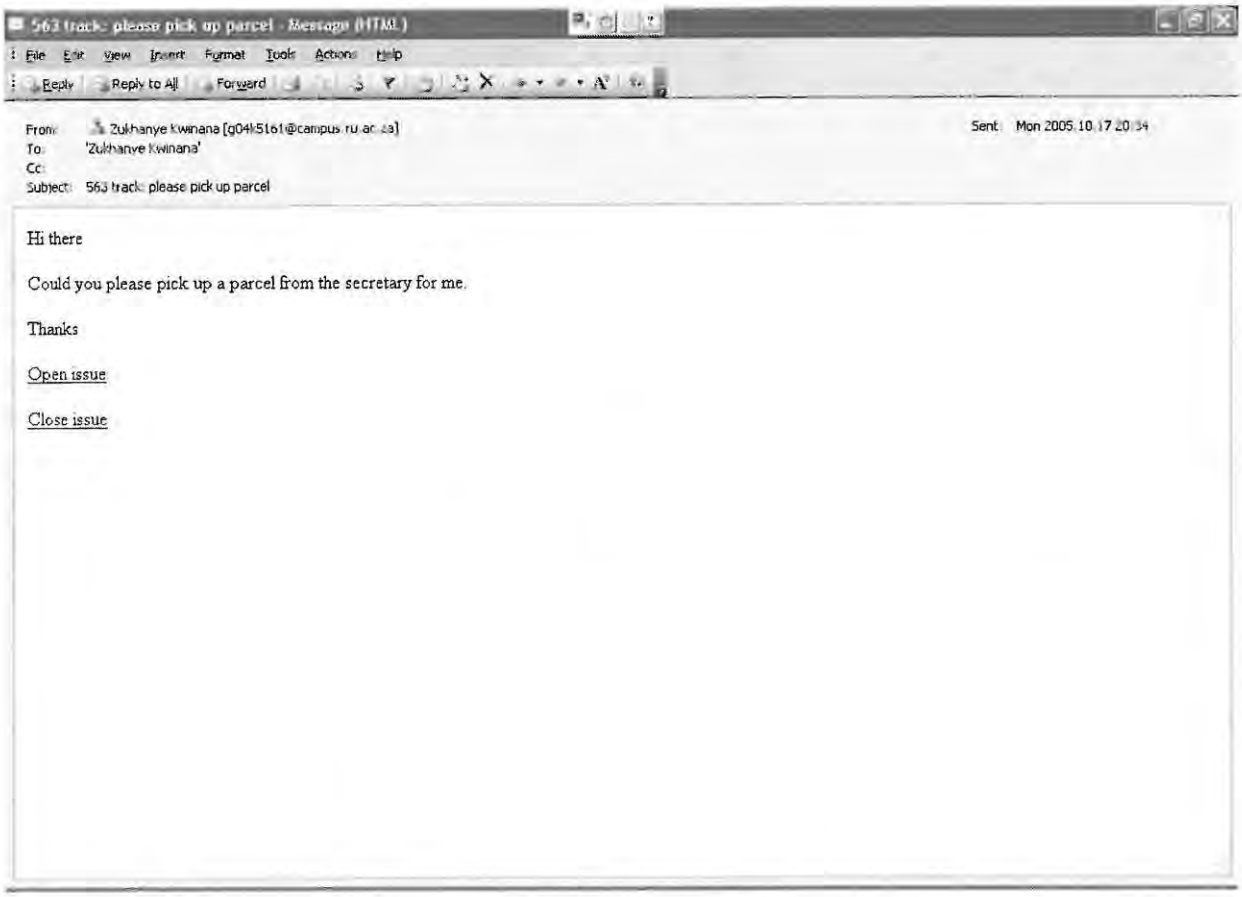


Figure 4.3: Tracked email with two hyperlinks added at the bottom

The hyperlinks were both appended with a unique number that is associated with the particular issue, so clicking either link allowed the receiver to work with the relevant issue. The received email had its subject appended with a unique number that corresponded to the particular issue it was related to. If the email did not contain all the information necessary to carry out the task, the receiver could reply to the email and a note would be added to the details of the email in the database and a new issue was not opened.

After the small database was developed, a web form was developed. The web form opened up when the second hyperlink was clicked. The web form, Figure 4.4 below, contains information about the issue, identified by the unique number appended to the hyperlink. The web form has a text input area so that the details about the delay can be recorded and a reminder set. When the submit button on the web form is pressed, the new information about the issue is recorded for future reference.

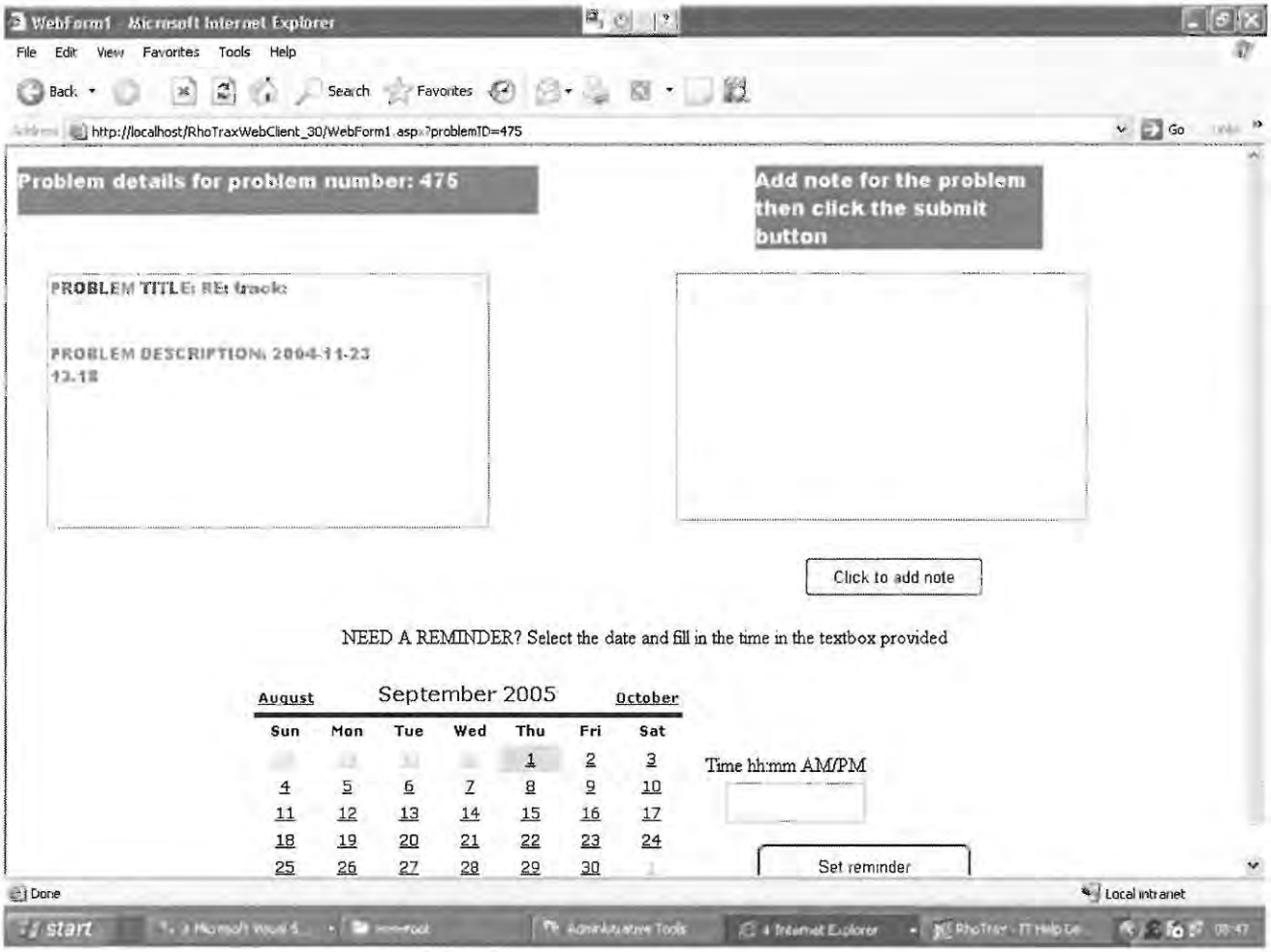


Figure 4.4: Web form used to add note about an issue and also set a reminder

We had estimated that getting the web form to work was going to take two days and it turned out that that was a reasonable estimate.

With all these different components working together, we had a small system that had a little of everything working, even though the components were going to be improved and some even discarded later. In that regard, we decided to make use of RhoTrax's database, and discard of our small experimental database, to continue with during our development. At this point it all looked like our email-based issue tracking system was coming together. It is worth noting at this point that although we are using web-based forms to view details and add notes about the issue, the user's starting point is always the email item.

4.5. Summary

In this chapter we provided a discussion about how XP handles software development projects. We then introduced user stories that aided the first iteration of the system. After the user stories were selected, a quick prototype was developed to help route the development in the right direction. After the prototypes, which showed us how to track our issue using the mail server, everything was arranged together using a skeletal database, which was later exchanged for one that is more stable. Using this database and RhoTrax, at the end of this chapter we have a seen that has a few of the functionalities, but is not good enough to be released yet.

Chapter 5

Iteration 2: linear and enriched workflow tracking

5. Introduction

Now that we had the seed of the email-based issue tracker by intercepting emails at the mail server, we moved on to build a more stable issue tracker with more features. The next section delves into interception and tracking at the mail client side by looking into Microsoft Outlook's add-in technology. As stated in the Chapter 2 section 2.1.1, two of the three Cs of groupware, namely communication and coordination, were being addressed by the user stories. In this chapter, the user stories are going to present new functionality and accommodate collaboration, the third C in groupware. In the user stories to follow, the individuals involved are collaborating with each other to accomplish a task. The project is that of reserving a car, accommodation or both by sending email to the relevant recipient, one after the other in a set order. Within any department there are tasks that are executed quite frequently and as a result the sequence of the steps that are taken can be preset because they are well known – our Computer Science Department is no different. Frequently occurring tasks are the reservation of a car from a car-hiring company or reserving accommodation for a trip, and the proper execution of these tasks is important. Complications occur when someone begins making arrangements but forgets to follow them through or the reservation reference number is misplaced and the person who made the reservation is unavailable.

By extending the already existing email-based issue tracking system we started to automate the above-mentioned tasks and introduced the first elements of a workflow system. However, we were mindful of our major objective: to capitalise on the users' comfort in working inside their habitat. This was also a practical next step, because often tasks like car hiring are managed via email, from one organization to another. This project is not the first to recognise that integrating with businesses we work with is important, just as we noted in Chapter 3 section 3.2.5 about BizTalk. This additional functionality still keeps email users in their inboxes, their habitat, and still assists in getting things done. Additions to the web service and the web forms were made to accommodate the new functionality. Software used during this project is briefly discussed in the following section, and finally we discuss test-driven development.

5.1. Outlook and its development model and the add-in technology

The interception of an email at the mail server side was completed and we realised that it was not practical to make everyone who wants to use the email-based issue tracking system change their mail server configurations. It became necessary then to move the interception away from the mail server to the mail client. Due to the fact that Microsoft Outlook 2003 is the most used email client within our department we decided to investigate how we could extend its functionality. As already noted in Section 4.3, Microsoft has a history of exposing an application's functionality to external programs. They introduced a technology called COM add-ins in Office 2000. If, for example, your project requires spell-checking functionality, you can leverage the Object Model exposed from Microsoft Word. If you are building an application that requires the functionality supplied by Microsoft Outlook 2003, you can leverage the associated object model [Troelsen 2004]. With the use of the Outlook 2003 object model you can interact with:

- The Outlook Calendar
- The Outlook Contacts database
- Outlook Notes and Tasks
- The user interface of Outlook (Explorers, Inspectors, CommandBars etc)

Add-ins can be built using any COM-compliant programming tool. COM add-ins are compatible across all Microsoft Office products. A COM add-in is a compiled DLL that is registered so that Office knows how to load and communicate with it. By writing an add-in as a DLL, you get the speed of running code in-process with the host application. This can provide tremendous performance benefits [Rizzo 1999].

With the use of a COM add-in, it is possible to access an email as it is sent, by catching the *ItemSend* event (this occurs when an Item is sent, in our case a mail Item). Once this event is fired, the email can be retrieved, and checked if the email had been tagged for tracking. If it is then we record the issue into RhoTrax, add the hyperlinks to the bottom of the email and append a unique number (corresponding to the issue) to the subject of the email. A short snippet of the code below shows how we got to access the different parts of an email with the more obvious part of the code left out.

```
using Microsoft.Office.Interop.Outlook;

private static void applicationObject_ItemSend(object Item,
                                               ref bool Cancel)
{

   MailItem newMail = (MailItem) Item;
   Application myOwnerApp = newMail.Application;
   NameSpace myNS         = newMail.Session;
   string subject         = newMail.Subject;
   string body            = newMail.Body;
   if (newMail.Subject.IndexOf("track:") > 0)

      { … }
}
```

Code segment 5.1: ApplicationObject_ItemSend event thrown when item sent

Everything about mail client tracking is the same as mail server tracking except where and how the email is intercepted. This means that once the email is received at the other end and the user performs the task, he/she clicks the first link to close the issue. If on the other hand there is some reason for a delay in accomplishing the task, he/she will click on the second link and submit the details about the delay to the issue tracker as a note via the web form. The main advantage of using the MailTrax plug-in is that emails that are sent out as well as those that are received in the user's inbox are tracked if they are tagged.

It is important to note that from the user stories and the previous chapter we want to keep users in their comfortable habitats. We have acknowledged the fact that people "live" in their email inboxes and get work and leisure accomplished with the assistance of their email inboxes. The user stories (and thus the system) keep the user in their inbox when necessary – that is when they need to delegate the task, when they are informed of any development regarding it, when they need to be reminded about it and when they need to close it. It is a useful "assistant" to anyone who wants to accomplish a task because it provides all the necessary functionality that is sufficient for keeping track of tasks. We do not try to hide the issue tracker but ensure that any interaction with the system is instantiated from the habitat so creating an issue becomes creating an email. As a result completing an issue becomes clicking a link in the email, updating an issue becomes clicking on a link in the email to open an update page of the web client. Next we move to the user stories for the workflow feature of the email-based issue tracking system.

5.2. The user story

Following XP, we start with a user story:

- *Jack emails the secretary asking for a hired car for his trip to Cape Town next week. Jack gives all the necessary information, and tags the mail for tracking through a specific workflow process, in this case a car-hire workflow process. The secretary receives the email with two links added to the end of the body. She clicks a link and this opens up a web form. She fills in the car details as well as Jack's details. When she clicks the submit button, an email gets sent to the hiring company used by the department with all the information. Once the hiring company arranges the car for Jack they will reply both to Jack and the secretary. Jack will know that everything has been arranged and will know the reference number to quote when he collects the car. The secretary will know for administrative purposes all the relevant information.*

This user story is different from the one in the previous chapter because in Chapter 4 the story is about a simple task that does not have a sequence of steps that are preset (a necessity for workflow). This user story also makes use of an external organization and thus an extension to the user story in Chapter 4. A similar story was developed for reserving accommodation and Figure 5.1 below shows the flow chart of the idea in the user story above.

In the above user story the tag in the email subject is the text *CarHireWF*. When the email-based issue tracking system intercepts an email and finds the keyword for car hiring, the appropriate workflow process can be instantiated (as an issue in RhoTrax), and two links are added. The first link opens up a web form to be filled in with car-hiring information; this is because the hyperlink that is added is different to the hyperlink for a simple task that is not workflow. The second hyperlink will be used to close the issue once it has been completed. When the hiring company respond, whether it is a positive or negative response, the system will intercept the email (by determining the domain name of the incoming email as well as the issue number that appears on the subject of the email) and record it. The response will be added as a note to the issue and all the relevant people informed of the outcome. Figure 5.2 shows the finite state machine of the car hiring user story.

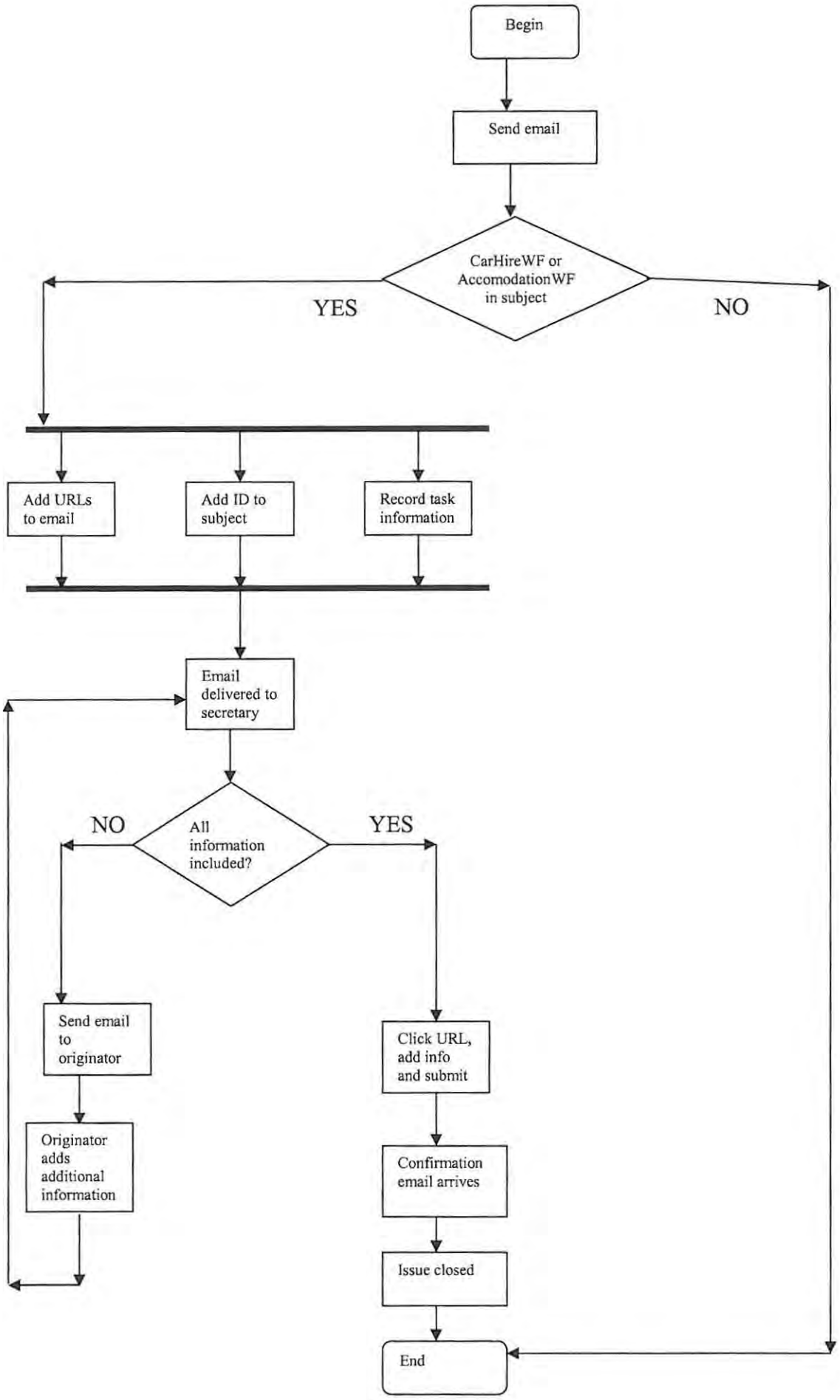


Figure 5.1: Flow chart for car hire or accommodation user story

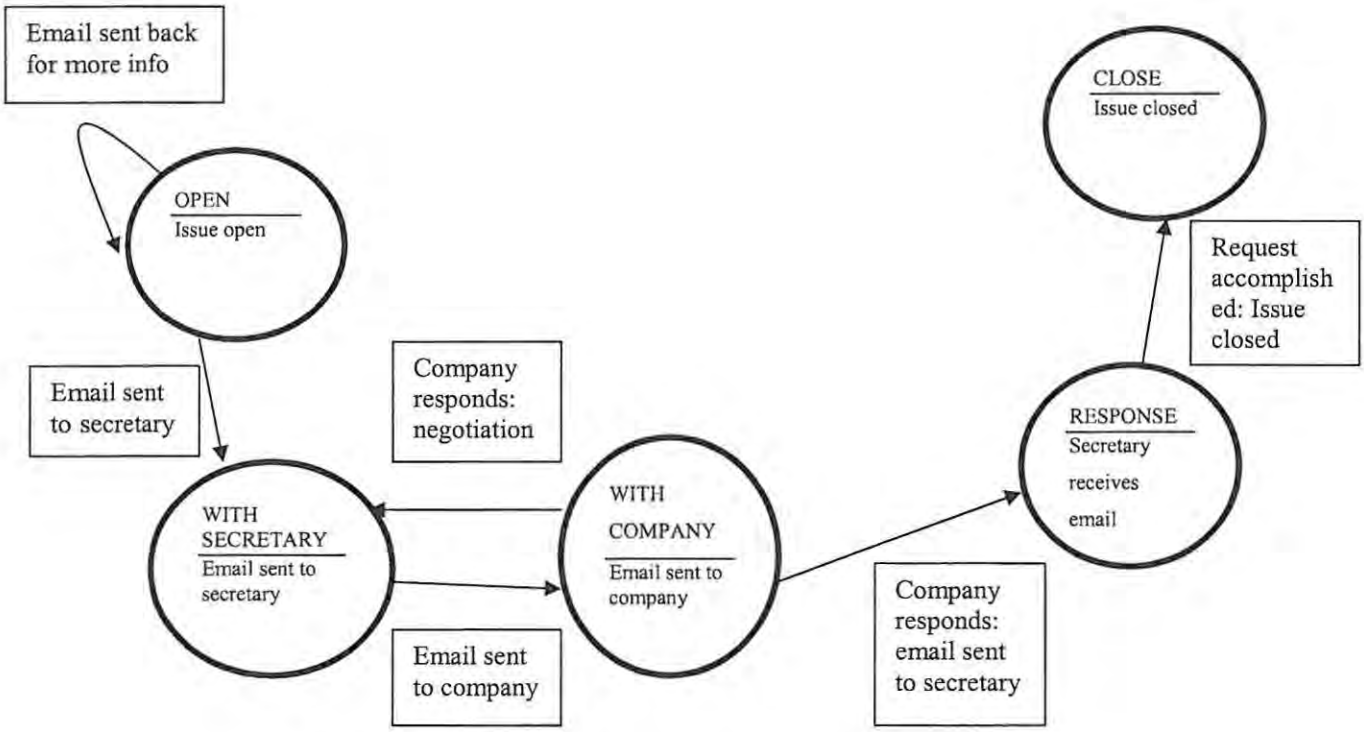


Figure 5.2: Finite state machine for the car hire or accommodation user story

To track hired-car and accommodation reservations we use RhoTrax and MailTrax. According to the user story above, when Jack sends an email to the secretary it will be tagged with the keyword *CarHireWF* (car hiring) or *AccommodationWF* (accommodation reservation). MailTrax will intercept the email and record its details into RhoTrax. The task is now in state OPEN. When the email arrives at the secretary it is in state WITH SECRETARY and it has a hyperlink appended to the body. The secretary can click the hyperlink to open up the web form, add car hiring details (or accommodation) and click the submit details. The task then goes into state WITH COMPANY. Figure 5.3 below shows the web form that the secretary will be filling in.

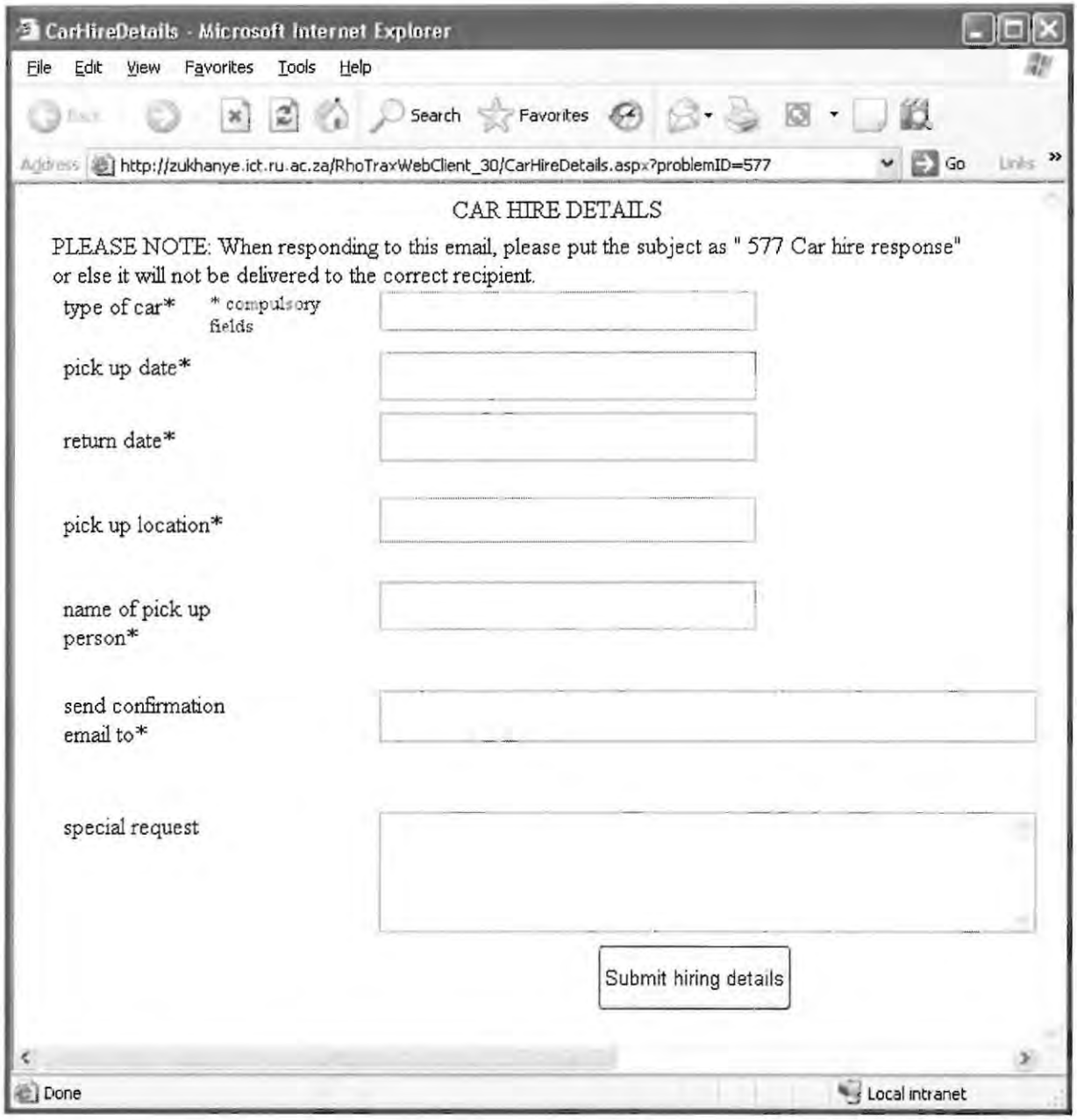


Figure 5.3: Web form for filling in car hire details

Once the details are submitted they will be sent to the supplier (car hiring company or accommodation establishment) that the department usually makes use of. The supplier will respond to the request via email (thus keeping everything in the inbox) and the issue will be in state RESPONSE. MailTrax will know when an email from the supplier arrives because of its domain name in the email address and because of the subject. When MailTrax identifies the email as a tracked issue, it will be intercepted, its details recorded into RhoTrax for future reference and a hyperlink added to it before it is released to the receiver. Now that the reservation is a success, the secretary can close the issue using the hyperlink in the email and

so the state of the issue will be CLOSE. The reason for the secretary being the one to close the issue is because our system cannot read and understand the contents of emails. If the company cannot provide the car that was the first choice, or the preferred type of accommodation, the secretary can ask for an alternative. A human eye is necessary to interpret the email and make a contribution if necessary. Also as an administrator she will want to know any such arrangements. The way that the system is arranged is such that it flows naturally, the user should not feel the difference in execution between ordinary car hiring and hiring using this issue tracker.

This iteration, with the main story being basic workflow, consisted of a few components. Time was taken to familiarise ourselves with Outlook and COM add-ins. We needed to write a small spike solution for a COM add-in to access an email as it was being sent. We had estimated that two days would be enough to learn COM add-ins and it turned out that our estimate was correct. After the two days and the newly acquired understanding for add-ins, we used the web service methods mentioned in Chapter 3 section 3.3 to record the information from the sent email and to save it to the database. After this, we developed a web form that is opened to fill in car or accommodation reservation details. This took about an hour to do and was a simple task.

Web browsers' ubiquitous nature supports the email-based tracking system as well as its extension into workflow. This is due to a large extent to the fact that with web browsers, the concern for deploying the application falls away. Users are able to use the functionality with minimum hassle. The uniformity, wide availability and simplicity of the interface makes a Web browser an ideal user interface and together with email, removes a lot of anxiety concerning the system [Miller, Sheth, Kochut and Palaniswami 1997].

5.2.1. Extensions

The user story above has been extended in a few directions. The originator can specify the account to be used when funding a hired car or accommodation request. This requires that we are able to specify choice in our workflow processes, and make the tracking engine capable of processing this choice. It is also possible for the originator to ask for both a hired car and accommodation to be arranged for a trip using the same email (the workflow language and the engine must then support parallelism). When there is uncertainty with regard to the issue, the

receiver of the email can respond to the sender asking for clarity. Tracking of this back and forth movement is recorded using the problem ID that is added to the subject of the email. This causes a loop to occur which can only be broken when the receiver has collected all the information necessary to perform a task and thus moves to the next part of the task. The issue tracker saves all these exchanges as notes that are related to the correct issue. These exchanges do not change the state of the issue.

The user story that addresses choice is similar to the car-hiring story above, the main difference is that the funding, and thus account details used by the secretary, comes from different sources. In our department the funds can be departmental or from other funds, such as research projects, depending on the nature of the request. If we continue with the user story about Jack above, when Jack emails the secretary he uses a particular keyword that associates his request to a particular account. This causes an email to be sent to the Head responsible for the account. The Head must first send an email that confirms that the request will be paid for, or else the secretary cannot make the necessary arrangements. When the Head has emailed the secretary with the confirmation, then she will make use of the hyperlink to open up the web form as per usual. She will fill in the web form and submit it and the rest follows as described above. Figure 5.4 below shows the finite state machine of the above user story.

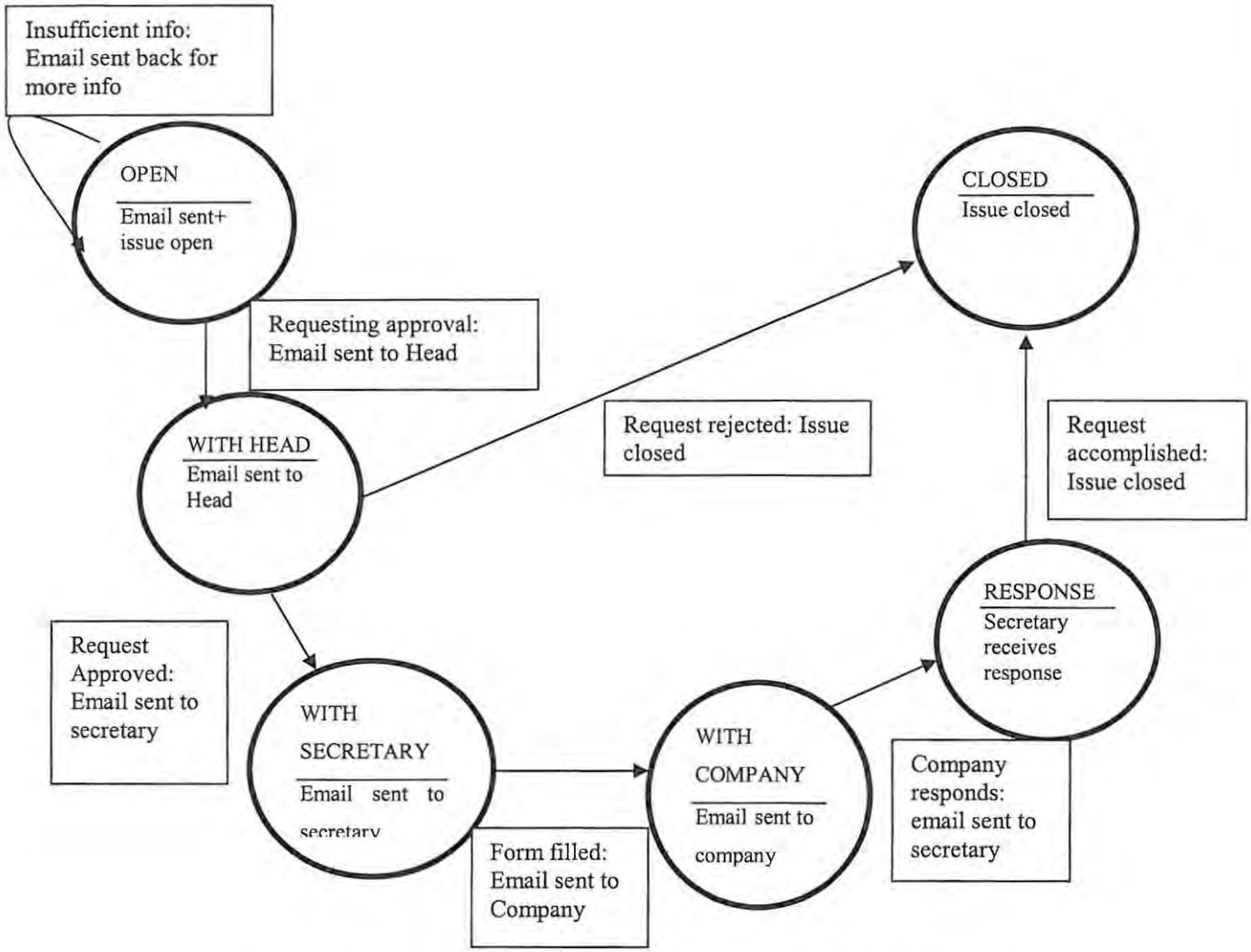


Figure 5.4: Finite state machine for user story for car-hiring or accommodation reservation

Figure 5.5 shows the flow chart of the user story that addresses choice. The user story that addresses parallelism is a combination of the car-hiring story together with arranging for accommodation. The story goes as follows:

- *Jack emails the secretary asking her to reserve accommodation and a car and gives her all the necessary information. When the secretary receives the email, she uses the web form to submit information.*

They both occur simultaneously, in parallel, in a similar way but in order for the workflow to be finished, the two parallel legs must be individually finished.

This part of the iteration went well as expected and did not take longer than anticipated to complete. The additions took almost three weeks to implement, together with changes to the database and web interface.

5.3. Web service and web interface additions

Changes to the database had to be made to accommodate the extension of the email-based issue tracking system. The basic architecture was there and few additions needed to be made. To the already existing problems table, three extra columns were added. Firstly, a workflow field, to record the type of workflow the issue belongs to; secondly, the last executed state of the workflow task; and thirdly, the due date of the workflow. Table 5.1, called the *tblStepsOfWorkflow* table, was added to contain details and steps of a workflow.

With the new table added, the web service's methods also increased in number and functionality to accommodate the workflow. With the added functionality and new tables to track a workflow issue, the workflow issue is submitted as per previous chapter using the `makeNewIssue` method. After the issue is submitted, the *problems* table is also updated to show the workflow details as well as the last executed step in the workflow.

tblStepsOfWorkflow		
Field	Data Type	Description
stepNumber	char	Position of particular step in whole workflow
workflowID	text	The ID of the workflow
stepDescription	text	Describes the particular step in question
sendToAction	text	The email address we will be sending information to
Next state	char	The "stepNumber" that follows the current one
dueDateWF	int	When the workflow needs to be completed

Table 5.1: Table containing workflow steps

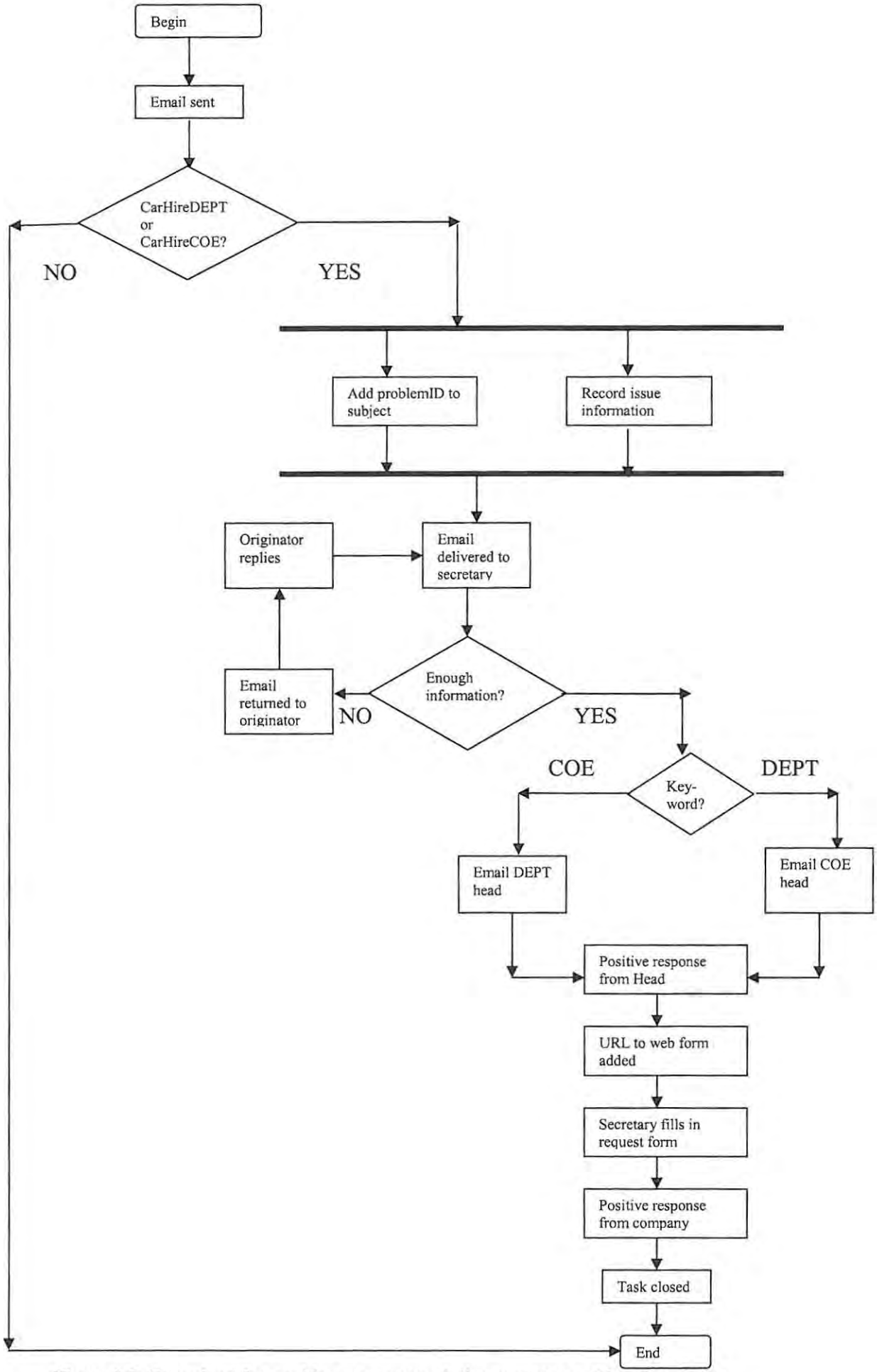


Figure 5.5: Flow chart for car hire or accommodation user story addressing choice

5.4. User roles in the system

Now that a prototype of the system exists, we define the different roles played by the users as seen in Figure 5.6. A general classification of the user-roles divides them into two groups: those inside and those outside our organisation. Those inside our organisation (*insiders*) can further be divided into those that are within our domain and those that are not. A further division of those within our domain occurs: primary users are the insiders in our domain that have installed the MailTrax add-in onto their Microsoft Outlook while secondary users are within our domain but do not have the add-in installed in their mail client. The primary users (i.e. those inside our domain with MailTrax installed) are the only ones that can instantiate new issues. They have direct access to RhoTrax, the web server and the tracking functionality, and the MailTrax add-in to create the close coupling with their email system. In principle, the secondary domain users have more rights than Friends or Outsiders, because they can use the RhoTrax rich front-end clients directly. But we have not exploited or considered this. They can also use the system (as can anybody else) by clicking on the hyperlinks in the tracked emails and either add a note or close the issue.

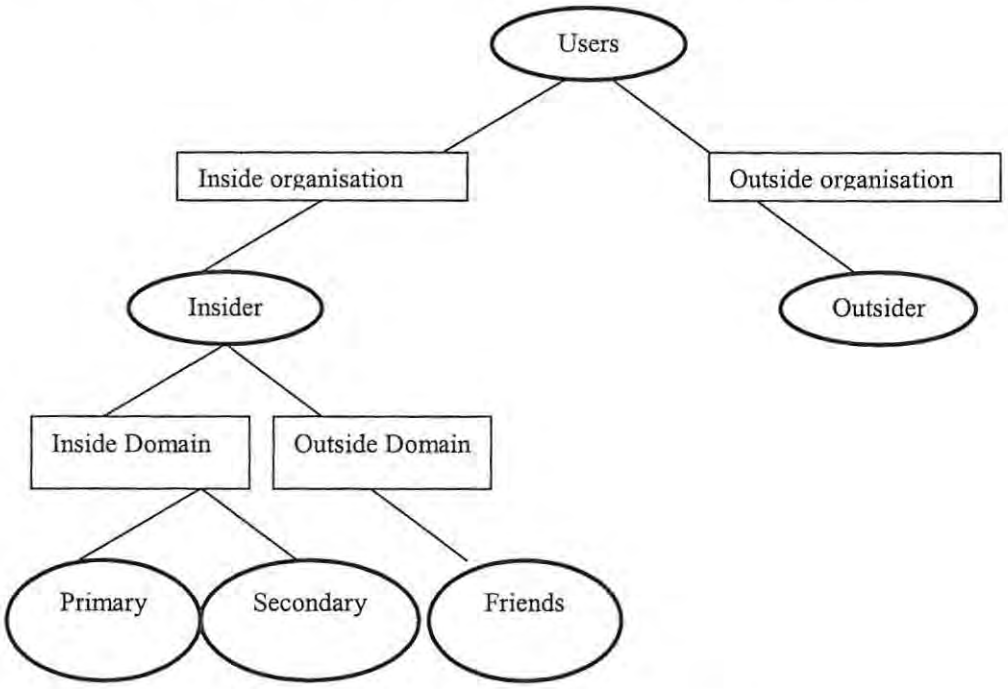


Figure 5.6: Tree diagram of the different roles

Outside our domain we have two classes of users, *friends* and *outsiders*. Both of these can be respondents to a tracked email. Friends are individuals within our organisation, just outside

our domain; outsiders are the individuals from outside our organisation. Friends interact with issues via a web-based interface or hyperlinks in the emails. The web server allows anonymous user access, and uses its own domain credentials to update the back-end issues on behalf of these respondents. Granting access to the outsiders means giving them limited services or the same access to the web server as used by the friends – i.e. they need to be able to use the web services and web pages we provide by exposing these services through our corporate firewall. Table 5.2 below illustrates the different roles and their possible actions.

DOMAIN BOUNDARIES	Inside Domain		Outside Domain	
CLASS OF USERS	Insiders	Insiders	Friends	Outsiders
TYPE OF USERS	Primary users (MailTrax add-in installed)	Secondary users (no add-in installed)	Tertiary users	
POSSIBLE ACTIONS	Instantiator and respondent	Respondent, but can also make use of other RhoTrax tools	Respondent, can use services via web, no need for firewall permission	Respondent, provided corporate policy allows access through the firewall

Table 5.2: Role of users

The system does not allow individuals from outside our department or organization to instantiate issues; this is because we want to keep track of our department's issues rather than those external to it.

When using the system, the insider can now send a tracked email to another insider, friend or outsider. The system is built such that when a primary user sends a tracked email, the email is intercepted and its details recorded (if it is a new issue) or the body of the email saved as a note in RhoTrax (if the issue already exists). When a primary user receives an email, the system checks whether the new email is a tracked workflow email that comes from either a

car-hiring or accommodation company. If it does, the system saves the body of the email as a note of the original issue.

Secondary and tertiary users (outside our domain and/or organization) can only respond to tracked email by either clicking on one of the hyperlinks or by replying as per normal to the email – just by clicking Microsoft Outlook's "reply" button. As above, when the primary user receives the reply to the tracked email, the system will intercept and record the email's body as a note on the system. The only way the tertiary users, more specifically the friends, can instantiate an issue in the system is by joining our domain and thus getting the necessary rights. Outsiders would similarly also need to be converted to insiders if they wanted the ability to instantiate issues. While this is not impossible it does have its drawbacks. The system is meant to keep track of our tasks and not those of other organisations.

If a primary user sends a tracked email to a secondary user, who in turn emails another secondary user before responding to the primary user, the emails between the two secondary users are not tracked. This means that the correspondence between the two secondary users is lost from the viewpoint of the system. The system can only keep track of email coming in or going out from the primary user's email client. If there is information in the lost correspondence that the primary user needs to know about, common sense forces the secondary to inform the primary user. When the primary user receives the response to the tracked email, our system will intercept the response (because of the problem ID in the subject as well as the keyword) and record the email as a note. As a result the information that is important to the primary user will not be lost due to the fact that secondary users do not have the MailTrax add-in installed into their email client.

If a primary user (P1) sends a tracked email to a secondary user (S1), who in turn sends an email to another primary user (P2), when the email arrives at P2 and when P2 responds to it, it will be tracked. This is because P2 has MailTrax installed, which identifies the email as tracked because of its subject and the problem ID. If however S1 were to change the subject of the email to not contain the keyword and the problem ID combination, then the email would not be tracked when it arrives or leaves P2.

5.5. Software used

This section discusses the software used during the process of developing our email-based issue tracking system.

All the code for the system is written in C# using Microsoft's Visual Studio 2003 which consists of the Microsoft Development Environment 2003 version 7.1.3088 as well as Microsoft's .NET Framework version 1.1.4322 service pack 1. Microsoft .NET Framework is the Microsoft Windows component for building and running various applications and XML Web services.

To keep versions of the code we employ the functionalities of Microsoft Visual SourceSafe version 6.0c. Visual SourceSafe is an ideal version control system for a small development team using Microsoft's Visual Studio .NET as it provides reliable code control [Microsoft 2005a].

NUnit version 2.2.0 is a unit-testing framework for all .NET languages [NUnit 2002]. It was used here to test the code that was written to make sure that it does what it was meant to do. This was important because we were using test-driven development during our implementation.

The database that is used is Microsoft SQL Server 2000 version 8. Microsoft SQL Server 2000 is a relational database management system that includes support for XML and HTTP.

The web server used is Microsoft Internet Information Services (IIS) Version 5.1. IIS is an integrated Internet service system that contains an HTTP server, an FTP server, an SMTP server and so on. It hosts and manages web sites and shares information across the Internet.

5.6. Test-driven development implementation

Test-driven development (TDD) goes hand in hand with XP and so we also made use of it. We believe that even though with strict TDD you should not write a single line of code without having a failing test, as you get more comfortable with XP obvious and extremely

simple tests can be omitted. This is to help save time for the more challenging tests and business codes. At the beginning of the project small, simple tests were written to help get us in step with TDD, and as the project progressed the simple tests started falling away.

We wrote tests for our business code, which is for the web service and RhoTrax. Below is an example of a test to see that after a new problem has been added, the number of problems increases by one.

```
[Test]
public void t01AddingIssue()
{
      int before = theTracker.GetNumProblems();
      theTracker.AddProblem("g04k5161", "test", "Struben Building", "000",
               "HelpDesk", "priority", "Computer Science", "NUnit Test",
               "This is an automated test");
      int after = theTracker.GetNumProblems();
      Assert.IsTrue(before+1 == after, "Failed to change the count after
                                  adding issue to IssueTracker");
      Assert.AreEqual(after , before +1 );
}
```

Code segment 5.2: t01AddingIssue Unit test

There are tests for things like whether the web service is contactable, whether we can get the different databases that are associated with RhoTrax and whether we can we update user details.

```
[Test]
public void t02canGetHelpDesks()
{
      string[] helpdesks = theProxy.getHelpDesks();
      Assert.IsTrue(helpdesks[0].Equals("NetServHelpDesk") &&
                        helpdesks[1].Equals("NetServTestDesk"));
}
```

Code segment 5.3: t02canGetHelpDesks Unit test

5.7. Summary

In this chapter we discussed the interception at the mail client side using Microsoft's add-in technology. The user stories for the additional functionality of enriched workflow were presented, after which the development that went into turning them into real features was delved into. Additions to the web service and web interface were also presented to support the

new functionality. The software used during this project was briefly presented. Finally test-driven development was discussed as it was used in the project.

Chapter 6

Conclusion

6. Introduction

In this concluding chapter we review our thesis. Our aim has been to implement an email-based issue tracking system that is extensible across organizational boundaries. Following the XP methodology, we have incrementally produced iterations towards the goal. We have discussed and justified the main decisions we have made and the impact and result of each decision, and outlined any possible extensions to the project.

6.1. Review and overall achievement

Ever since the term CSCW was coined, research on how technology could assist groups of people to work together even if they were spatially separate has grown. It has grown to include groupware, workflow management systems and workflow. Email is viewed to be the most successful and widespread of all groupware with people receiving and delegating tasks via email, within and outside their organization. Task management using emails is not a new concept and many researchers have attempted it and they have been successful in their way, just as our system is a success in its own way. As Ducheneaut and Bellotti [2001] noted, email users spend a lot of their time in their electronic habitats (inboxes), using their inboxes for more than just sending and receiving text messages. There have been many issue tracking systems that have been developed to help people manage tasks. The main problem with these systems is that they limit their functionality to people within their organization. In this day and age it is rare that an organization works in isolation, and more often than not it will need to delegate a task to another organization, or cope with work that spans boundaries.

Our work has explored combining the functionality of an issue tracking system with that of the almost ubiquitous email. Our aim was to demonstrate the feasibility of linking our familiar email habitat to an external issue-tracking system in a way that would promote managing our tasks, both inside and outside our organizational boundaries. We kept in mind the point by Terzis and Nixon [1999] – that users do not like to change the way they work, especially if they have already spent a significant amount of time and money in training – when we thought of this system. The system mostly caters for tasks that cannot be completed immediately, similar to that of Gwizdka [2001]. We can intercept the email either at the mail

server side or at the mail client depending on the user preference. The issue-tracker we made use of was a locally adapted version of the open source Liberum system. Our version, which provides a web-service API for enabling coupling to other software, is called RhoTrax. Our client-side plug-in for Outlook was produced from scratch, and is called MailTrax. The mail server we chose to demonstrate the server-side processing was MailEnable. Tracking at the mail server had the advantage that it was mail client independent, but it had drawbacks which meant that its users had to change their mail server configurations. We did however move away from mail server tracking because it forced people to change their email configurations and also meant that we could not track tasks that go across organisational boundaries, as it would mean the outsiders would also need to use our mail server. Mail client tracking was the more promising of the two places where interception occurs because it allowed crossing of organisational boundaries.

When the originating email is intercepted, it is checked for a keyword in the subject that indicates that it should be tracked. If it contains that keyword the information in the email instantiates a new issue in RhoTrax and two hyperlinks are appended to the body of the email and a unique issue number added to the email subject. The receiver can do two things when he/she receives the task delegating email:

- Attend to the issue immediately and successfully, and click the corresponding link to close the issue.
- Due to unforeseen circumstances the receiver might be unable to accomplish the issue immediately, thus he/she can click the corresponding link, set a reminder about when to get a reminding email and record the reason for the delay. When the reminding email arrives, the receiver can once again attempt to accomplish the task or add another note about further delays.

The above is the first functionality of our system and it works satisfactorily, keeping track of pending tasks and assisting individuals accomplish them.

The system was expanded in iterations, one user story at a time, to include a number of workflow features. The second main user story was a commonly occurring task within our department whose steps are known and thus can be automated – booking a car or accommodation. When an email with the request was sent to the secretary, its subject would

be tagged with particular text. When the email was intercepted and it was tagged with the keyword, then its details were recorded including what type of workflow the issue was. The email would have two links appended to its body: one opened up a web form for the secretary to fill in for reserving a car and submit the information, the other link closed the issue. When the form was filled in, the secretary submitted the information and an email was sent to the company to make the reservation. When the company responded, the email was intercepted and because of the unique number in the subject as well as the domain name in the email address, the email was recorded. When the secretary received the email with the response, she clicked the link to close the issue. These additions required expanding the functionality of RhoTrax – adding workflow elements to the data store, adding a primitive workflow engine and exposing some new web methods via the web service.

This user story was subsequently developed to include choice, parallelism and looping, typical control structures needed to express more complex workflows. Reserving a car or accommodation using specific funds meant using a particular keyword and depending on the keyword, the issue travelled a particular route and was approved by someone responsible for the funds – this denoted choice. Parallelism occurred when both car and accommodation reservations were done at once via the same email. This had its own unique keyword and meant two legs of the task had to occur in parallel with the task only ending when both the legs were completed. If a task was delegated but did not have all the information necessary to complete it, the receiver could send it back asking for clarity. This would create a loop that can only be broken when all the information necessary to complete the task was available. This required further enhancements to the workflow engine and the language for expressing the workflows.

Our analysis of the roles played by the various parties in tracked email exchanges led to the differentiation between two roles: the role of initiator of issues, and the role of respondents to an issue. There are two significant organizational boundaries. The first is the boundary created by our own domain of authentication. We called users on the domain the *insiders*. These users are recognized and authenticated on our own domain: their domain credentials gives them permission to access and use the back-end issue track system directly, potentially allowing different modes of access and use of the issue tracker. Some insiders become adopters of our system, that is, they install the necessary MailTrax add-in and they are thereafter known as Primary users. They are the only group who are able to instantiate new

issues for tracking. Those individuals within our domain, but who do not have the MailTrax add-in installed are called secondary users and cannot instantiate issues.

Outside our domain, but still within our organization's intranet, is the second class of users, which we call *friends*. We provide a web-based interface or clickable links in the email for them to interact with issues. The web server allows anonymous user access, and uses its own domain credentials to update the back-end issues on behalf of these respondents.

The third class of users is the *outsiders*: those outside our organization, who have to traverse our firewalls. Granting access to them requires that the web server is visible to outsiders. In principle, we could offer a constrained or limited service to outsiders, or we could give them access to the same web service as used by the friends group, or we could firewall their access entirely.

In our prototype system at this time we have only implemented two views of the system: one for the primary users who can initiate new tracked tasks, and one web-based anonymous system for everybody else: secondary users who have not adopted the system, friends outside our domain but inside the organization, and outsiders. For our experimental system we did not expose our web server to the outside world, so our current system is limited to use amongst insiders and friends. Figure 6.1 below is a simplified diagram of the different roles and where they exist within our domain.

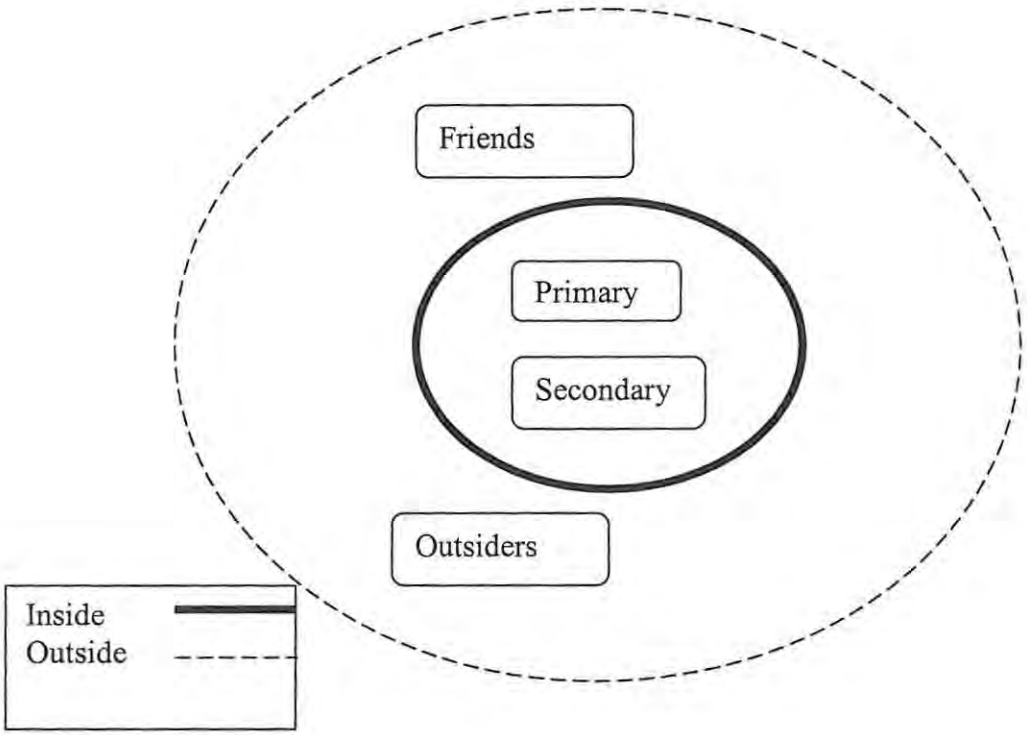


Figure 6.1: User roles inside and outside our domain

The methodology used during this project was eXtreme Programming (XP). XP was chosen because it is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements. Instead of forcing a software development project to make design decisions and adhere to them for the whole of the project, XP allows decisions to be made incrementally, and encourages revisiting or refactoring code as the requirements and demands grow. Using XP for the duration of this project has been a good decision and provided all the support a research developer needs to explore her ideas.

In the true spirit of XP we started off with a simple story, just to track an email from one person to another. This grew incrementally to accommodate workflow that allowed issue tracking inside and outside an organization. The user stories increased in difficulty and as a result made an impact in the design of the system. This also included keeping states of the issues as they moved from one person to another. The workflow itself was enriched to accommodate choice, parallelism and looping, as depicted in Table 6.1.

User stories	Design decisions
Requesting a simple task with no preset steps	Tracking
Tasks that occur regularly and their steps are preset	Linear workflow (car hire or accommodation)
Reservations using particular funds, reserving a car and accommodation or asking for clarity.	Enrich workflow (choice, parallelism, loop)

Table 6.1: User stories and their effect on the design decisions

The following section discusses the decisions we took during the course of the investigation, the impact they had and the result of each decision to the project.

6.2. Decisions made

- **XP methodology** – When this project started, the only definite thing we knew was that existing systems that track tasks have a few drawbacks, as mentioned in the introduction to Chapter 1 namely:
    - Individuals are isolated from the rest of the organization if they are not using the system.
    - Removing the user from a well-known environment and forcing them into an unknown interface that they need to learn encounters resistance.
    - Workflow and task tracking systems usually have a limited scope: using them is generally limited to within one's organization or department.
    - Email is just used to send emails by these systems but they do not play a central role for example delegating the task – that occurs at the system.

  With these drawbacks identified, we knew we wanted to develop a system that is free from them but we did not know how the system would work. That is the main reason why we chose XP because XP works when there are a lot of unknowns. We also used XP because with each user story and each iteration, the specifications of the system increased incrementally. This is evident because we started off with a simple user story that grew into linear workflow and then to enriched workflow. Even though we did not begin with a system design and architecture, we ended up with something that resembles it because of XP.

- **Client and server tracking** – The issue tracking functionality of our system had to be able to intercept and track emails within our organization as well as outside it. After searching around we made the decision that to accommodate tracking within our organization we could place the tracking functionality at the mail server. We could not expect other organizations to make use of our mail server, changing settings would have been annoying at best and there would have been too much traffic at the mail server. To accommodate tracking of issues in emails destined for outside our organization we placed the interception and tracking functionality at the mail client. These two decisions supported our aim of implementing an email-based issue tracking system that is extensible across organizational boundaries and gave us no problems during implementation.

- **Architecture (web services and applications)** – During implementation, Microsoft Visual Studio web services and web applications were used. The developer had previous experience with the above-mentioned technologies and had developed a liking for them and as a result there was minimal contest between which technologies to use. Also Microsoft has developed a reputation for themselves and their feature-rich products and documentation and as a result we were confident that the choices we made were going to deliver the proposed system. This decision impacted our system in such a way that we had a stable platform to build upon and this was an advantage.

- **Choice of software** – In Chapter 5 section 5.5, the software we used during this project was discussed. The choice to use Microsoft Visual Studio .NET and C# was made mainly because the author had experience in using the technology, as well as Microsoft being a well-established company with reliable products. The software provided all the functionality we were looking for ranging from COM add-ins to web services. There was always enough documentation available if there was a need for it. Visual SourceSafe was a logical step forward because we were already using Microsoft products. It also provided ideal versioning features and was stable. To accommodate XP practices we needed a testing framework to support the tests written during the project. NUnit is a unit-testing framework for all .NET languages and so an obvious choice. It was simple to use, and reliable as a result we never battled with it. We chose SQL Server database and the Microsoft Internet Information Services web

server for the same reason as the other software, that is, reliability and stability, and it can support the other software decisions made. All the software choices made were taken to support the bigger picture and they were useful decisions.

- **Off the shelf components (Liberum, MailEnable)** – As stated in Chapter 3, section 3.4, in this project we wanted to use already existing components and tie them together to provide the functionality we were looking for. We made this decision early on in the project and during the project we did not regret the decision. The main reason behind this is that we wanted to see if the components already developed and being used within our organization could sufficiently support our system. Building a system from scratch would take the attention away from the main system that allows interception and tracking of issues in emails.

- **Keywords in subject vs. menu** – As XP works in an incremental fashion so were our decisions as the implementation moved forward. At the beginning of the project, having the author of the email place a keyword in the subject of the email to invoke the tracking was the simplest thing that could work. As the project progressed, and keywords increased in number it became apparent that soon remembering all the keywords was going to turn into a task. At the end of this project the number of keywords to remember is still reasonable but it is a set-up that can be improved. As a result a feature that could be added is a menu list of all the keywords so that the author of the email chooses from it and does not have to remember the keywords off by heart. This can be a future extension to the system as it is not crucial in the system's functionality.

- **XP changed structure of thesis** – As stated in the introduction of Chapter 4, many software development projects are structured such that there is first an introduction, and then a literature review, followed by the design of the system and then the implementation, then comes the discussion and conclusion. For a while we tried to force this write-up to conform to the norm but the attempt was abandoned when it became obvious that we were trying to fit a square peg into a round hole. XP does not follow this norm. XP has user stories, then spikes (if necessary), then programming and then refactoring for each and every iteration. We found the XP methodology

influencing the structure of the write-up, and this in turn led to an unusual thesis in which the structure of the thesis in some sense mirrors the XP methodology.

6.3. Possible future extensions

Currently the hyperlinks that are added to the emails contain the unique identifier of each issue in RhoTrax. Each unique identifier is an incremental integer and as a result it is possible for malicious users to guess other hyperlinks by just changing that part of the hyperlink. In this regard an extension to this thesis can be replacing the unique identifier that is seen by the user with a Globally Unique Identifier (GUID). A way of doing this is by having a table that can link a particular GUID to a unique identifier of each problem. Doing this greatly improves the security of RhoTrax and the integrity of the information contained within it.

While we have also explored two ways of supporting primary users – via the MailTrax plug-in to an Outlook client, or via the MailEnable mail server, we have not yet explored the issues that may arise from using the two systems in tandem with each other.

The current agents (both shown in the appendices - MailTrax at the client side and the program that is executed by MailEnable at the server side) are currently hard-wired for a small number of workflow and tracking situations. We recognize that we would require a much more general system in the real world: that the list of available workflows would need to be determined solely in the RhoTrax system, so that an organization could add new workflows and keywords for recognition without needing to reinstall or change the interceptor agents.

6.4. Conclusions

The main thought behind this thesis was to see if we could improve task management by merging email and issue tracking systems. After our prototype was completed and the system had been used we have come to the following conclusions:

- The bringing together of these environments was a good concept and made task delegation and task management a bit more comfortable and efficient because it kept the users in their habitats yet provided issue tracking "intelligence".

- Because the system kept users in their habitat, delegating a task became sending an email, being reminded of a task became receiving an email, getting task-related information as well as closing the issue became clicking on a hyperlink.

Even though the prototype has limited features it successfully shows that we have a valid concept.

The system was developed using the eXtreme Programming methodology for software development. This was the first time the developer had used XP and familiarising herself with it took a while. Once the developer had become familiar with the methodology, it became an interesting experience and thus has the following to comment about:

- XP does not use new concepts; instead it uses practices that are as old as software development is but uses them "extremely". It follows then that to gain all the advantages of XP, you need to adhere to the practices.
- During the prototype development, religiously following the XP practices was difficult but we believe we remained true to the spirit of XP. As a result at the end of the thesis we believe that XP does have some merits as a software development methodology. Its highlight is the rule "do the simplest thing that could possibly work". This meant that we could worry about tomorrow's problems tomorrow. It allowed us the freedom to not try and think about the future and its many unknowns.
- Another practice that made life a bit easier is the unit tests. It is easy to tell which part of the code has been compromised by a change almost immediately instead of noticing the bug days or even weeks down the line when so much has changed in the code.
- As an individual developer, it was impossible to employ pair programming but following the other practices still proved fruitful.
- We extremely enjoyed the small releases/iterations because they gave us a sense of accomplishment. Also after each iteration, we could review the direction we were taking and if we felt it was misguided, we could change it. This worked hand-in-hand with the incremental design decisions that we took as we knew more about the prototype instead of making big ones when we knew very little of the future.

Overall the idea of email meets issue tracking developed to show merit. Using XP to explore merits behind the initial thought produced a rewarding prototype. This thesis can now, in the true sense of XP, be used as an iteration towards the development of an even better email-

based issue tracking system that is comfortable and can be used across organizational boundaries.

References

Adaptation Software — Adaptation Software, *Test-Driven Development: Way Fewer Bugs*, available from <http://www.adaptionsoft.com/tdd.html> last accessed 21 November 2005

Baecker, Grudin et al. 1995 — Baecker, R.M., Grudin, J., Buxton, W.A.S., and Greenberg, S. (Eds.) 1995, *Readings in Human-Computer Interaction: Towards the Year 2000*, 2nd Edition, Los Altos, CA: Morgan Kaufmann Publishers

Beck 2000 — Beck, K. 2000, e*Xtreme Programming explained*, Boston, MA: Addison-Wesley

Bellotti, Ducheneaut et al. 2000 — Bellotti, V., Ducheneaut, N., Howard, M., Neuwirth, C., and Smith, I. 2000, *Innovation in Extremis: Evolving an Application for The Critical Work of Email and Information Management*, ACM Conference on Designing Interactive Systems (DIS 2000), pp 181-192, available from <http://www.parc.com/research/publications/files/4364.pdf> last accessed 06 July 2005

Bellotti, Ducheneaut et al. 2003 — Bellotti, V., Ducheneaut, N., Howard M., and Smith I. 2003, *Taking email to task: the design and evaluation of a task management centred email tool*, ACM Conference on Human Factors in Computing Systems (CHI 2003), pp 345-352, available from <http://www.parc.com/research/publications/files/4678.pdf last accessed 21 November 2005

Bellotti, Dalal et al. 2004 — Bellotti, V., Dalal, B., Good, N., Flynn, P., and Bobrow, D.G. 2004, *What a To-Do: Studies of Task Management Towards the Design of a Personal Task List Manager*, ACM Conference on Human Factors in Computing Systems (CHI2004), pp 735-742, available from <http://www.parc.com/research/publications/files/5087.pdf> last accessed 21 March 2005

Bergman, Griss et al. 2002	Bergman, R., Griss, M., and Staelin C. 2002, *A Personal Email Assistant*, HP Labs 2002 Technical Reports, available from <http://www.hpl.hp.com/techreports/2002/HPL-2002-236.pdf > last accessed 21 March 2005
Caro, Guevara et al. 2000	Caro, J.L., Guevara, A., Aguayo, A., and Gálvez, S. 2000, *Workflow Management Applied to Information Systems in Tourism*, Journal of Travel Research, vol. 39, pp. 220-226
Christlinks 1994	Christlinks, *A glossary of computer and communications jargon*, 1994, available from <www.christlinks.com/glossary2.html> last accessed 18 April 2005
Coad 2002	Coad, P. 2002, *What is test-driven development?*, available from <http://bdn.borland.com/article/0,1410,29690,00.html> last accessed 18 April 2005
Dix, Finlay, Abowd and Beale 1998	Dix, A., Finlay, J., Abowd, G., Beale, R. 1998. *Human-Computer Interaction* (Second Edition), Upper Saddle River, NJ, USA: Prentice Hall
Document Management Avenue	Document Management Avenue, Document Management Avenue Glossary, available from <www.documentmanagement.org.uk/pages/glossary.htm> last accessed 18 April 2005
Dourish 1998	Dourish, P. 1998, *The State of Play*, Introduction for the special issue of the journal Computer Supported Cooperative Work on "Interaction and Collaboration in MUDs", vol. 7, pp. 1-2
Ducheneaut and Bellotti 2001	Ducheneaut, N. and Bellotti, V. 2001, *Email As A Habitat: An Exploration of Embedded Personal Information Management*, ACM Interactions, September-October, pp 30-38, ACM Press, available from

<http://www.parc.com/research/publications/files/4360.pdf> last accessed 21 March 2005

Ellis and Wainer 1999 — Ellis, C. and Wainer, J. 1999, *Groupware and computer supported cooperative work*, in Multiagent systems: a modern approach to distributed artificial intelligence, USA: MIT Press, pp. 425 – 457 available from <http://www.ic.unicamp.br/~wainer/papers/cscw-book.pdf> last accessed 21 November 2005

e-workflow 2005 — e-workflow, *What is workflow?*, available from <http://www.e-workflow.org> last accessed 04 December 2005

Grinter 2000 — Grinter, R.E. 2000, *Workflow systems: Occasions of success and failure*, Computer Supported Cooperative Work (CSCW), vol. 9, no. 2, pp. 189-214(26), Norwell, MA, USA: Kluwer Academic Publishers

Grudin 1994 — Grudin, J. 1994, *CSCW: History and Focus*, IEEE Computer, vol. 27, Issue 5, pp. 19-26

Grudin and Poltrock 1997 — Grudin, J. and Poltrock, S.E. 1997, *Computer-Supported Cooperative Work and Groupware*, in M. Zelkowitz (Ed.), Advances in Computers, vol. 45, pp. 269-320, Orlando: Academic Press, available at <http://www-users.itlabs.umn.edu/classes/Spring-2004/csci5980-terveen/grudin-and-poltrock-97-overview.pdf> last accessed 06 July 2005

Gwizdka 2001 — Gwizdka, J. 2001, *Supporting Prospective Information in Email*, Conference on Human Factors in Computing Systems, pp. 135-13, available from <http://portal.acm.org/citation.cfm?id=634150> last accessed 21 March 2005

Gwizdka 2002 — Gwizdka, J. 2002, *TaskView- Design and Evaluation of a Task-based Email Interface*, Proceedings of the 2002 conference of the Centre for

Advanced Studies on Collaborative research, available from http://delivery.acm.org/10.1145/790000/782119/p4-gwizdka.pdf?key1=782119&key2=3518854311&coll=GUIDE&dl=GUIDE&CFID=62988461&CFTOKEN=52553110 last accessed 21 November 2005

Gwizdka 2002a — Gwizdka J. 2002, *Future Time In Email – Design And Evaluation Of A Task-Based Email Interface,* available from <http://www.imedia.mie.utoronto.ca/people/jacek/pubs/2002/CASCON2002_JGwizdka_final.pdf> last accessed 09 March 2005

Hazemi, Hailes et al. 1996 — Hazemi, R., Hailes, S., Wilbur, S., Pitsika, M., and Wilbur, S. 1996, *UKERNA ACOL Project Report: Systems, Requirements and Products in Asynchronous Collaboration. Version 2.0*, The JNT Association, U.K. <http://www.ja.net/old/documents/archive/del1.html> last accessed 09 November 2004

Kwinana, Wentworth and Terzoli 2004 — Kwinana, Z.N., Wentworth, P., and Terzoli, A. 2004, *An email based issue- tracking workflow system that is extensible across organizational boundaries*, SATNAC 2004 (Southern African Telecommunications Networks and Applications Conference), September, Stellenbosch, South Africa.

Kwinana, Wentworth and Terzoli 2005 — Kwinana, Z.N., Wentworth, P., and Terzoli, A. 2005, *An email based issue- tracking workflow system that is extensible across organizational boundaries*, International Association for Development of the Information Society (IADIS) Virtual Multi Conference on Computer Science and Information Systems (MCCSIS 2005), April.

Lei and Singh 1997 — Lei, Y. and Singh, M.P. 1997, *A Comparison of Workflow Metamodels*, Proceedings of the ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling, available from

	<http://osm7.cs.byu.edu/ER97/workshop4/ls.html> last accessed 04 July 2005
Liberum 2000	Liberum, 2000, *Liberum Help Desk*, available from <http://www.liberum.org/> last accessed 21 November 2005
Liebert Coporation 2005	Liebert Coporation, 2005, *Network Terminology Glossary*, available from <www.liebert.com/support/glossary/net_gloss.asp> last accessed 09 March 2005
Linktionary 2001	Tom Sheldon's Linktionary, 2001, *Groupware* available from <http://www.linktionary.com/g/groupware.html> last accessed 09 March 2005
Lococo and Yen 1998	Lococo, A. and Yen, D.C. 1998, *Groupware: computer supported collaboration*, Telematics and Informatics, vol. 15, no. 1-2, pp. 85-101
Lyman and Varian	Lyman, P. and Varian, H.R. 2000, *How much information?*, available from <http://www.sims.berkeley.edu/how-much-info/> last accessed 30 March 2005
MailEnable 2004	MailEnable, *MailEnable – Evaluation Considerations*, available from <http://www.mailenable.com/benefits.asp> last accessed 06 July 2005
MailEnable 2004a	MailEnable, *What is a pick up event*, available from <http://www.mailenable.com/kb/Content/Article.asp?ID=me020028> last accessed 06 July 2005
MailEnable 2004b	MailEnable, *Platform requirements*, available from <http://www.mailenable.com/requirements.asp> last accessed 21 November 2005
Malone, Grant,	Malone, T.W., Grant, K.R., Lai, K.-Y., Rao, R. and Rosenblitt, D.A. 1989,

Lai, Rao and Rosenblitt 1989 — *The Information Lens: An intelligent system for information sharing and coordination*, in M.H. Olson (Ed.), Technological support for work group collaboration, pp. 65-88. Hillsdale, NJ: Lawrence Erlbaum

Microsoft 2001 — Microsoft, *Working with Microsoft Outlook objects*, available from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/modcore/html/deovrWorkingWithMicrosoftOutlookObjects.asp> last accessed 21 November 2005

Microsoft 2005 — Microsoft, *System Requirements for Exchange Server 2003*, available from <http://www.microsoft.com/exchange/evaluation/sysreqs/2003.mspx> last accessed 04 December 2005

Microsoft 2005a — Microsoft, *Microsoft Visual SourceSafe: Product Overview*, available from <http://msdn.microsoft.com/vstudio/previous/ssafe/productinfo/overview/ > last accessed 21 November 2005

Miller, Sheth, Kochut and Palaniswami 1997 — Miller, J.A., Sheth, A.P., Kochut, K.J. and Palaniswami, D. 1997, *The future of web-based workflow*, Proceedings of the International Workshop on Research Directions in Process Technology, Nancy, France, available from <http://citeseer.ist.psu.edu/cache/papers/cs/2019/http:zSzzSzra.cs.uga.edu zSzpublicationszSzWorkflowzSz85-proc-tech.pdf/miller97future.pdf > last accessed 10 December 2005

Ministry of Finance 2000 — Israeli Ministry of Finance: Information Systems. 2000, *Glossary of term*, available from <www.mof.gov.il/micun/gloss1.htm> last accessed 09 March 2005

Motiwalla and Nunamaker 1992 — Motiwalla, L.F. and Nunamaker, J.F., Jr. 1992. *MAIL-MAN: a knowledge-based mail assistant for managers,* Journal of Organizational Computing, vol. 2, pp. 131-54

Neurauter 2002 — Neurauter, T. 1999, *CSCW*, available from <http://www.neurauter.at/Diplomarbeit/html/node13.html> last accessed 09 March 2005

Newkirk and Vorontsov 2004 — Newkirk, J.W. and Vorontsov A.A. 2004, *Test-Driven Development in Microsoft .NET*, US: Microsoft Press

NUnit 2002 — NUnit, *What is NUnit?*, available from <http://www.nunit.org/> last accessed 21 November 2005

Olap Report 2005 — Pendse, N. 2005, *Glossary* available from <www.olapreport.com/glossary.htm> last accessed 09 March 2005

Phoenix 2005 — Phoenix Technology Group, 2005, *What is workflow?*, available from <http://www.phoenix.ie/workflow/whatis.html> last accessed 18 April 2005

Podgorny, Walczak et al. 1999 — Podgorny, M., Walczak, K., Warner, D., and Fox, G.C. 1999, *Internet groupware technologies – Past, Present, and Future trends*, available from <http://www.collabworx.com/legacy/tango/Documents/Papers/BIS/body_bis.html> last accessed 09 March 2005

Quixa Solutions 2005 — Quixa Solutions, *What is workflow?*, available from <http://www.quixa.com/workflow/what.asp> last accessed 04 December 2005

Rhett 2004 — Rhett, J. 2004. *Request v3: A Modular, Extensible Task Tracking Tool*, available from <http://support.isite.net/Projects/Request/lisa98paper.html> last accessed 06 July 2005

Rizzo 1999 — Rizzo, T. 1999, *COM Add-ins Part I: Introducing an Office 2000 Solution*

for the Entire (Office) Family, available from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dno2kta/html/msotrcom.asp> last accessed 06 July 2005

Spurgin 2003 — Spurgin, K.M. 2003, *Personal Information Management: A Review of the Literature*, available from <http://www.infomuse.net/papers/PIMLitReview.doc> last accessed 21 November 2005

Takkinen 2002 — Takkinen, J. 2002, *From Information Management to Task Management in Electronic Mail*, available from <http://www.ep.liu.se/diss/science_technology/07/32/digest.pdf> last accessed 06 July 2005

Terzis and Nixon 1999 — Terzis, S. and Nixon, P. 1999, *Building the next generation groupware: A survey of groupware and its impact on the virtual enterprise*, available from <http://www.cs.tcd.ie/publications/tech-reports/reports.99/TCD-CS-1999-08.pdf> last accessed 06 July 2005

Arch Hacker 2004 — The Arch Hacker, *BizTalk and SOA*, 2004, available from <http://thearchhacker.blogspot.com/2004/12/biztalk-and-soa.html> last accessed 21 November 2005

Troelsen 2004 — Troelsen, A.W. 2004, *An Introduction to Programming Outlook 2003 Using C#*, available from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/ol03csharp.asp> last accessed 06 July 2005

Venolia, Dabbish, Cadiz and Gupta 2001 — Venolia, G.D., Dabbish, L., Cadiz, J.J., and Gupta, A. 2001, *Supporting Email Workflow*, available from <http://research.microsoft.com/research/coet/Email/TRs/01-88.pdf> last accessed 09 March 2005

Wake 2000 — Wake, W.C., 2000, *Extreme Programming Explored*, Boston, MA: Addison-Wesley

Wells 1999 — Wells, D.1999, *The Rules and Practices of eXtreme Programming*, available from <http://www.extremeprogramming.org/rules.html> last accessed 21 November 2005

XProgramming 1999 — XProgramming, *XP Practices*, available from <http://www.xprogramming.com/images/circles.jpg> last accessed 21 November 2005

Zhao, Kumar et al. 2000 — Zhao, J.L., Kumar, A., and Stohr, E.A., 2000, *Workflow-centric Information Distribution through Email*, Journal of Management Information Systems, vol. 17, no. 3, pp.45-73, available from <http://citeseer.ist.psu.edu/cache/papers/cs/20359/http:zSzzSzshell.bpa.arizona.eduzSz~lzhaozSzJMIS00-ZKS.pdf/zhao00workflowcentric.pdf> last accessed 18 April 2005

Appendix

ItemSend method

This code listing is taken from the Microsoft Outlook MailTrax add-in. The `applicationObject_ItemSend` method is where the interception takes place each time an email is sent. We show how we scan the subject of the email and interact with the back-end RhoTrax system when we want tracking to take place.

```
private static void applicationObject_ItemSend(object Item, ref bool
                    Cancel)
{
carHireURL= "http://zukhanye.ict.ru.ac.za/RhoTraxWebClient_30/
            CarHireDetails.aspx?probl emID=";
issueOpenURL="http://zukhanye.ict.ru.ac.za/RhoTraxWebClient_30/WebForm1.as
            px?problemID=";
MailItem newMail= (MailItem) Item;
Microsoft.Office.Interop.Outlook.Application myOwnerApp =
            newMail.Application;
Microsoft.Office.Interop.Outlook.NameSpace myNS = newMail.Session;
subject= newMail.Subject;
body= newMail.Body;
if (newMail.Subject.IndexOf("track:") > 0 ||
            newMail.Subject.IndexOf("CarHireWF")>0 ||
            newMail.Subject.IndexOf("AccommodationWF") > 0 ||
            newMail.Subject.IndexOf("AccomodationCarHireWF") > 0 ||
            newMail.Subject.IndexOf("CarHireCOE") > 0 ||
            newMail.Subject.IndexOf("CarHireDEPT") > 0)
{
  it = new Clientlib.IssueTracker(helpdesk,"g04k5161",url);
  try
  {
      string str= newMail.Subject.Substring(0,3
      p = it.GetAnyProblemById(Convert.ToInt32(str));
      if (!p.status.Equals("Closed"))
      {
          string userID= p.userid;
          Note note= new Clientlib.Note(newMail.Body ,DateTime.Now, userID,
                                        0);
          p.addNote(note);
          if (newMail.Subject.IndexOf("CarHireDEPT") >0 ||
                  newMail.Subject.IndexOf("CarHireCOE")>0)
          {
           carHireURL="<a href="http://zukhanye.ict.ru.ac.za/
                    RhoTraxWebClient_30/CarHireDetails.aspx?
                    problemID="+problemID+"\"> Car Hire </a>";
           newMail.HTMLBody= newMail.HTMLBody + "\n " +carHireURL;
           it.sendEmailSimple(it.getITResponseFromHeadEmail(helpdesk,
                "CarHireDEPT"), adminEmail, "Please approve the following
                request","Could you kindly approve the request below
                \n\n" + p.description);
          }
      }
  }
```

```
catch (System.Exception )
{
  try
  {
     problemID = makeNewIssue(subject, body);
  }
  catch (System.Exception e)
  {
     MessageBox.Show(e.Message, e.StackTrace);
  }

newMail.Subject= problemID +" "+ newMail.Subject;
if       (newMail.Subject.IndexOf("CarHireWF")>0
         || newMail.Subject.IndexOf("AccommodationWF")>0
         ||newMail.Subject.IndexOf("AccomodationCarHireWF") > 0
         || newMail.Subject.IndexOf("CarHireCOE") > 0
         || newMail.Subject.IndexOf("CarHireDEPT") > 0)
{
    if   ( newMail.Subject.IndexOf("CarHireCOE") > 0
         || newMail.Subject.IndexOf("CarHireDEPT") > 0)
      {}
    else
     {
      newMail.HTMLBody= newMail.HTMLBody + "\n\n"+ "<a
                  href=\"http://zukhanye.ict.ru.ac.za/RhoTraxWebClient_30
                  /CarHireDetails.aspx?problemID="+problemID+"\"> Car
                  Hire </a>" + "\n\n"+ "<a href=\"http://
                  zukhanye.ict.ru.ac.za/RhoTraxWebClient_30/CloseProblem.
                  aspx?problemID="+problemID+"\"> Close Problem </a>";
     }
     if (newMail.Subject.IndexOf("CarHireWF") > 0)
     {
       int result= it.updateITProblemWFInfo(helpdesk,
                  Convert.ToInt16(problemID), "CarHireWF");
     }
     if (newMail.Subject.IndexOf("AccommodationWF") > 0)
     {
       int result= it.updateITProblemWFInfo(helpdesk,
                 Convert.ToInt16(problemID),"AccommodationWF");
     }
     if (newMail.Subject.IndexOf("AccomodationCarHireWF") > 0)
     {
       int result= it.updateITProblemWFInfo(helpdesk,
                 Convert.ToInt16(problemID), "AccomodationCarHireWF");
     }
    if (newMail.Subject.IndexOf("CarHireCOE") > 0)
     {
       int result= it.updateITProblemWFInfo(helpdesk,
                 Convert.ToInt16(problemID), "CarHireCOE");
     }
    if (newMail.Subject.IndexOf("CarHireDEPT") > 0)
     {
       int result= it.updateITProblemWFInfo(helpdesk,
                 Convert.ToInt16(problemID), "CarHireDEPT");
     }
    answer= it.updateITProblemState(helpdesk, Convert.ToInt16(problemID));
    int timing= it.lastITExecutedStateDuedate(helpdesk,
               Convert.ToInt16(problemID));
    stateTimer = new System.Timers.Timer();
    stateTimer.Elapsed+=new ElapsedEventHandler(stateTimer_Elapsed);
    stateTimer.Enabled= true;
```

```
    stateTimer.AutoReset= false;
    stateTimer.Interval= timing * 1000;
    Problem p = it.GetAnyProblemById(Convert.ToInt16(problemID));
    alltext= p.description;
    }

else
 {
   issueOpenURL= issueOpenURL+problemID;
   newMail.HTMLBody= newMail.HTMLBody + "<a
                     href=\"http://zukhanye.ict.ru.ac.za/
                     RhoTraxWebClient_30/WebForm1.aspx?problemID="
                     +problemID+"\"> Open Issue </a>"+ "\n\n\n ";
   newMail.HTMLBody= newMail.HTMLBody + "<a
                    href=\"http://zukhanye.ict.ru.ac.za/
                    RhoTraxWebClient_30/CloseProblem.aspx?problemID="
                    +problemID+"\"> Close Issue </a>";

  }
 }
}
```

Mail transfer agent code

This code listing is relevant to intercepting mails at the server, using the MailEnable MTA server. MailEnable will execute this program (with appropriate command line parameters) when it receives email. It provides the hook that allows us to recognize which emails need to be tracked, and to interact with the RhoTrax back-end appropriately.

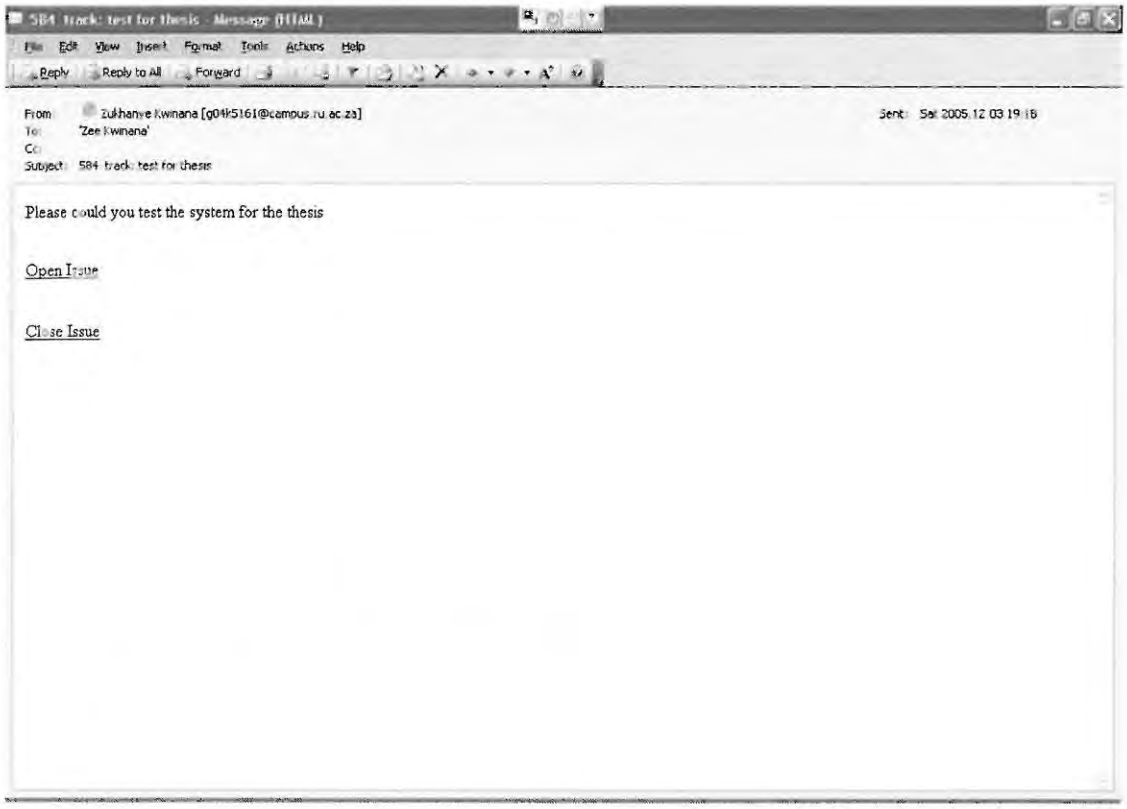
```
static void Main(string [] args)
{
if (args.Length < 2) return;
string MailFileArg = args[0];
string ConnectorArg = args[1];
RegistryKey reg = Registry.LocalMachine.OpenSubKey("SOFTWARE\\Mail
                  Enable\\Mail Enable");
string MailDataPath = (string) reg.GetValue("Data Directory");
string MessageFilePath= MailDataPath + "\\Queues\\" + ConnectorArg +
                  "\\Inbound\\Messages\\" + MailFileArg;
CDO.Message msg = LoadMessageFromFile(MessageFilePath);
string people= msg.To + msg.BCC + msg.CC;
if  (msg.Subject.IndexOf("track:") > 0  ||
    msg.Subject.IndexOf("Progress report:") >0  )
{
    fromSenderDB= msg.From;
    fromSenderDB = fromSenderDB.Remove(0,(fromSenderDB.IndexOf("\"") + 1));
    fromSenderDB = fromSenderDB.Remove(0,(fromSenderDB.IndexOf("\"") + 1));
    toReceiverDB= msg.To;
    toReceiverDB = toReceiverDB.Remove(0,(toReceiverDB.IndexOf("\"") + 2));
    toReceiverDB = toReceiverDB.Remove(0,(toReceiverDB.IndexOf("\"") + 2));
    subjectDB= msg.Subject;
    bodyDB= msg.TextBody;
```

```
    statusDB= "open";
    dateOpenDB= msg.SentOn;
    if (msg.Subject.IndexOf("track:") > 0)
    {
       string url= "http://zukhanye.ict.ru.ac.za/
                    RhoTraxWS_34/TrackerService.asmx";
       string helpdesk = "NetServHelpDesk";
       IssueTracker it = new IssueTracker(helpdesk,"g04k5161",url);
       try
       {
          string str= msg.Subject.Substring(0,3);
          p = it.GetAnyProblemById(Convert.ToInt32(str));
          string userID= p.userid;
          Note note= new Clientlib.Note(msg.TextBody , DateTime.Now,
          userID, 0);
          p.addNote(note);
       }
       catch (System.Exception )
       {
           if ( msg.To.IndexOf(people)<0 && msg.BCC.IndexOf(people) < 0 &&
                msg.CC.IndexOf(people) < 0)
               {
                  problemID = makeNewIssue(subjectDB, bodyDB);
               }
            msg.TextBody= msg.TextBody + " http://zukhanye.ict.ru.ac.za/
                          RhoTraxWebClient_30/WebForm1.aspx?problemID="
                          + problemID + "\n http://zukhanye.ict.ru.ac.za/
                          RhoTraxWebClient_30/CloseProblem.aspx?problemID="
                          + problemID;
            msg.Subject= problemID +" "+ msg.Subject;
       }
     }
  }
}


public static string makeNewIssue(string subject, string body)
{
  try
  {
      string url= "http://zukhanye.ict.ru.ac.za/RhoTraxWS_34/
                   TrackerService.asmx";
      string helpdesk = "NetServHelpDesk";
      IssueTracker it = new IssueTracker(helpdesk,"g04k5161",url);
      Problem p = it.AddProblem("g04k5161",g04k5161@campus.ru.ac.za,
                 "hamilton bulding","046 2222","EmailTracking","LOW",
                 "Computer Science", subject, body);
      return p.problemid.ToString();
   }
    catch (System.Exception d)
    {
       return d.Message;}
    }


public static CDO.Message LoadMessageFromFile (string path)
{
ADODB.Stream stm = new ADODB.StreamClass();
try
{
```

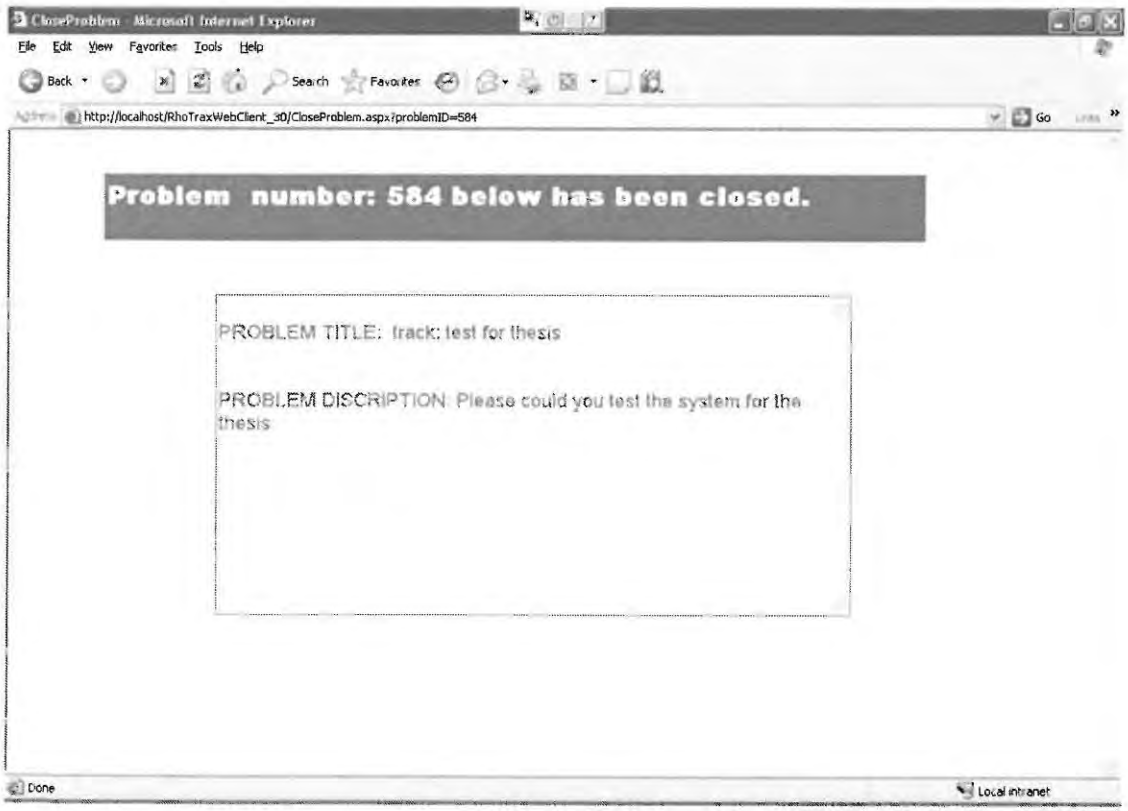
```
    stm.Open(System.Type.Missing,
            ADODB.ConnectModeEnum.adModeUnknown,
            ADODB.StreamOpenOptionsEnum.adOpenStreamUnspecified, "", "");
    stm.LoadFromFile(path);
    CDO.Message message = new CDO.MessageClass();
    CDO.IDataSource ds =  message.DataSource;
    ds.OpenObject(stm, "_Stream");
    return message;
}
catch (Exception ex)
{
}
return null;
}

private static void SaveMessageToFile( CDO.Message msg, string path)
{
ADODB.Stream stm = new ADODB.StreamClass();
stm.Open(System.Type.Missing,
         ADODB.ConnectModeEnum.adModeUnknown,
         ADODB.StreamOpenOptionsEnum.adOpenStreamUnspecified, "", "");
stm.Type= ADODB.StreamTypeEnum.adTypeText;
stm.Charset = "UTF-8" ;
CDO.IDataSource ds =  msg.DataSource;
ds.SaveToObject(stm, "_Stream");
stm.SaveToFile(path, ADODB.SaveOptionsEnum.adSaveCreateOverWrite);
}
```

Stepping through of the system

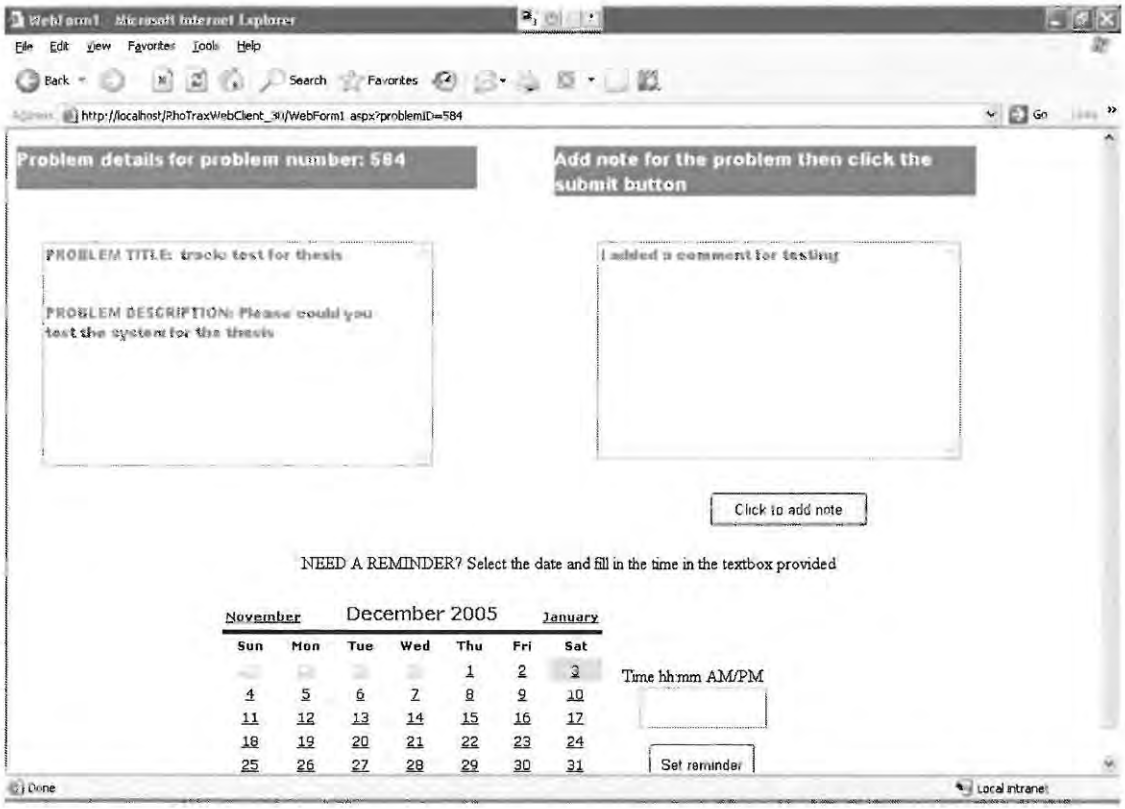
Step 1: Primary user sends tracked email. When the receiver (whether primary, non-primary, friend or outsider) receives the email, it has two hyperlinks added at the bottom (*Open Issue* and *Close Issue*).



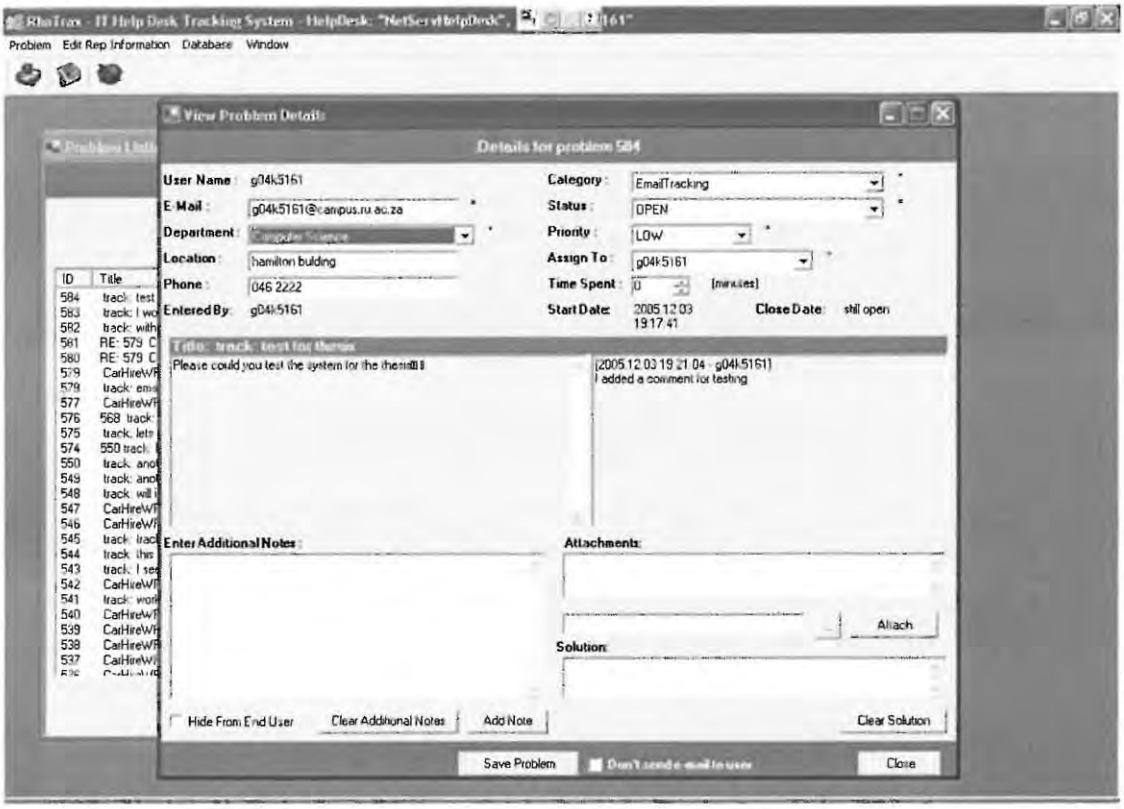
Close Issue is clicked when the requested task has been completed and the issue is closed. By clicking *Close Issue*, the web form below is opened.



If the requested task cannot be completed immediately, then the second link, *Open Issue* is clicked. This opens up the web form below. The receiver will then fill in the reason for the delay and submit it. This updates the issue in RhoTrax.



Below is a picture of RhoTrax with the new issue, and the comment that has just been added.



The steps for a workflow-tracked email are almost identical. The main difference (from the user's view) is the web form to fill in the details of the reservation. Below is a screen shot of the form.