CUSAT IT SUPPORT MANAGEMENT

IMPLEMENTING A HELP DESK FOR THE CUSAT WITH USE OF ITIL

PROJECT REPORT

INTERNSHIP AT COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY, INDIA

SEPTEMBER – DECEMBER 2000

DELFT, JANUARY 2001

Mark Dumay
Remco Groeneweg

Delft University of Technology
Faculty of Information Technology and Systems
Department Computer Science

Foreword

This is a report of the project performed during an internship at the Cochin University of Science and Technology, India. The internship is part of the curriculum of the master's program Computer Science, at the faculty of Information Technology and Systems, Delft University of Technology. It has been performed for a period of three months, during the months September until December 2000.

We would like to thank several people for their support and advice during our internship. First of all our two supervisors, David Peter and Wouter de Jong, where the former provided local support and guidance, and the latter from the Netherlands. Roy Kessler, who stayed for a period of three weeks in India, provided us with sound advice and feedback, and gave us much insight into organizational aspects. Arjo Rothuis, responsible for the financial aspects, and Bert Geers, supervisor of the INFOCUS project, without their support the internship would not have been possible at all.

During the project we encountered some specific problems related to the project. The following people gave us support, solutions, or advice. Hans Kompier, representative of the company Business Components International bv, gave us permission to use a database generation tool, which saved us a lot of work. Robert van der Kleij, who gave his advice and insight on subjects as caching and trees. Hans Geers, who gave a useful solution for storing and controlling specific tree structures. Rob Nievaart, who provided specific software.

Information and insight on a more global level, but nonetheless very helpful, were given by the following people. Reijer Boon, living in Cochin for a period of two years, who besides giving advice, has contributed to our pleasant stay in India. Eelco van Noot and Gerrit Versluis, consultants for the INFOCUS project, who contributed to a successful start of our project.

The following people largely contributed to our pleasant stay in India. The Americans who stayed in the guesthouse: Anna, Beth, Elizabeth, Gina, Jessie, Jon, Joy, Kayo, Matt, Nora, Shefali, and Susan, as well as their supervisors Tamula and Xioming. Our fellow student from Eindhoven, Martijn, and our fellow students from Cochin, Joji, Manesh, and Ranjith. And finally the consultants from the Netherlands, Freek and Marc.

We are very thankful to the people listed above, and all others we forgot, who enabled us to perform our project successfully, and contributed to our experience in India.

Mark Dumay and Remco Groeneweg
Delft, January 2001

Table of contents

Preface

This document describes a project to improve the services provided by the Information Resource Management (IRM) staff of the Cochin University of Science and Technology (CUSAT). It is part of a project called INFOCUS, which "… aims at an improvement of the institutional capacity of CUSAT in the areas of academic and administrative management as well as research and development through the upgrading of its data processing capacities and its information resource management"[1]. It is a cooperation project between the CUSAT and the Centre for International Cooperation and Appropriate Technology (CICAT), located in Delft, The Netherlands.

The Help Desk ITIL Support Tool (HIST), as this project is called, will be introduced in the first chapter. After this general background of the project, a description of the theory related to HIST will be given. The subjects Information Technology Infrastructure Library (ITIL) and object-oriented software engineering will be briefly described there. It is followed by a chapter on the requirements analysis, which will give details on the encountered situation and the proposed system. Next the system design will be discussed; it is based on the requirements obtained from the previous chapter. The chapter on object design gives insight into the design trade-offs that have been made, and the actual aspects of the system design that have been implemented. Implementation details are covered next, dealing with issues as user participation and training during the project. Finally, the conclusions and recommendations for the HIST project are given.

[1] See [08], Appendix 1, paragraph 1.

Management summary

The Helpdesk ITIL Support Tool (HIST) project aims to improve the services provided by the Information Resource Management (IRM) staff of the Cochin University of Science and Technology (CUSAT). It is part of a project called INFOCUS, which implements a complete information infrastructure for this university.

In order to meet with the goal stated above, the 1st line support of the existing help desk facility of the university is redesigned. First, a tool that logs all enquiries of the help desk is implemented. This results in a powerful management tool to enable the process of improving IRM services. Second, the support process itself now adheres to the IT Infrastructure Library (ITIL), a well-known standard in the field of information technology. It defines several different user roles, each with their own responsibility. Therefore, members of the IRM staff now have a clear task description.

A disconnected client/server architecture has been used to implement the system, with Microsoft SQL server as database management system and Java as the implementation language. A configuration management database has been created, extended with support for a help desk facility. The IRM staff can manage the complete system, but they are not capable of extending its functionality independently.

After the successful completion of the HIST project, it is recommended to redesign the 2nd and 3rd line support of the IRM. ITIL defines at least two processes, known as problem management and change management, that could support this. Combined with the results of the HIST project, this should result in an efficient and manageable service support for the university.

Abbreviations

API	Application Programming Interface
CI	Configuration Item
CICAT	Center for International Cooperation and Appropriate Technology
CMDB	Configuration Management Database
CUSAT	Cochin University of Science and Technology
DBMS	Database Management System
DUT	Delft University of Technology
EJB	Enterprise Java Beans
HD	Help Desk
HIST	Help Desk ITIL Support Tool
ICT	Information and Communication Technology
INFOCUS	Information Policy Development and Implementation CUSAT
IRM	Information Resource Management
IS	Information Systems
IT	Information Technology
ITIL	Information Technology Infrastructure Library
JDBC	Java Database Connectivity
JDK	Java Development Kit
MHO	Joint Financing Programme for Cooperation in Higher Education
ODBC	Open Database Connectivity
RFC	Request For Change
SQL	Structured Query Language
UML	Unified Modeling Language

1 Introduction

The activities described in this document are related to the INFOCUS project, which is supervised by Delft University of Technology (DUT). The aim of this project is to implement a complete information infrastructure for the University of Cochin (CUSAT). Besides the technical infrastructure itself, a general information policy is being developed as well in a Information Resource Management (IRM) sub-project. The information management is structured according to ITIL and to directives developed by the Delft University. The project is funded by the MHO and has a schedule of approximately four years and a total budget of Hfl. 4 million.

Primary focus of the HIST project will be to setup and implement a help desk (HD) tool for the university, supported by the IRM. Occasion of the project are the recommendations of a technical mission report, May 2000. It recommends the setup of a simple database system, which logs all inquiries from the IRM's users. This should result in a powerful management tool to enable the process of improving IRM services[2].

The HIST project is performed as an internship activity for two students of the Netherlands. It is part of the third year of the curriculum of the study Computer Science, as given on the Delft University of Technology, and represents a total of eight credits. The aim of this activity is to give students some hands-on experience with an ICT related project, with a main focus on practical software engineering.

[2] See [08], page 13, paragraph 4.

2 Theory

This chapter describes the theoretical aspects of the research performed for this project. It is separated into two paragraphs. The first paragraph, about ITIL, is the frame of reference for the entire project. The methodology used for the software engineering aspects is summarized in the second paragraph.

2.1 IT Infrastructure Library

This paragraph gives a general overview of the IT Infrastructure Library, which is the frame of reference for the HIST project. Preceded by a general introduction, the actual ITIL set of relevance will be described. The next subparagraphs will address the topics configuration management, problem management, change management, and help desk respectively. It is completed by a short description of the relationship between ITIL and general infrastructure management.

2.1.1 Introduction

The IT Infrastructure Library, commonly known as ITIL, is a set of about 40 books, and each addresses a specific subject on the field of IT management. It is developed by the Central Computer & Telecommunications Agency (CCTA), which supports the British government with its IT development. The methodologies they develop during that process are being published as what is known as ITIL.

The goal of ITIL is 'to facilitate improvements in efficiency and effectiveness in the provision of quality IT services and the management of the IT infrastructure within any organization'[3]. To realize this goal, several subjects on the field of IT management are discussed, which together should prove as a complete reference for the entire field of study. ITIL has a process-oriented view, and several related subjects are clustered in sets. In total 9 sets exist at this moment, grouped on four distinct levels:

- Long-term strategic goals;
- Medium-term strategic goals;
- Short-term operational goals;
- Environmental infrastructure management.

The relationship between set and level is not always clear, however. The first three levels are also known as 'IT service provision and IT infrastructure management', and the fourth as 'Environmental'.

[3] See [06], page 7, paragraph 1.

2.1.2 The service support set

As stated in paragraph 2.1.1, the complete ITIL set can be grouped on four distinct levels. The service support set belongs to the short-term operational goals, and is specifically related to the context of the HIST project. It addresses the following topics:

- Configuration management;
- Problem management;
- Change management;
- Help desk;
- Software control & distribution.

Of these topics, Software control & distribution has no specific relevance for the HIST project, so it will not be covered. The relationships between the processes are being shown in figure 2-1. CMDB stands for Configuration Management Database, and will be described in detail in paragraph 2.1.3.

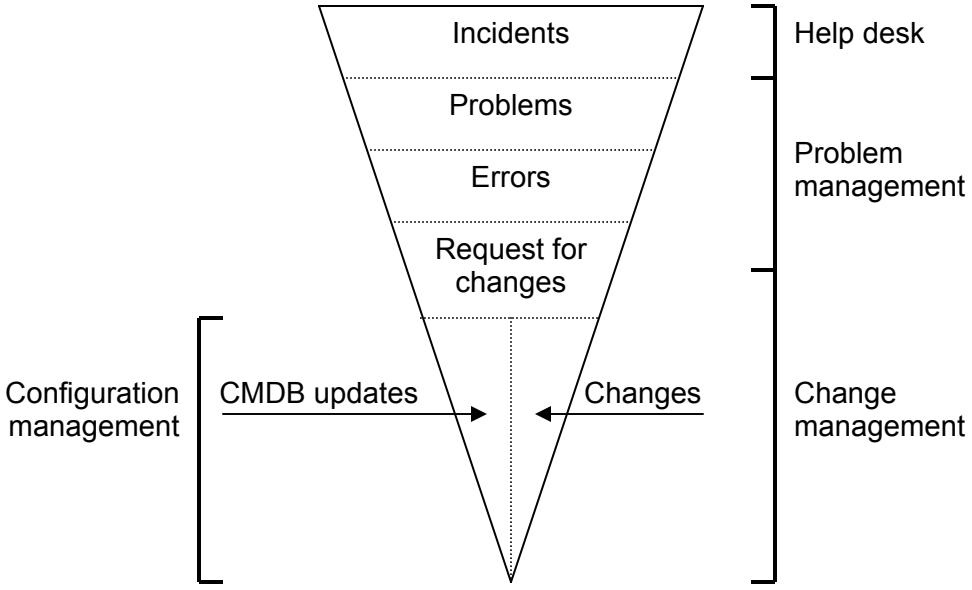

figure 2-1 Relationships between service support processes

2.1.3 Configuration management

This module gives guidance on all aspects of configuration management including planning implementation and running a configuration management function to control the components of 'live' systems.

Configuration management is a discipline, normally supported by software tools, that gives IT management precise control over IT assets by allowing IT management to:

- Specify versions of configuration items (CIs) in use and in existence on IT infrastructure and information on the status, owner, and relationships of these items;
- Maintain up-to-date records containing these pieces of information;
- Control changes to the CIs by ensuring changes are made only with the agreement of appropriate named authorities;
- Audit the IT infrastructure to ensure it contains the authorized CIs and only the CIs.

Configuration items include hardware devices, computer programs, documentation, telecommunication services, computer centre facilities and any others that the organization wishes to control.

The Configuration Management Database (CMDB) is the central database that stores all defined configuration items. Examples of its potential use are:

- To list all the component CI and their version number, within a packaged release;
- To identify the CIs affected by a scheduled (authorized) change;
- To list all requests for change (RFCs) relating to one particular CI;
- To identify all CIs purchased from a particular supplier within a specific period;
- To identify all equipment at a given location.

The CMDB is the central data storage for several processes contained in the service support set.

2.1.4 Problem management

Problem management is a disciple for handling all types of failures, called incidents in ITIL terminology. It aims not only to minimize the impact of failures when these occur, but also to correct the root cause of a failure. It consists of four major elements:

- Incident control, which restores normal services in case of a failure;
- Problem control, which tries to overtake the root cause of an incident;
- Error control, which corrects existing problems;
- Generate management information for the first three elements.

The process of problem management could be described as proactive, since its main object is to prevent the occurring of problems in the first place.

2.1.5 Change management

Change management can be described as a process that manages requests for changes (RFCs). These requests are usually initiated by the processes of HD (paragraph 2.1.6) and problem management (paragraph 2.1.4), but can also come from the user group. The processes that are part of change management are:

- Acceptation of RFC;
- Evaluation and planning;

- Processing of RFC;
- Coordination and acceptance.

Change management has relationships with the processes of service level management, problem management, configuration management, capacity management, network service management, and software control & distribution.

2.1.6 Help desk

The help desk offers support for end-users of IT services. It is responsible for the following activities:

- Incident control;
- User/IT division interface;
- Business operations support;
- Management information.

An important feature of the HD is that it depends on the use of software tools, particularly for incident control.

Since the HD is the primary user/IT division interface, it has a central role within the support service. Many connections with separate processes are possible. The relationships between the HD, problem management, and change management are shown in figure 2-2.

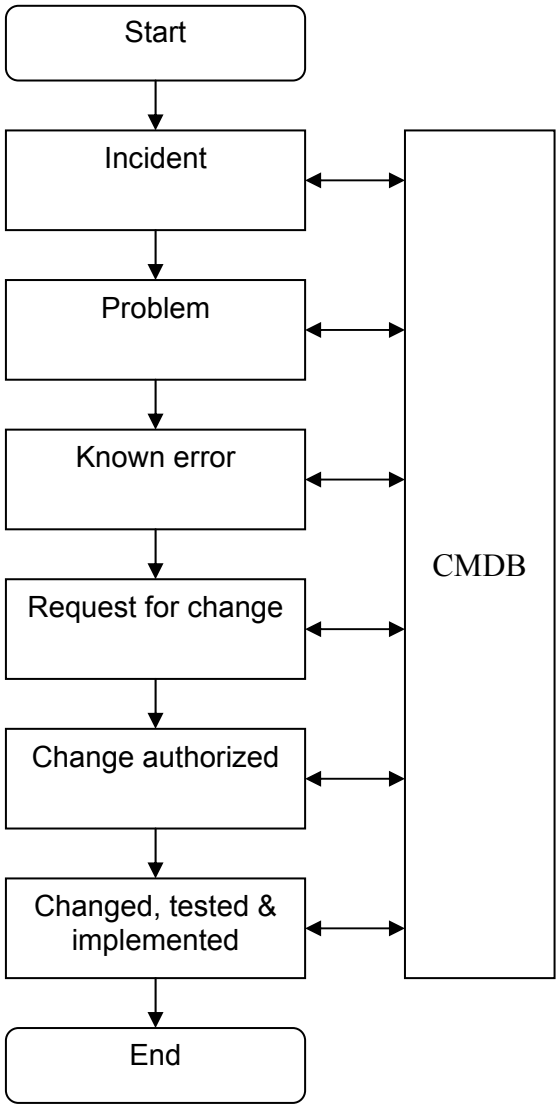

figure 2-2 Interfaces to problem and change management

2.1.7 Relationship with IT management

While ITIL is a rather comprehensive set of process descriptions, it is in general not considered to be a good manifest for a project. The lack of any implementation details largely contributes to this observation. See [18] and [21] for a more detailed description of this characteristic. In combination with the model of Looijen [19], the definition of tasks and responsibilities is covered better. This project will merely use ITIL as a context definition.

2.2 Object-oriented software engineering

This paragraph describes the approach that is used for the software engineering tasks of the HIST project. It is based on the methodology developed by Bruegge and Dutoit, and is covered in detail in [01]. First, the approach will be introduced and placed in context. It is followed by a description of activities that are related to a typical development process. Finally, the notation used in this project will be covered.

2.2.1 Introduction

The problem of building and delivering complex software systems on time is one of constant investigation and research. In response of the desolate state of many software systems, the term software engineering was introduced back in 1968. Engineering has the goal 'to build a high-quality product using off-the-shelf components and integrating them under time and budget constraints'[4].

The main cause of failing to deliver the right product at the right time is because of the complexity of many software systems. And to remain useful they need to evolve with the end-users' needs and the target environment. The technique described in this paragraph is object-oriented (also known as OO), which is mainly related to the actual development of such a system. Modern languages like Java are strongly object-oriented, but they have no real important impact on the (functional) design of software systems.

2.2.2 Development activities

A software-engineering project can be divided in several different activities. Complexity is dealt with by constructing models of the problem domain or system. Development activities include:

- Requirements elicitation;
- Analysis;
- System design;
- Object design;
- Implementation.

During the requirements elicitation, the client and developers define the purpose of the system. As a result the system is described in terms of actors and use cases, where actors represent external entities that interact with the system, and use cases describe the possible actions between an actor and the system for a given piece of functionality.

During analysis, developers aim to produce a model of the system. This model should be correct, complete, consistent, unambiguous, realistic, and verifiable. The use cases from

[4] See [01], page 3, paragraph 2.

the requirements elicitation are transformed into an object model that completely describes the system.

During system design, developers define the design goals of the project and decompose the system into smaller subsystems. They also select strategies for building the system, such as persistent data management and control flow. It results in a clear description of these strategies, a subsystem decomposition, and a deployment diagram representing the hardware-to-software mapping of the system.

During object design, developers define custom objects to bridge the gap between the analysis model and the platform defined during system design. This includes describing object and subsystem interfaces, selecting off-the-shelf components, and restructuring of the object model. This object model, along with constraints and description, is the result of object design.

During implementation, developers translate the object model into source code. It spans the gap between the detailed object design model and a complete set of source code files.

2.2.3 Unified Modeling Language

Throughout the software-engineering process, specific models and related materials make use of an annotation called Unified Modeling Language (UML). It allows a graphical representation of such models, in a consistent and clear way. This paragraph describes the various entities that are used in this document.

2.2.3.1 Use case diagrams

Use cases are used during requirements elicitation and analysis to represent the functionality of the system. They focus on the behavior of the system from an external point of view. A use case describes a function provided by the system that has a visible result for an actor. An actor describes any entity that interacts with the system, so it could e.g. be a user or another system. A sample use diagram is given in figure 2-3. The actor is called User, while the functionality is depicted by the oval with the label RequestForSupport.

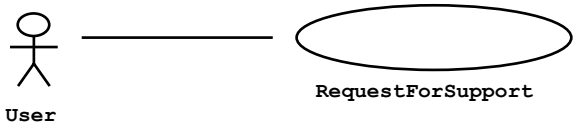

figure 2-3 Sample use case diagram

2.2.3.2 Class diagrams

Class diagrams are used to describe the structure of the system. Classes are abstractions that specify the common behavior of a set of objects. Class diagrams describe the system in terms of objects, classes, attributes, operations, and their associations. Numbers on the ends of associations denote the number of links an object can have with another (or itself). A sample class diagram is given in figure 2-4. The name of the class is ResponsibleOfficer, and it has three different attributes.

figure 2-4 Sample class diagram

3 Requirements analysis

This chapter describes the results of the requirements analysis performed for the HIST project. The first paragraph gives an introduction to the proposed system, where the purpose and scope of the project will be described. Next the current situation of the HD will be examined closely, followed by the proposed system. This final paragraph will contain the requirements of the system, as well as several descriptive models.

3.1 Introduction

This paragraph gives an introduction to the system to be built. First the purpose of the system will be explained, followed by its scope. Finally the objectives and success criteria of the project will be addressed.

3.1.1 Purpose of the system

The overall goal of the HIST project is to improve the first-line support services provided by the IRM, as well as the perception of its users. To meet with this objective, the existing helpdesk is to be restructured according to the guidelines of ITIL (see paragraph 2.1). To assist the IRM staff in their activities, an HD tool will be implemented.

The newly introduced methodology and tool should have almost immediate effect once they are implemented. Part of the tool is a configuration database, which contains all existing hard- and software configurations in use at the university. Inquiries can be dealt with more quickly. The logging of inquiries gives valuable management information, so the IRM staff can be managed more effectively as well. The configuration database is also provided to make the tool future-ready; it enables the implementation of the ITIL processes problem and change management.

Naturally, when the restructured HD is functioning well, the perception of its users should improve too. In addition, the HD should profile itself in the CUSAT community to increase the awareness of the services it is offering[5].

3.1.2 Scope of the system

The tool that will be implemented as part of the HIST project is integrated closely with the HD management itself. The primary defined task for the HD is 1st line IS support. ITIL defines several processes, each connected to the HD facility (see paragraph 2.1.2). In order to improve the provided services, a system that can log all inquiries received by the HD should be setup. This system relies on principles as described by incident

[5] See [08], Appendix 7, page 5, paragraph 3.

management and configuration management, which therefore define the scope of the system.

The processes that are started after first line support, such as problem management and change management, are beyond the scope of the HIST project. Once HIST is implemented successfully however, these specific processes should be examined closely as well.

3.1.3 Objectives and success criteria of the project

As stated in paragraph 3.1.1, the objective of the project is to improve the first-line support services provided by the IRM. The following indicators can measure the effectiveness of the HD[6]:

- Percentage of the incidents answered without having to refer to second or third line support or suppliers;
- Relation between the number of calls per location of user and the HD total calls;
- Average time needed to solve a problem, discriminating between actual time for problem solving and administrative, logistical time.

As a result of the implemented system, detailed management information will become available, allowing for better management control. This management improvement is hard to define, however.

The secondary objective of the project is to initiate a full ITIL implementation regarding service support. As stated in paragraph 3.1.2, the scope of the system is limited to incident support (also known as 1st line support). Once the HIST project has been implemented successfully, processes regarding problem management and change management should be started. To meet with this objective, the fundamental part of these processes, namely the CMDB, will be designed during the HIST project to cope with these future requirements.

3.2 Current situation

The current situation is given in [08], Appendix 7, page 5, paragraph 2. Below is a summary of that paragraph. Investigation learned that no significant changes to that situation have occurred between the time of that report and the present.

At the time of writing, an HD facility is in use, and is supported by two staff members of the IRM. They give first-line support, by answering telephone queries between 09.00 – 16.00 daily. Officially, this service started early January 2000. The HD keeps partial record of the questions it receives inside a logbook. Often the technical staff of the IRM

[6] See [08], Appendix 7, page 4, paragraph 3.

handles the hardware-related questions, they perform their services ‘in situ’ most of the time.

The HD is situated in the server room of the IRM center, where several services for the CUSAT network are provided. As such, the HD is also instrumental in trouble-shooting and problem solving with respect to these servers.

3.3 Proposed system

This paragraph addresses the proposed system, part of the HIST project. First an overview will be given, followed by a description of the system requirements. They are separated into three distinct sections, namely functional, non-functional, and pseudo requirements.

3.3.1 Overview

The future system will consist of two major subsystems: one subsystem covers configuration management, the other covers the help desk functions. The configuration management subsystem is built around a configuration management database (CMDB), see paragraph 2.1.3 for more details. Combined with an application front-end it gives the configuration manager a tool to administer configuration items. The help desk function depends on the CMDB to get information on configuration items for user support. The help desk tool supports the help desk employees on how to manage incidents, problems, as well as to provide a universal view for all employees on pending subjects. As additional functionality, a small web-based subsystem might be built, to provide clients with an interface for reporting incidents as well as to track the status of pending problems.

3.3.2 Functional requirements

Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any external system with which the system interacts.

The configuration management subsystem’s main functions include adding configuration item records, removing configuration item records and generating views on the CMDB. Change management is covered in the database in the form of change records. All functionality aims at providing a convenient and stable front-end to the CMDB.

The HD subsystem’s main function is to provide an interface for administering incidents and problems. All calls, emails, and other inquiries are logged and put in a database in the form of incident records. When an incident appears to be a problem, a member of the problem support group is notified. On his screen he can view all pending problems,

including the CI involved and a problem description. When the problem support group solves a problem, they close the problem record and notify the client.
To get full performance out of the help desk tool, it is depending on the CMDB to get up-to-date information regarding CIs. An additional view on the CMDB is created for the help desk employees.

As secondary functionality, a web-based facility that enables the reporting of problems is required, additional to the phone support given. The information sent here is redirected to the HD staff. The web facility should also provide the option to view the current status of any pending problems, to relieve the HD staff of unnecessary calls.

3.3.3 Nonfunctional requirements

Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. They include quantitative constraints, such as response time or accuracy.

3.3.3.1 User interface and human factors
Because the adoption of the application by the end users is a critical success factor, special effort must be made to create an easy-to-use interface. In order to measure the effectiveness of the user interface regular meetings with the end users will be held as soon as the prototype is ready. In these meetings the usefulness of the tool will be discussed.

3.3.3.2 Documentation
Various documents related to the project will be published. The most important one will be the project documentation, which describes processes as requirement elicitation and system design. On a more detailed level, documentation for source code is published as well. In order to enable the staff to manage the tool without being dependent on third parties, a management manual will re made too.

3.3.3.3 Performance characteristics
The tool will be programmed in the Java language. The object files of this programming language, so-called class files, run inside a Java Virtual Machine (JVM). Actually this is an interpreter, having some drawbacks on performance issues. As with the arrival of new Java Development Kits (JDKs), with the latest version 1.3, many of these performance problems have been overcome.

3.3.3.4 Error handling and disaster recovery
Error handling can be divided in the following subjects:
- Network communication problems;
- Database communication problems;
- Database errors.

Network communication and database communication are expected to be very reliable. In case one of the communication lines fails during operation, technical staff must be notified to make sure normal operation resumes as soon as possible.
Special attention must be paid to disaster recovery. Regular backups of the database should be made. This backup procedure may be highly automated with the use of scripts. With the application being very dependent on this database, it is very important that the database retains its integrity at all times.

3.3.3.5 System modifications

The overall system design will be documented in the project document. The system design will be annotated using UML, see paragraph 2.2.3 for more details. System modifications can be made after careful study of the system design model. Because the system is implemented with object-oriented techniques, system modifications may be made as long as the general architecture is respected.

3.3.3.6 Security issues

For each user an account is needed to log on the system. This account will be the same as the MS SQL Server 7 login that is stored in the master's database. Every user has a specific role, giving him/her adequate permissions to modify data in the database. This role also gives the user access to adequate views on the database as well as permission to use user's specific tools to work with.

3.3.4 Pseudo requirements

Pseudo requirements are requirements imposed by the client that restrict the implementation of the system. Typical pseudo requirements are the implementation language and the platform on which the system is to be implemented. They have usually no direct effect on the user's view of the system.

Five different constraints have been identified for the HIST project, they are shown in table 3-1. The platform and database are imposed by the client, the development language is a choice of the development team. The hardware is imposed by the current available infrastructure.

Description	Technology
Platform	Microsoft Windows NT 4
Language	Java
Database management system	Microsoft SQL Server 7
Server hardware	Pentium II, 128 MB RAM
Client hardware	Pentium, 32 MB RAM

table 3-1 Technical constraints

3.4 System models

This paragraph describes the models that were made during the requirements elicitation. First several scenarios will be described. Together they lead to the use case model, which is addressed in the next paragraph. It contains the model as well as the descriptions of each use case.

3.4.1 Scenarios

A scenario is a 'narrative description of what people do and experience as they try to make use of computer systems and applications' [02]. A scenario is a concrete, focused, informal description of a single feature of the system from the viewpoint of a single actor. It is merely used to give insight into the desired system, and is not meant as a complete representation of this system. The scenarios in this paragraph are ordered by the actor that initiates them, these are `User`, `ConfigurationManager`, and `DirectorOfIT`.

3.4.1.1 User initiated scenarios

Scenario name	**`ReportServerProblem`**

Participating actor instances	`Bob: User`
	`Liny: HelpDeskEmployee`
	`Reny: TechnicalEmployee`

Flow of events	1. Bob, typing his report, wants to save his document on the File Server in the main hall. The File Server however crashed and fails to save the document. The client computer returns a 'connection failed' error message. 2. Bob calls the Help Desk to report the problem. Liny answers the phone. First, she asks for the users' name, login name, and contact information. Then she asks for a brief description of the problem. This information is entered into HIST. Obviously, the call is an incident report. 3. Liny checks the validity of the incident report. HIST notifies Reny about the problem. 4. Reny is notified by HIST and solves the problem. A server reset puts it back into operation. Reny closes the incident record in HIST.

table 3-2 Scenario reportServerProblem

Scenario name	**`HelpNeededOnWord`**
Participating actor instances	`Bob: User` `Liny: HelpDeskEmployee`
Flow of events	1. Bob, typing his report, wants to know how to add

	tables in his document. He can't find it in the help documents either. 2. Bob calls the Help Desk to ask. Liny answers the phone. First, she asks for the users' name, login name, contact information. Then she asks for a brief description of the problem. This information is entered into HIST. Obviously the call is a request for help. 3. Liny explains Bob how he can put tables in his document using her own expertise on Word. 4. The phone call is ended.

table 3-3 Scenario helpNeededOnWord

Scenario name	**`WebmailLoginSupportNeeded`**
Participating actor instances	`Bob: User` `Liny: HelpDeskEmployee`
Flow of events	1. Bob wants to check his email using CUSAT's webmail. However his login/password combination is not accepted. 2. Bob calls the Help Desk to get support on his problem. Liny answers the phone. First, she asks for the users' name, login name, contact information. Then she asks for a brief description of the problem. This information is entered into HIST. Obviously the call is a request for support. 3. Liny validates that the person she's speaking with is actually Bob. Then she resets his password. Bob can get his new password at the helpdesk office. 4. The phone call is ended.

table 3-4 Scenario webmailLoginSupportNeeded

3.4.1.2 ConfigurationManager initiated scenarios

Scenario name	**`AddingChangeRecord`**
Participating actor instances	`John: ConfigurationManager` `Alice: TechnicalEmployee`
Flow of events	1. John, the `ConfigurationManager`, has ordered Alice to replace an old network hub in the Server room with a new state-of-the-art switcher. 2. Alice replaces the hub with the switcher in the Server room. 3. Alice adds a new change record in the CMDB containing a reference to the old CI. 4. Alice types in all the relevant information about the

	new CI, the switcher, and closes the change record. 5. HIST automatically updates the CMDB: the change record is stored in the Change DB, the status of the old CI is updated. 6. John validates the correct replacement and validates the new CI record.

table 3-5 Scenario addingChangeRecord

Scenario name	**`AddingNewCIRecord`**
Participating actor instances	`John: ConfigurationManager` `Alice: TechnicalEmployee`
Flow of events	1. John, the `ConfigurationManager`, has ordered Alice to install a network hub in the Exercise Room to connect all the computers with each other. 2. Alice installs the network hub in the Exercise Room. 3. Alice adds a new CI record in the CMDB. 4. Alice types in all the relevant information about the hub and closes the CI record. 5. John validates the correct installation of the network hub and validates the CI record.

table 3-6 Scenario addingNewCIRecord

3.4.1.3 DirectorOfIT initiated scenarios

Scenario name	**`GenerateIncidentReportStatistics`**
Participating actor instances	`David: DirectorOfIT`
Flow of events	1. David logs onto HIST. 2. In HIST he selects 'Incident Report Statistics'. 3. HIST generates the Incident Report Statisics. 4. David prints out the Statistics and logs off.

table 3-7 Scenario generateIncidentReportStatistics

3.4.2 Use case model

A scenario is an instance of a use case, that is, a use case specifies all possible scenarios for a given piece of functionality. An actor initiates a use case. After its initiation, a use case may interact with other actors as well. A use case represents a complete flow of events through the system in the sense that it describes a series of related interactions that result from the initiation of the use case.

The actual model is shown in figure 3-1, each separate use case is described in the tables following this figure. The only noticeable difference between this descriptions and the

model, is the abstraction of `TechnicalEmployee`. This role actually inherits all rights from `ConfigurationManager`, but it also has the right to report an incident.

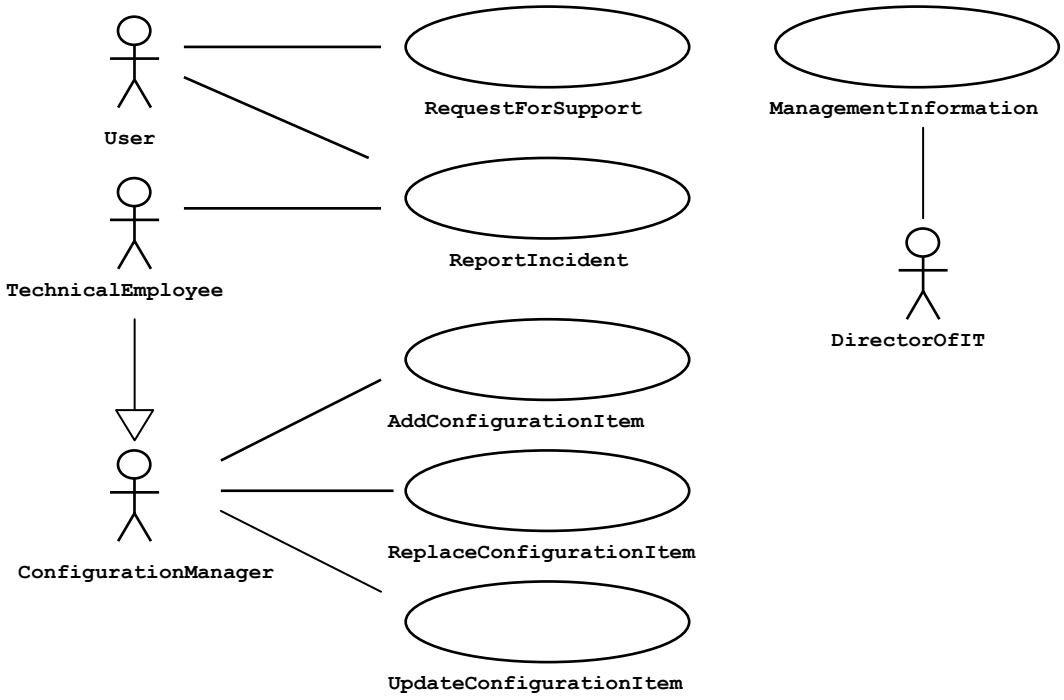

figure 3-1 Use case model of HIST

Use case name	**ReportIncident**
Participating actor	Initiated by `User` Communicates with `HelpDeskEmployee` Communicates with `TechnicalEmployee`
Entry condition	1. The `User` contacts the Help Desk either by phone or email to report an incident. This report consists of name, login name, contact information and a brief description of the problem.
Flow of events	2. The `HelpDeskEmployee` enters the report into HIST. 3. The `HelpDeskEmployee` checks if the incident is a problem. 4. The `TechnicalEmployee` is notified by HIST.
	5. The `TechnicalEmployee` solves the problem. He closes the Problem Report in HIST.
Special requirements	

table 3-8 Use case ReportIncident

Use case name	**RequestForSupport**
Participating actor	Initiated by `User`

	Communicates with `HelpDeskEmployee`
Entry condition	1. The `User` contacts the Help Desk either by phone or email to request for support. This request for support consists of name, login name, contact information and the question.
Flow of events	2. The `HelpDeskEmployee` enters the request for support into HIST. 3. The `HelpDeskEmployee` checks whether support is given on this subject.
Exit condition	4. The `HelpDeskEmployee` gives support.
Special requirements	

table 3-9 Use case RequestForSupport

Use case name	**`AddConfigurationItem`**
Participating actor	Initiated by `ConfigurationManager` Communicates with `TechnicalEmployee`
Entry condition	1. The `ConfigurationManager` orders the `TechnicalEmployee` to install a new Configuration Item (CI).
Flow of events	2. The `TechnicalEmployee` installs the new CI. 3. The `TechnicalEmployee` adds a new CI record into the CMDB using HIST.
Exit condition	4. The `ConfigurationManager` checks the correct installation of the CI and validates the CI record in the CMDB.
Special requirements	

table 3-10 Use case AddConfigurationItem

Use case name	**`ReplaceConfigurationItem`**
Participating actor	Initiated by `ConfigurationManager` Communicates with `TechnicalEmployee`
Entry condition	1. The `ConfigurationManager` orders the `TechnicalEmployee` to replace an existing CI.
Flow of events	2. The `TechnicalEmployee` replaces the CI. 3. The `TechnicalEmployee` adds a new change record into the CMDB using HISTs front-end.
Exit condition	4. The `ConfigurationManager` checks the correct installation of the CI and validates the CI record in the CMDB.
Special requirements	A change record consists of a pointer to the old CI record, a date, the UserID of the `TechnicalEmployee` and a pointer to the new CI record. The old CI record should not really be deleted, but

	instead placed in a ‘history‘ table.

table 3-11 Use case ReplaceConfigurationItem

Use case name	**`UpdateConfigurationItem`**
Participating actor	Initiated by `ConfigurationManager` Communicates with `TechnicalEmployee`
Entry condition	1. The `ConfigurationManager` orders the `TechnicalEmployee` to update a Configuration Item (CI).
Flow of events	2. The `TechnicalEmployee` updates the CI.
Exit condition	3. The `ConfigurationManager` validates the CI record in the CMDB.
Special requirements	

table 3-12 Use case DeleteConfigurationItem

Use case name	**`ManagementInformation`**
Participating actor	Initiated by `DirectorOfIT`
Entry condition	1. The `DirectorOfIT` requests management information.
Flow of events	2. HIST retrieves the requested information from the CMDB.
Exit condition	3. HIST returns all relevant management information.
Special requirements	This managementInformation can be incident report statistics, viewing the CMDB change log, etc.

table 3-13 Use case ManagementInformation

4 System design

System design is the first step for the description of the internal structure of a system. As the analysis model describes the system completely from the actor's point of view, the system design is the basis for actual system development. This chapter describes the results from the system design process, starting with the design goals in the first paragraph. This is followed by a elaboration on the software architecture implementation, which justifies the reason for custom building a HD tool. The chapter is closed by a description of the proposed software architecture, which will act as basis for the object design process part of the HIST project.

4.1 Design goals

Design goals identify the qualities a system should focus on. They are derived from the nonfunctional requirements (see paragraph 3.3.3) of the system. To guide the process of defining these design goals, several criteria are mentioned in [01]. They can be grouped in the following categories:

- Performance;
- Dependability;
- Cost;
- Maintenance;
- End-user.

The performance of the HIST tool is not extraordinary, in the present situation it will have to deal with at most five concurrent connections. The throughput of the system should be high enough to be able to handle ten concurrent connections, with a reasonable response time (several seconds) for each action.

The system does not have to meet with extreme conditions for robustness, reliability, availability, and fault tolerance, since it is a rather low-profile system, and not essential for the continuation of the HD process. In addition, because the system is only to be accessed by a local network, security is limited to the definition of several access roles.

Since the development time of the system is fixed, and all necessary infrastructure is already available, no extra costs for the system are accounted for. Staff training will be done during the project, with approximately a couple of hours allocated per week. Maintenance will be done jointly with already installed systems, and does not require specific cost assignments. Only in case of a required purchase of commercial tools, extra funds should be allocated.

The HIST project only addresses the 1st line support of the HD, but clearly should be able to deal with future enhancements and/or modifications (see paragraph 3.1.1). Adding or changing of the system's functionality should be made as easy as possible. This also implies that the readability of the system code should be as high as possible.

Since the existing HD is to be restructured according to ITIL (see paragraph 3.1.1), it is clearly important to make an effortless transition from the current situation to the new one. The delivered system should therefore focus to support the work of the user as good as possible, as well as to provide an easy to use interface.

The various trade-offs, which are derived from the criteria listed above, are given in table 4-1. These trade-offs are merely used to illustrate possible conflicting design goals, and what should be done if such a situation occurs.

Trade-off	Rationale
Extensibility vs. delivery time	To deal with future enhancements, the system should be easy to extend. When the project is behind schedule, however, focus should be on delivering a functional system and not an extendible system.
Modifiability vs. reliability	Since it is possible functional requirements of the system will change in the future, modifiability is an issue. This may not compensate the system's reliability, however.
Usability vs. delivery time	Although it is important to deliver a usable system, delivery time is fixed. The focus should be on meeting functional requirements first, when development is behind schedule.

table 4-1 Design goal trade-offs

4.2 Software architecture implementation

The initial idea for the implementation of a helpdesk, in combination with configuration management, was to use a commercial HD tool. Nowadays, many HD tools can be bought in a standard package. Most of these tools implement some or most ITIL guidelines on help desk, configuration management, problem management and change management (see paragraphs 2.1.3 to 2.1.6 for details about this topics). To help the decision-making process, demos of the tools 'Support Magic' and 'Heat' were examined. Additional information was found in [18] and [21]. This information was then compared with a technical mission report and its recommendations [08], together with communication from a Dutch consultant who has setup a similar HD tool in Dar-Es-Salaam. This communication can be found in Appendix 1.

The study on professional HD tools showed that almost all of them are mainly 'feature-driven'. This means that the basis of these programs is the amount of functions they support. That is somewhat the Achilles' heel of these tools: the great functionality makes them complicated and difficult to use.
In Dar-Es-Salaam, the IRM has experience with helpdesk tools. They first tried to implement the commercial support tool 'Heat'. This implementation completely failed, mainly due to the complexity of the program. Local aspects also played an important role.

For the implementation of ITIL guidelines strict procedures must be formed. When there aren't any procedures, e.g. a 'do as you please' atmosphere, these tools cannot function well. In the end, a simple helpdesk tool was built that only supported the main helpdesk functions. Conclusion of the consultant was that all commercial helpdesk tools are an overkill for relatively small IT infrastructures like in Dar-Es-Salaam and Cochin.

In consultation with the project supporters in Cochin and Delft, David Peter and Wouter de Jong, it has been decided to build a simple custom tool, fully adapted to local requirements, rather than to purchase a commercial package. Main argument was the uncertainty of the success of such an implementation.

4.3 Proposed software architecture

This paragraph describes the proposed software architecture, which is derived from the requirements elicitation (chapter 3) and the design goals (paragraph 4.1). First an overview is given, which briefly describes the chosen architecture and identified subsystems. Next the actual subsystem decomposition is described, with details on each subsystem's responsibilities and contained objects. The hardware/software mapping is addressed thereafter, which elaborates the various options for the proposed software architecture. Persistent data management focuses on the database design and how it is managed by a DBMS. It is followed by a paragraph on access control and security that describes the user model and security issues of the system. The paragraph is closed with a description of global software control, which addresses synchronization and concurrency issues.

4.3.1 Overview

The proposed system will be based on a client/server architecture, where the server contains a central data storage and the clients contain presentation and application logic subsystems. To obtain loosely coupled functional domains, six different subsystems have been identified. Communication with the database will be done by a single subsystem, configuration management, incident control, and user management are other major subsystems, responsible for the core of the application. They are used by two different user interfaces, of which at least one is web-based.

4.3.2 Subsystem decomposition

Finding subsystems is a volatile activity driven by heuristics. Subsystem decomposition is an iterative process. The initial subsystem decomposition should be derived from the functional requirements. Another heuristic for subsystem identification is to keep functionally related objects together.

Analysis of the use case model in paragraph 3.4.2 together with the functional and nonfunctional requirements of the system (paragraphs 3.3.2 and 3.3.3) identified the following subsystems:

- CMDBCommunication;
- ConfigurationManagement;
- IncidentControl;
- UserManagement;
- JFCUI;
- ServletUI.

Each of them is discussed in the following paragraphs in detail. The identification of these subsystems is not formally specified, and could easily provided different results for different system designers. In this case, the derived functional requirements are maintained within four different subsystems. The UserManagement and CMDBCommunication subsystems are merely technical design issues.
The first four subsystems make use of the Facade pattern, which '... reduces dependencies among classes by encapsulating a subsystem with a simple unified interface.'[7] The last two are user interfaces, which interact with the subsystems above.

4.3.2.1 CMDBCommunication

The `CMDBCommunication` subsystem is responsible for all communication with the CMDB. Through the `CMDBFacade` it provides all other subsystems with a safe and convenient interface for storage management. It incorporates facilities to cache data with the use of an `ObjectCache`. This should result in less network traffic and therefore better performance. The subsystem is pictured in figure 4-1.

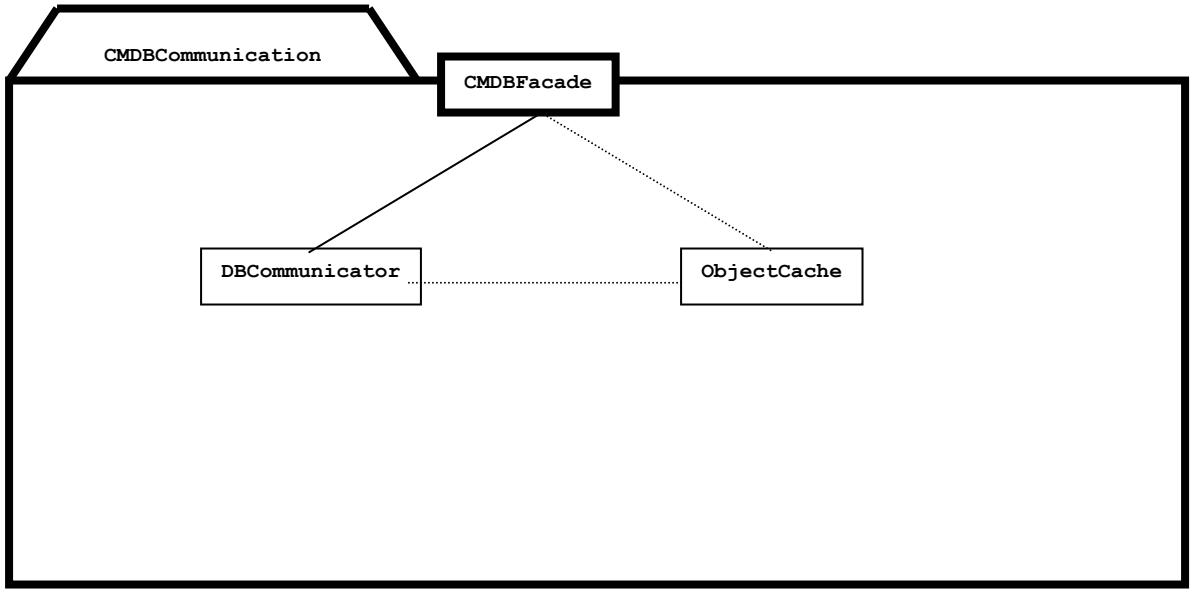

figure 4-1 CMDBCommunication subsystem

[7] See [01], Appendix A.6, page 503, paragraph 1.

4.3.2.2 ConfigurationManagement

The `ConfigurationManagement` subsystem is responsible for the management of configuration items, represented by the `ConfigurationItem` class. All changes in the CMDB can be viewed in the `History`. For every change in the CMDB a new `ChangeRecord` is created. The `ConfigurationManagement` subsystem is accessed through the `ConfigurationItemFacade`. The subsystem is pictured in figure 4-2.

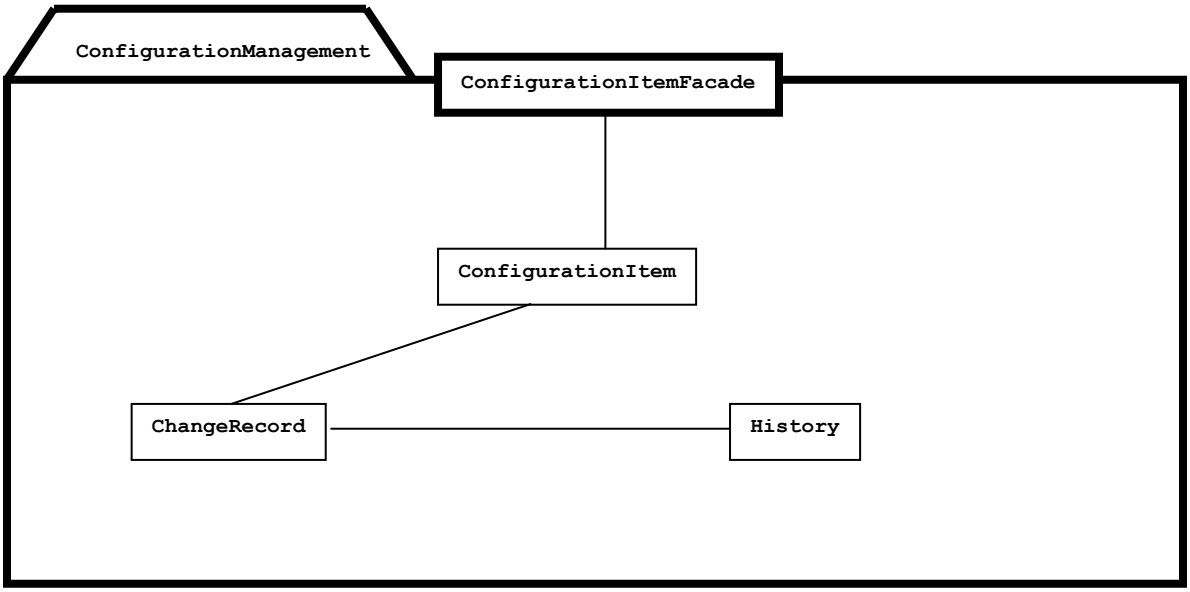

figure 4-2 ConfigurationManagement subsystem

4.3.2.3 IncidentControl

The `IncidentControl` subsystem is responsible for the handling of an `Incident`. If the `Incident` appears to be a `Problem`, a `Notification` is sent to someone of the the support group, which is a `User` of HIST as well. The `Client` receives a confirmation when the `Problem` is solved. The `IncidentControlSubsystem` is accessed through the `IncidentFacade`. The subsystem is pictured in figure 4-3.

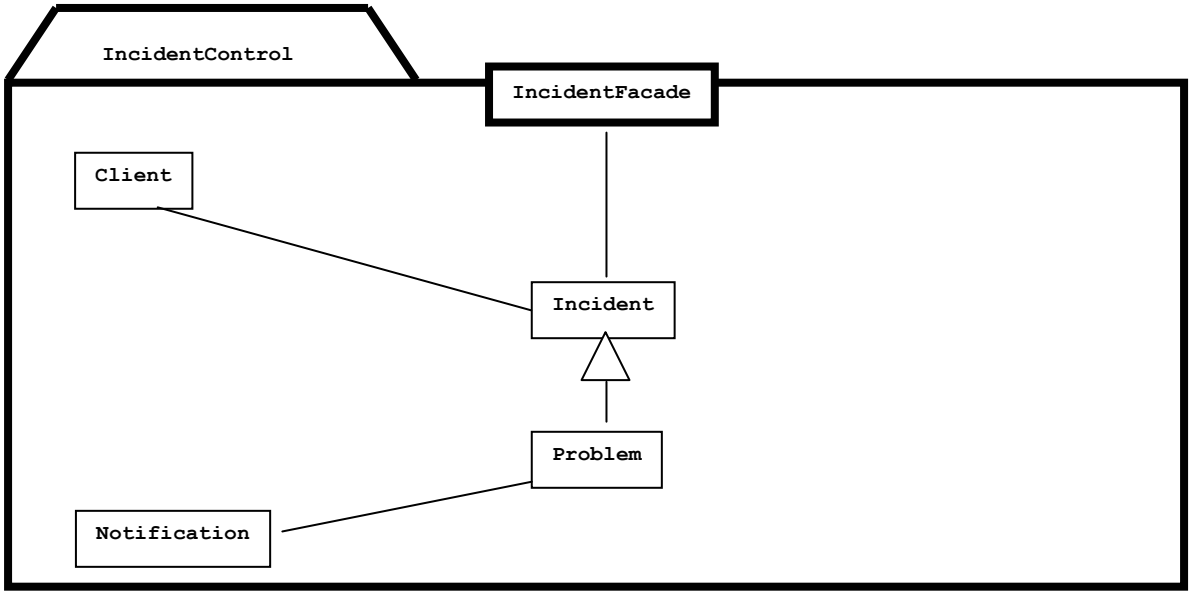

figure 4-3 IncidentControl subsystem

4.3.2.4 UserManagement

The `UserManagement` subsystem is responsible for the authorization of users who start the HIST application. A `User` has a `Permission` profile to conduct his or her specific tasks. A `User` consists of a username/password combination and is stored in the database. The `UserManagement` subsystem can be accessed through the `UserFacade`. The subsystem is pictured in figure 4-4.

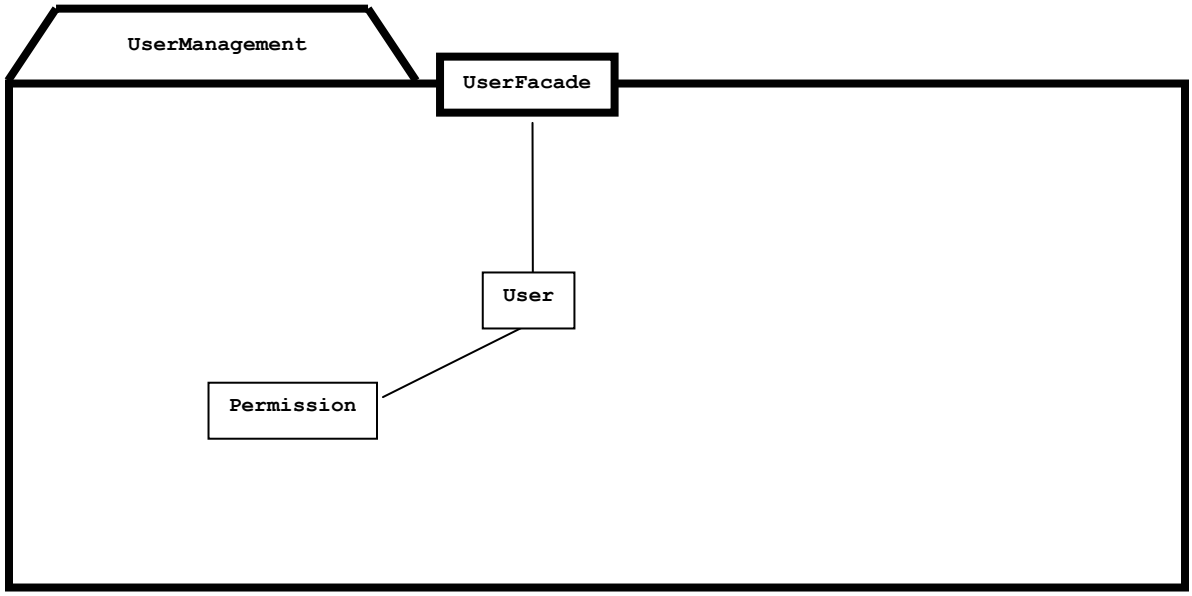

figure 4-4 UserManagement subsystem

4.3.2.5 JFCUI

The `JFCUI` subsystem is responsible for providing the users with an interface to the application. It provides dialogs for CIs, incidents, and problems. The subsystem is pictured in figure 4-5.

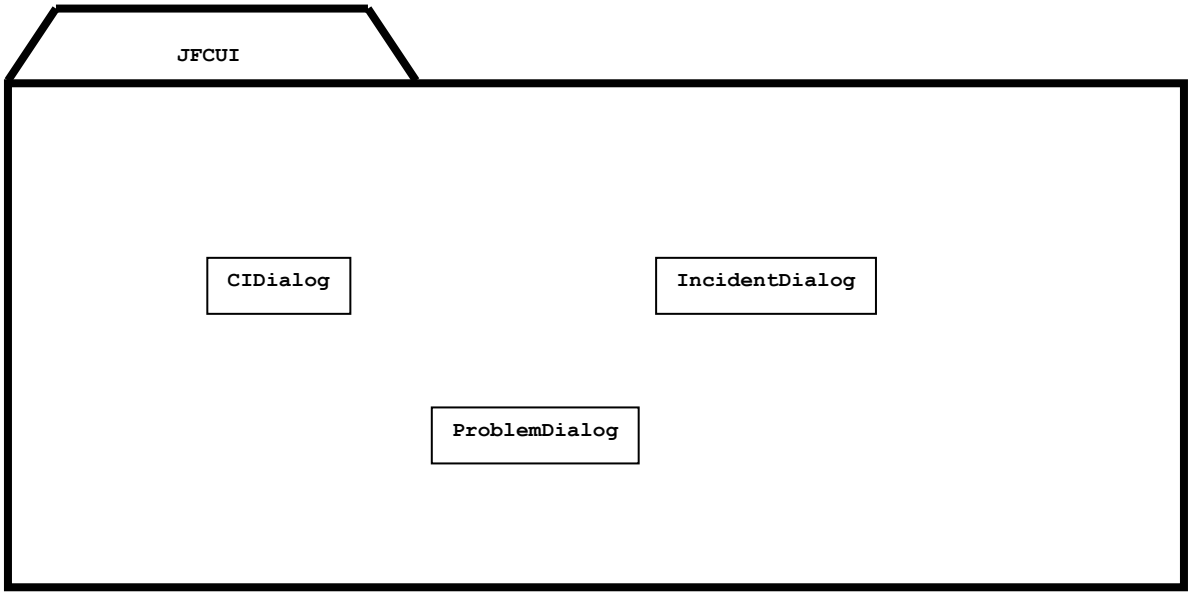

figure 4-5 JFCUI subsystem

4.3.2.6 Servlet UI

The `ServletInterface` subsystem is responsible for providing HD clients with an interface for reporting an `Incident` through the `ReportIncidentServlet`. It also provides feedback on pending problems with the use of a `ProblemInformationServlet`. The subsystem is pictured in figure 4-6.

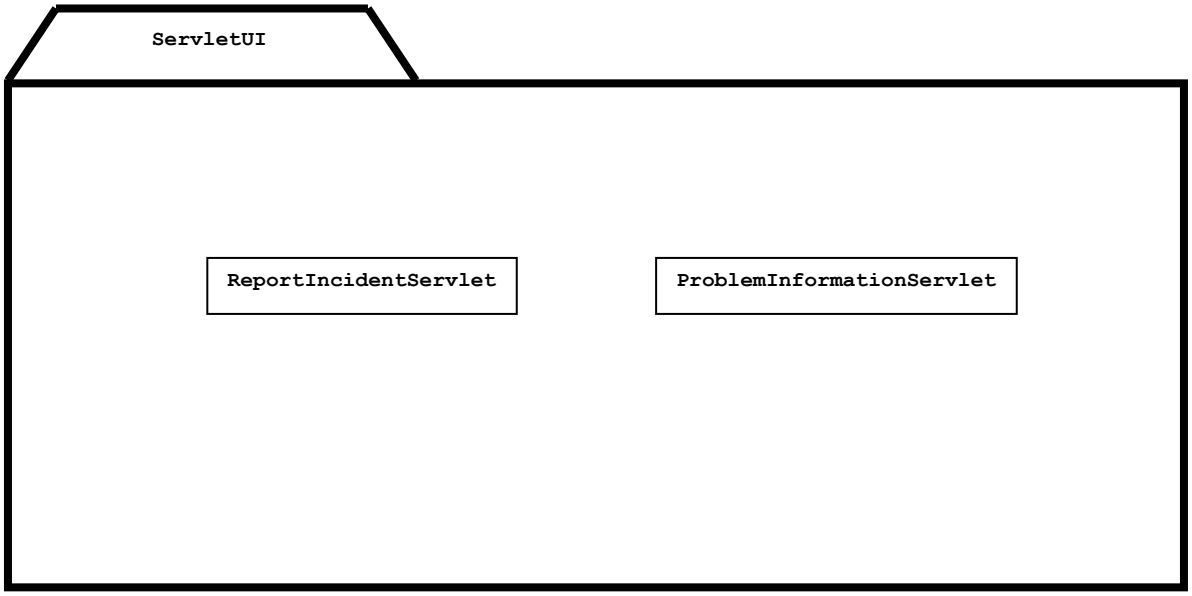

figure 4-6 ServletUI subsystem

4.3.3 Hardware/software mapping

At this moment, there are many different architectures one can choose from to base a software system on. The proposed application requires a central database storage, which can be accessed by multiple clients (see paragraph 3.3.1). Two architectures that comply with the criteria are studied here; they are the client/server and three-tier architecture.

4.3.3.1 Client/server architecture

The client/server architecture can be characterized as a central application and/or data server, which provides it services to multiple clients, connecting through a network. Clients can access these services simultaneously, which imposes that proper locking of data and/or processes is volatile.

The client/server architecture in general does not prescribe a certain setup of the application, but the earliest implementations use the server only as a central data storage, and user interface and application logic are located on the client. The downside of this setup, however, is that it requires a rather high-performance of the client machine. It is also more difficult to control the software installations, since they are decentralized. Another option is to place the application logic and data storage on a central server, and the client only has a front-end for the services. The World Wide Web is an example of this, where different machines logon to servers, browsing their content.

4.3.3.2 Three-tier architecture

Many enterprise solutions are built using a so-called three-tier architecture nowadays, which is partly due to the enormous success of intranet and browser technologies. The architecture is based upon the concept that any solution can be split up into three different functional domains:

- Data;
- Business;
- Presentation.

Each of these domains is mapped onto a different set of hardware items.

The data tier contains the central database, also known as the persistent data storage. In a typical situation, it is provided by a relational database management system (DBMS), such as Microsoft SQL Server.

The business tier contains the business rules of the application, and could be seen as the application logic. It can include a workflow engine, which takes care of proper handling of processes. In respect to the data tier, many processes controlled by the business tier are treated as transactions. The most challenging part of many solutions is the mapping of object data to relational data storage.

The presentation tier is the actual front-end for the user. A common solution is to build a lightweight browser interface, which can be accessed through intranet and/or internet. Since the application logic and data are stored on different tiers, the client's system

demand is rather low. Synchronizing is typically done by session management, which means the business tier keeps track of its clients and their state.

The aims of the three-tier architecture are twofold. First, by separating a solution into three distinct functional domains, management of the application can be performed more easily. Second, the solution is more scalable. The client can use a pretty low profile system, and extra hardware can be placed where necessary (e.g. this could be different for IO-bound and CPU-bound applications). The aim is to minimize inter-tier communications, because in a typical solution network communications are the real performance bottleneck.

4.3.3.3 Architecture implementation with Java

After examination of the two architectures described in the previous two paragraphs, the focus is now on how such architectures can be implemented by making use of the Java Application Programming Interface (API).

Communication with the database can be done in two distinct ways. Open Database Connectivity (ODBC) is a de facto standard for such a communication, and is supported by all major software vendors. However, in order to interact with the Java environment, a second interface is required, which is known as Java Database Connectivity (JDBC). One could use a bridge between JDBC and ODBC, or use a native JDBC driver, bypassing ODBC completely.

A three-tier architecture requires a different approach for application setup than client/server based architectures. Enterprise Java Beans (EJB) allows the building of server-side, component-based applications. Sun Microsystems' definition of EJB is

> The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the Enterprise JavaBeans specification. [8]

EJB is a *Distributed Object Architecture*. Examples of other technologies based on this architecture are Java RMI, CORBA, and Microsoft's DCOM (featured in what is now known as COM+).

4.3.3.4 Comparison between architectures

Two designs for the application have been made, in order to be able to choose the most suitable. The first 'sketch' basically was a three-tier architecture without the use of EJB. The advantage of this system is, like any three-tier architecture application, its flexibility. Because of its highly modular buildup, changes in the 'business-tier' (also referred to as 'middle tier'), can easily be copied in the other layers. The main disadvantage is its complexity. A separate service must run on the server that listens for client requests. Secondly, availability is another issue that's disputable in this approach. Because a

[8] Sun Microsystems' *Enterprise JavaBeans™ Specification, v1.1.* Copyright 1999 by Sun Microsystems, Inc.

separate service runs on the server containing the program's logic, each and every client is dependent on this service. Once it crashes, nobody is able to use the application anymore. See figure 4-7 for an illustration of the architecture.

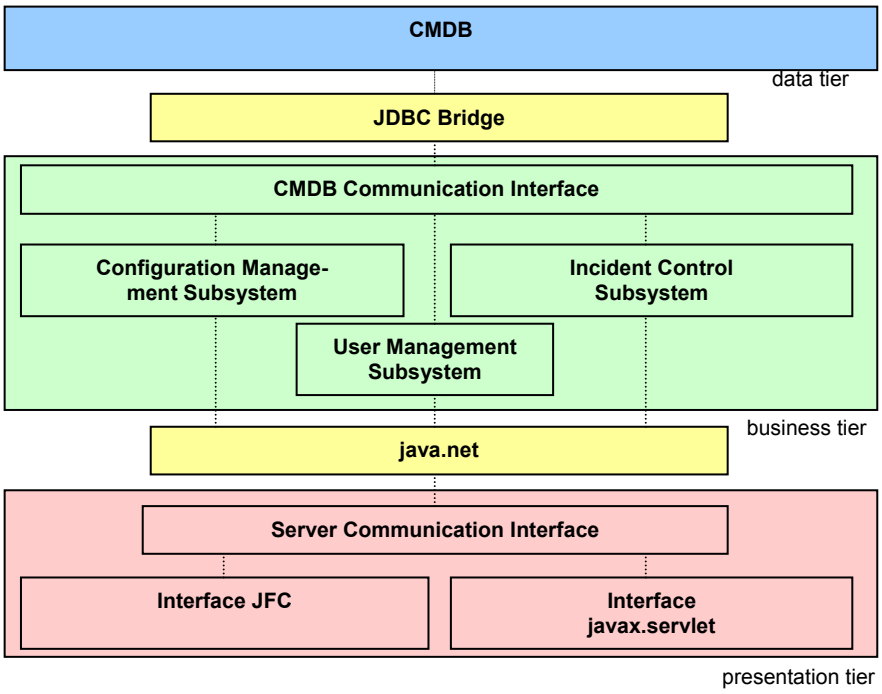

figure 4-7 Three-tier architecture design

The second design is based on the classic client/server architecture. All functionality and presentation are put together in the client program. All data is kept on a server. This approach has the main advantage of its simplicity. The main disadvantage is that this approach is less flexible. If some change in procedures occurs, software engineers must manually adjust the client program. All users must install this new client in order to work with the new procedures. See figure 4-8 for an illustration of the architecture.

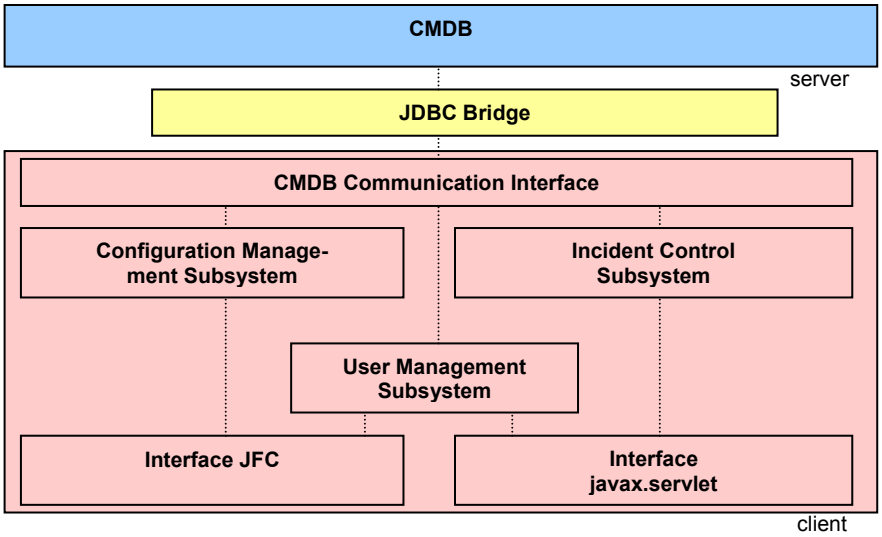

figure 4-8 Client/server architecture design

The User Interface (UI) is also a design issue: the choice is either to have a web-based application or a UI based on the Java Foundation Class (JFC). The main drawback of a web-based application is its increased complexity. In web-based applications, subjects as session management must be taken into account. Another problem is to keep the user's view up-to-date. Using simple HTML/XML, users must press the 'refresh' button of their browser to get up-to-date information. The incorporation of applets makes everything more complex. The main advantage of the web-based approach is that no additional client program is needed; instead users just use their browser. For client convenience, a small web-based subsystem will be built, to provide clients with an interface for reporting incidents as well as to track the status of pending problems (see paragraph 3.3.2).
The alternative is the use of the Java Foundation Classes (JFC). The JFC provides a sophisticated framework to build enhanced user interfaces. Using Swing components it is very suitable to display views on tables [14].

A comparison between client/server and three-tier architecture is given in table 4-2 and table 4-3. Criteria for dependability and cost are taken into account.

Design criterion	Client/server	Three-tier	Definition
Robustness			Ability to survive invalid user input.
Reliability		-	Difference between specified and observed behavior.
Availability	+	-	Percentage of time that system can be used to accomplish normal tasks.
Fault Tolerance			Ability to operate under erroneous conditions .
Security	-	+	Ability to withstand malicious attacks.

table 4-2 Dependability criteria for architecture comparison

Design criterion	Client/server	Three-tier	Definition
Development cost	+	--	Cost of developing the initial system
Deployment cost		-	Cost of installing the system and training the users
Maintenance cost	-	+	Cost required for bug fixes and enhancements to the system
Administration cost		+	Money required to administer the system

table 4-3 Cost criteria for architecture comparison

As far as this application concerns the three-tier architecture is an 'overkill'. Because the application will be quite simple, the cost of applying this technology clearly overtakes the advantages of its flexibility. Furthermore, this technology is still under much development.
The conclusion of this architecture discussion is to choose for the client/server approach. Simplicity and reliability of the application are of the main requirements for the application (see design goals in paragraph 4.1). The characteristic of the desired client/server model, is that the request and update of data can actually span a long period of time (hours). The client application will run for as long as eight consecutive hours, during which the database may be updated by different users. Locking of the database is therefore not possible (nor desired). This model is known as the disconnected client/server model. The user interface will be built using the JFC.

4.3.4 Persistent data management

As described in paragraph 4.3.3, the system will be built according to the client/server architecture, where the server will provide the central data storage. As already stated by the pseudo requirements (see paragraph 3.3.4), this will be managed by Microsoft SQL Server 7. First a design issue regarding trees will be covered. Next a mechanism called data caching is addressed, that should improve performance for client and server communication, followed by the actual data scheme of the relational database. Finally the encapsulation of the database will be described.

4.3.4.1 Storage of tree structure in relational database

The specification of the configuration management system introduces a special relationship between two CIs, the parent-child relationship. Any CI can have multiple child CIs, and together they form a graph. Since it is logically impossible to have cycles in this graph, (when parent A has child B, child B cannot be the parent of parent A, and child B can only have one parent), it is actually a special form of graph, namely a tree [16]. The difference between a tree and a graph is illustrated in figure 4-9.

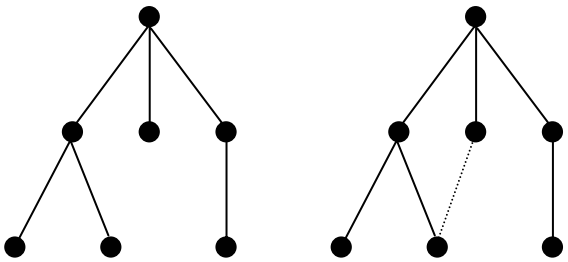

figure 4-9 Difference between tree and graph

The challenge in storing a tree in a relational database is to include constraints that prohibit cycles, but at the same allow for an easy retrieval of the data. One has the possibility to solve this through software, data model, or a combination of these. Several algorithms have been designed to detect cycles in software, but in general it is favorable to keep this constraint part of the data model. In this way, it is simply not possible to store such an invalid structure, instead of using a validation algorithm that could contain bugs.

The tree to be stored has the following characteristics:

- Each CI can have 0..n childs;
- Each CI has 1 parent;
- The root has no parent.

The most straightforward solution is to use a foreign key relationship, in this case self-referencing. Unfortunately, this data model allows the introduction of cycles. To solve this, a special root node will have to be introduced, which may not contain a parent reference, while all other CIs should have one foreign key reference. The structure of the proposed table is shown in table 4-4.

ConfigurationItem
ID
fkParentID {nullable}

table 4-4 ConfigurationItem table with self-referencing key

Another solution is given by [07], which introduces the usage of domains. To specify this domain, two denominators are used, which are LeftValue and RightValue. The smallest left value is 1, while the largest right value is equal to the total number of CIs, multiplied by two. The structure of the proposed table is shown in table 4-5.

ConfigurationItem
ID
LeftValue
RightValue

table 4-5 ConfigurationItem table with domain specification

The advantage of this solution is that it allows an ordering of the siblings at the same level. To illustrate how the domain specification works, look at figure 4-10.

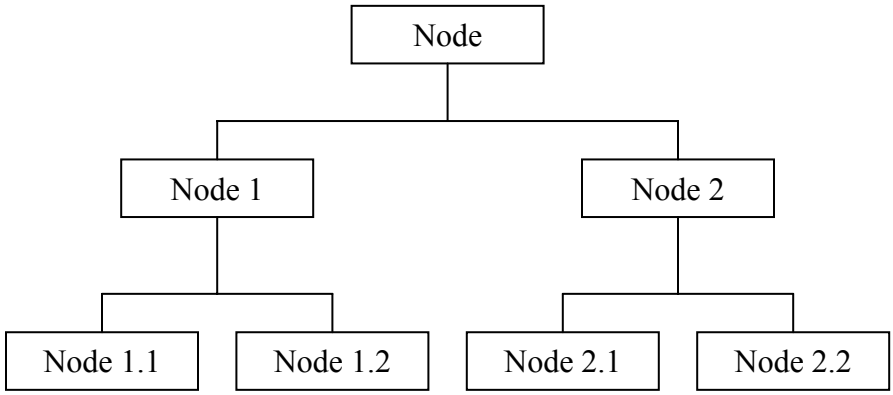

figure 4-10 Sample tree structure

To view the assignments of left and right values, review the entries in table 4-6.

ID	Name	LeftValue	RightValue
1	Node	1	14
2	Node 1	2	7
3	Node 1.1	3	4
4	Node 1.2	5	6
5	Node 2	8	13
6	Node 2.1	9	10
7	Node 2.2	11	12

table 4-6 Assignment of domain values for sample tree nodes

The assignment is being done as followed:

- A LeftValue of a parent node has the value (lowest left value of childs minus 1);
- A RightValue of a parent node has the value (highest right value of childs plus 1);
- A leaf node has the RightValue (LeftValue plus 1);
- Each number between LeftValue and RightValue of the root should be assigned;
- Each number only exists one time within the LeftValue and RightValue columns.

When the two solutions are compared with each other, the most obvious difference is the inclusion of ordering for siblings. The first method described does not support this feature, while second one does. However, ordering of siblings is no characteristic of the desired tree. Another feature of the desired tree, is the difference by first level (excluding root) and next level CIs. In this respect, top-level CIs are regarded as main information source, while next level CIs are treated as detailed information about a top level CI. The desired selection statement (select all top level CIs) can be managed more easily by the first solution than by the second one.

4.3.4.2 Data caching

As described in paragraph 4.3.2.1, the CMDBCommunication subsystem will provide a caching mechanism, in order to improve performance. This cache will be placed on the client, and provide services to the defined subsystems. Paragraph 4.3.3 has identified the desired system architecture as a disconnected client/server model, which means locking of data is not possible. In order to detect if the cache is out-of-sync, a so-called timestamp mechanism is a good solution. It means that the tables in the database that can be updated by users should contain a date field, which identifies the time at which the record was inserted or updated. When the last timestamp stored in the database is different from the one known by the cache, this means the cache is out-of-sync and should be refreshed. Since it is not possible to notify clients of a required refresh operation, a scheduled polling of these timestamps by initiative of the client is required. A reasonable interval would be fifteen minutes, since the system does not have many concurrent users and data is not frequently updated (see paragraph 4.1).

4.3.4.3 Data scheme

The conceptual database model has been divided into two different packages: Configuration and Help Desk. The Configuration package is responsible for the storage and retrieval of CIs, with detailed information. Incident control and logging is supported by the Help Desk package. The data model of the Configuration package is depicted in figure 4-11.

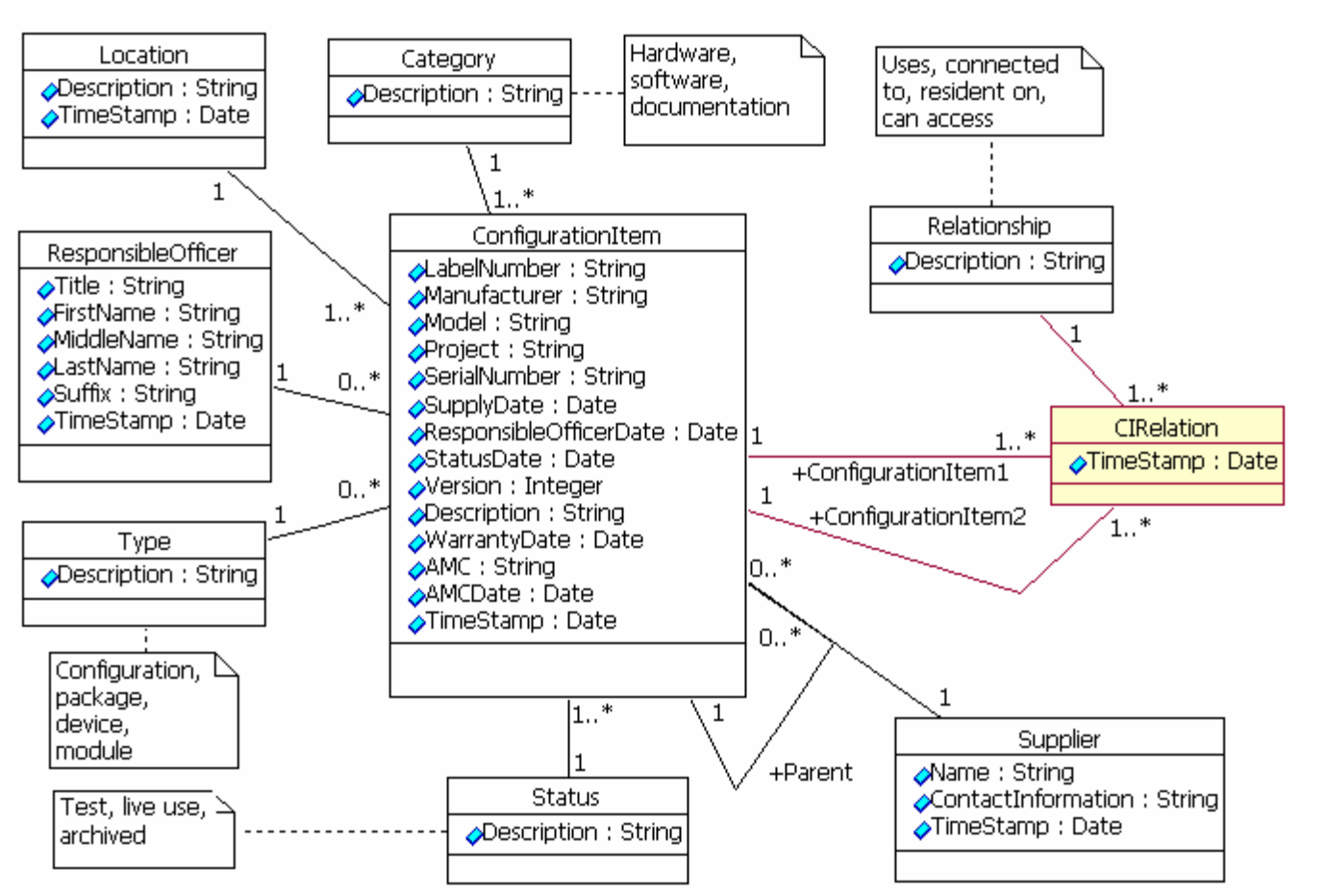

figure 4-11 Configuration data model

The various tables of the data model are described in table 4-9 through table 4-13.

Category		
Stores the category information of the CI.		
Column name	*Type*	*Description*
Description	String	Category description. Records in this column are not likely to change after initial entry.

table 4-7 Category table description

CIRelation		
Defines the relationship between two distinct CIs. →*ConfigurationItem, Relationship*		
Column name	*Type*	*Description*
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-8 CIRelation table description

ConfigurationItem		
This is the central table of the Configuration package and stores the information that is specific to a single CI. Relationships with other tables are provided to store non-redundant information. The *parent* self-referencing relationship is tree information.		
Column name	*Type*	*Description*
LabelNumber	String	Gives the information of the label that is physically attached to the CI.
Manufacturer	String	The name of the CI's manufacturer.
Model	String	Gives model description as given by the supplier.
Project	String	The optional project name which has financed the CI.
SerialNumber	String	The serial number of the CI.
SupplyDate	Date	The date at which the CI has been supplied. →*Supplier*
ReponsibleOfficerDate	Date	Date the current officer became responsible. →*ResponsibleOfficer*
StatusDate	Date	Date the current status was applied. →*Status*
Version	Integer	Automatically updated version number, incremented when replacing an existing CI.
Description	String	General description of the CI.
WarrantyDate	Date	Optional date at which the warranty expires.
AMC	String	Optional Annual Maintenance Contractor.
AMCDate	Date	Optional date at which AMC expires.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-9 ConfigurationItem table description

Location		
Stores the information of the location where the CI is resident. In case of CIs that are given out, this information tells location the CI belongs to, not the current location.		
Column name	*Type*	*Description*
Description	String	Location description.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-10 Location table description

Relationship		
The relationship table stores the relationship information between two distinct CIs.		
Column name	*Type*	*Description*
Description	String	Description of the relationship between two CIs. Records in this column are not likely to change after initial entry.

table 4-11 Relationship table description

ResponsibleOfficer		
Stores the personal information of the officer who is responsible for a CI. AN officer can be responsible for multiple CIs.		
Column name	*Type*	*Description*
Title	String	Optional title of the responsible officer.
FirstName	String	The first name of the responsible officer.
MiddleName	String	Optional middle name the responsible officer might have.
LastName	String	The last name of the responsible officer.
Suffix	String	Optional suffix of the responsible officer's name.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-12 ResponsibleOfficer table description

Status		
The current status of a CI, which tells if the CI is active or not.		
Column name	*Type*	*Description*
Description	String	The status description. Records in this column are not likely to change after initial entry.

table 4-13 Status table description

Supplier		
The supplier who supplied the CI to the organization.		
Column name	*Type*	*Description*
Name	String	The name of the supplier.
ContactInformation	String	Optional unstructured contact information of the supplier.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-14 Suplier table description

Type		
Type gives the type information of a CI.		
Column name	*Type*	*Description*
Description	String	Type information description of a CI. Records in this column are not likely to change after initial entry.

table 4-15 Type table description

The data model of the Help Desk package is depicted in figure 4-12.

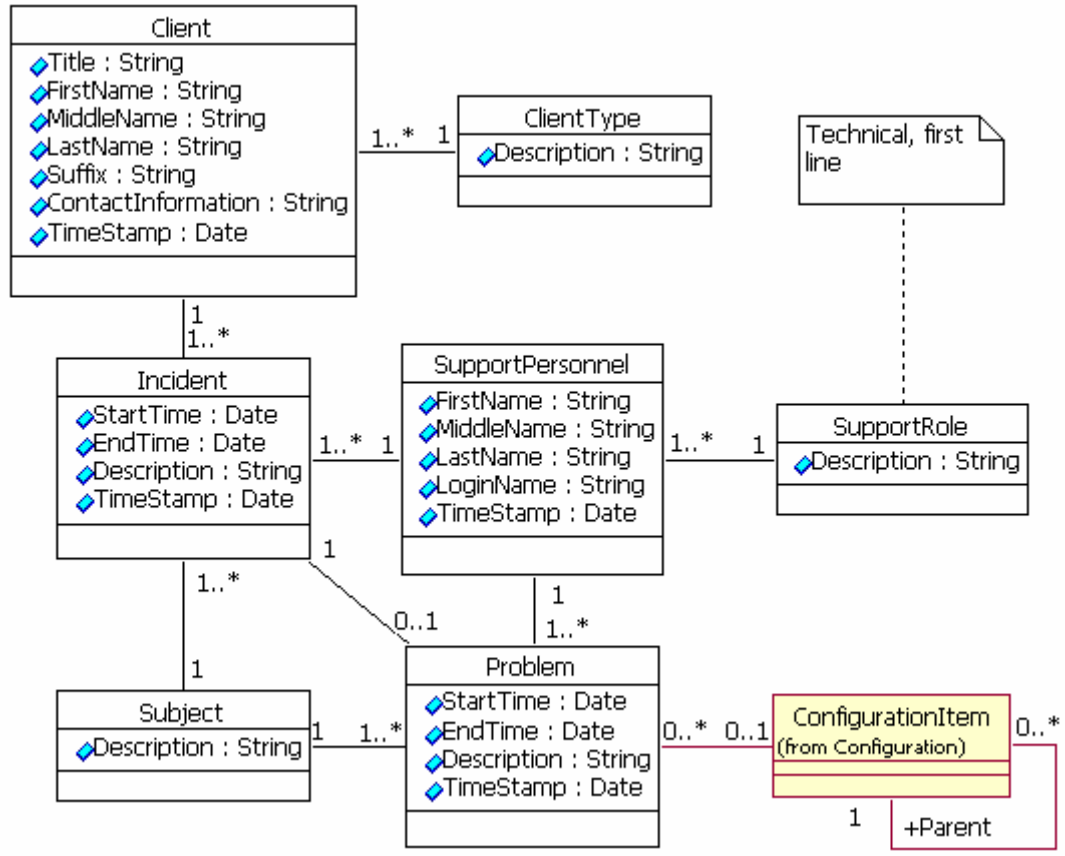

figure 4-12 Help Desk data model

The various table of the data model are described in table 4-16 through table 4-22.

Client		
Stores the personal information of clients, who can report an →*Incident*.		
Column name	*Type*	*Description*
Title	String	Optional title of the client.
FirstName	String	The first name of the client.
MiddleName	String	Optional middle name the client might have.
LastName	String	The last name of the client.
Suffix	String	Optional suffix of the client's name.
ContactInformation	String	Detailed information on how the client can be contacted.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-16 Client table description

ClientType		
Gives the category the →*Client* belongs to.		
Column name	*Type*	*Description*
Description	String	Category description.

table 4-17 ClientType table description

Incident		
The incident table keeps information on logged inquiries received by the HD. An incident could optionally lead to a →*Problem*. An incident in logged by a →*SupportPersonnel* and is reported by a →*Client*. The →*Subject* is a global incident description.		
Column name	*Type*	*Description*
StartTime	Date	The time the incident occurred.
EndTime	Date	The time the incident was handled.
Description	String	A detailed description of the incident.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-18 Incident table description

Problem		
The problem table keeps information on problems reported by the HD. A problem is handled by a →*SupportPersonnel* and is linked with an →*Incident*. The →*Subject* is a global problem description.		
Column name	*Type*	*Description*
StartTime	Date	The time the problem was reported.
EndTime	Date	The time the problem was handled.
Description	String	A detailed description of the problem.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-19 Problem table description

Subject		
The subject table gives a global indication of an →*Incident* and a →*Problem*.		
Column name	*Type*	*Description*
Description	String	The subject description.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-20 Subject table description

SupportPersonnel		
Stores the personal information of support personnel, who can be responsible for solving problems or handling incidents. →*SupportRole*		
Column name	*Type*	*Description*
FirstName	String	The first name of the support person.
MiddleName	String	Optional middle name the support person might have.
LastName	String	The last name of the support person.
LoginName	String	Login name of the support person used to access the system.
TimeStamp	Date	Time at which the item has been stored or updated in the database.

table 4-21 SupportPersonnel table description

SupportRole		
Stores the personal information of support personnel, who can be responsible for solving problems or handling incidents.		
Column name	*Type*	*Description*
Description	String	Description of the role the support person has.

table 4-22 SupportRole table description

4.3.4.4 Encapsulation of the database

As described in paragraph 4.3.2, access to the database will be regulated by a separate subsystem, called CMDBCommunication. This subsystem will control the access to the database, as well as manage the connections with it. Since this subsystem is actually located on the client machine, a connection for each single user is required.

To improve the performance of the application, a caching mechanism is introduced that is kept on the client. The subsystem has the responsibility to synchronize this cache with the actual database. Since the application is developed making use of the Java language and the object-oriented approach, another issue is the mapping of object calls to relational data. The subsystem will allow for a object-oriented request mechanism, and map this internally to relational-oriented methods. Again, the cache should be helpful. The services of the CMDBCommunication subsystem are exposed by a facade, which is being called by the various other defined subsystems.

4.3.5 Access control and security

Every user will have its own account in the database. Users with the same permission level will be grouped together in a certain role. Different roles will be created for `HelpDeskEmployee`, `ConfigurationManager`, `TechnicalEmployee`,

`DirectorOfIT`, each with its own level of permission. This database account will also be used to provide users access to the application itself.

Communication between the client application and the database is done through a JDBC bridge or driver. There is a security risk in this way of communication: if someone who is working on a computer on the same network cable installs a so-called sniffer program, he/she may able to retrieve the password of a user of HIST. This risk can be reduced if we restrict access to the database to the local network. The risk can be further reduced if we restrict user access to the database to a small set of 'necessary' stored procedures. Users must be prevented from being able to modify database tables directly. Several commercial JDBC bridges or drivers also have the option to encrypt the communication using 40-bits encryption or even higher. Making use of such a bridge or driver should make it virtually impossible for unauthorized users to access the database.

4.3.6 Global software control

The client program needs to be kept up-to-date with the database. Once one client changes something in the database, all other client programs must be notified to synchronize with the changes. There are a few options to achieve this:

1. Let clients communicate with each other using a communication object. In order to archive that clients know each other, in an additional table in the database the 'online' client addresses are stored.
2. Install a server object that knows all the clients and notifies each and all of them if something changes.
3. Let all clients 'poll' the database regularly to check whether anything has changed. Timestamps in the database can be used to check this without performance drawbacks. The `ObjectCache` in the `CMDBCommunicationSubsystem` can keep track of changes in the database.

The first option has the main drawback that it will increase network traffic dramatically. On top of that: it requires quite a lot of extra effort to install it. The most elegant is option number 2, however many extra costs are involved in creating a server object. The third option is the most realistic one, the refresh time interval will be manually adjustable, and a refresh button will be installed. This option will therefore be used in the system design, see paragraph 4.3.4.2 for more details.

5 Object design

Object design closes the gap between system design and the final software implementation. Subsystems identified are refined and specified in more detail, resulting in package and interface definitions. This chapter starts with the trade-offs that have been made, followed by a package definition. Due to their size, the class-interface definitions can be found in a separate document.

5.1 Object design trade-offs

During the system design, several design goal trade-offs have been specified, which are described in paragraph 4.1. The major constraint for the project is delivery time, since it is fixed. During the object design, several trade-offs had to be made in order to meet the goal of delivering a functional system on time.

The first design issue that has been dropped involves the caching mechanism, described in paragraphs 4.3.4.2 and 4.3.6. In the current situation, information from the database is read at the moment the client program is started, and only actions performed by this client are synchronized. This imposes no real problem, however, since in the current situation access to the database is only done through one client. A caching mechanism that is also responsible for resynchronization would not justify the time investment that has to be made. To be future-ready, however, the data scheme does support so-called timestamps. See paragraph 4.3.4.3 for more details.

Paragraph 4.3.2.6 specified the Servlet UI subsystem, which provides a web-based interface for client to view pending problems and to report incidents. This feature has been cut due to time constraints. Due to the current limited amount of incidents to be handled by the HD, this does not have much impact. When the support given by the HD facility is getting larger, this feature would be a good investment, however. The current architecture should allow for a straightforward implementation.

The UserManagement subsystem, described in paragraph 4.3.2.4, has been cut back in functionality too. Access control is now only regulated on database level, but not on application level. In practice this means all users who have login permission can access all parts of the application, but will receive an error when an operation requires specific permission(s). These errors are generated by the database management system.

The configuration management database acts as a basis for the entire application. The current implementation only supports the ITIL processes incident management and configuration management (see paragraph 3.1.2). The process change management is only partially supported, where the most notable difference is the lack of change records. Together they should provide a complete history of changes, but this is not supported by the current implementation.

The final design issue that has been modified is the feature of generating management information. The current implementation has a limited support only: it displays incidents and problems filtered by day. Specific information can be retrieved from the database, however, by specification of so-called SQL queries.

5.2 Packages

Due to the relatively small size of the project, the subsystems defined in paragraph 4.3.2 almost directly map to a package defined in the object design. The subsystem Servlet UI described in 4.3.2.6 has been removed from the system design (see paragraph 5.1 for more details). The dependencies between the five identified packages are depicted in figure 5-1.

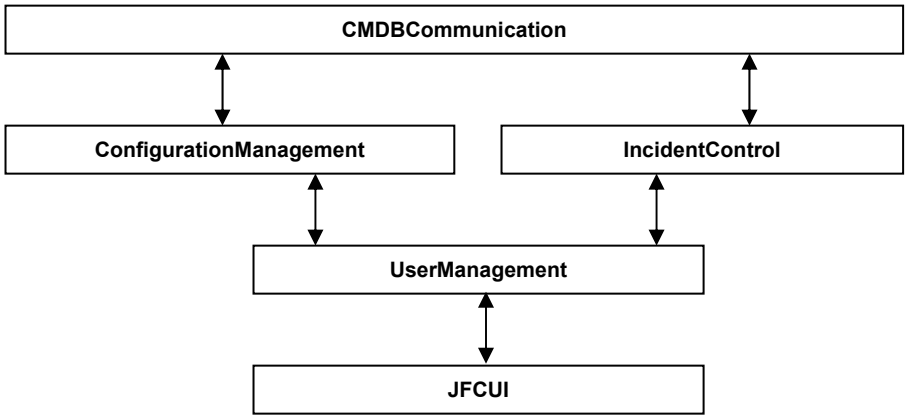

figure 5-1 Package dependencies

As can be seen in the figure, the CMDBCommunication package is the package that communicates with the ConfigurationManagement and IncidentControl packages. As described in paragraph 4.3.2.1, it is responsible for be communication between the application and the database.

The UserManagement package acts as a regulator between clients and the application. In this case, the client is represented by the JFCUI package, since this package defines the actions that can be performed by a client.

The identified constraints and rules are defined and implemented in the ConfigurationManagement and IncidentControl packages. Together they could also be described as the application logic.

6 Implementation

This chapter deals with some implementation details on a project level. Primary focus of this chapter is the interaction between the designers and users of the application.

6.1 Implementation goals

Early on in the project, it has been decided that the actual design and implementation would not be done jointly with the IRM staff, the primary users of the application. The main reason is their lack of some fundamental concepts required to implement the desired application. These concepts include:

- Object-oriented programming in Java;
- Database design and implementation with SQL Server.

In addition, the staff has no experience or understanding of global software engineering concepts, such as described in [01]. It was therefore not feasible to work on this project jointly, since training would simply not be possible within the limited time span available. The aims of the project in respect with the IRM staff have therefore been formulated in the following goals:

- The IRM staff should have a global understanding of the ITIL concepts configuration management and incident management.
- The IRM staff should be able to manage the system independent of any third parties.
- The IRM staff should have influence on the tool implementation on a fundamental and detailed level.

This does mean, however, that the IRM staff will not be able to manage additional functional requirements, or will able to solve any bugs. If adjustments on a source code level are required, they will have to be done by external parties.

6.2 User training

In order to meet with the goals identified in paragraph 6.1, several efforts have been made. The introduction and familiarization on the ITIL concepts has mainly been done during staff meetings. Appendix 2 gives the minutes of such a meeting. Computer presentations as well as group discussions were the main methods used. During the early sessions, the staff members were able to raise their opinion about the desired tool.

During the start of the project, an introduction course on database administration with Microsoft SQL server has been given. An overview of this course is given in an external document. In addition with a management manual, this should provide the IRM staff with the knowledge to manage the system, without having to rely on any third parties.

In order to familiarize the staff with a functional as soon as possible, several prototyping concepts have been used[9]. However, because of the high dependencies between the various subsystems, incremental prototyping was not very successful. A working version of the tool was ready at two-third of the project, at that time joined sessions with the users were started. The final week was reserved for on-site training and resolving of any small bugs.

[9] See [17], page 67, paragraph 4.10.

7 Conclusions and recommendations

The final chapter of this report describes the conclusions that are made after evaluation of the design goals and their results. Based upon the conclusions, several recommendations are made next.

7.1 Conclusions

The overall aim of the HIST project is to improve the first-line support services provided by the IRM, as well as the perception of its users. To meet with this objective, the existing helpdesk is restructured according to the guidelines of ITIL. A tool has been built to assist the IRM staff in their activities, which supports the logging of inquiries and gives access to a so-called configuration management database.
Both design goals set for the tool have been met on time, and have resulted in a basic, but fully functional system. Staff training had a high priority, with focus on both understanding of concepts as hands-on experience. With knowledge of several ITIL defined processes the staff is not dependent on external parties in order to manage the application. Actual service improvement is hard to measure if purely based upon these conclusions, however.

On an organizational level a trade-off had to be made between the functionality of the application and the participation of the IRM staff, the users. It would have been beneficial if the staff could have been more involved in the design process itself, but this was not feasible due to their lack of expertise. Coordination between this project and several others would have benefited both the staff and the project, since it would have been possible to limit the set of used tools and technologies, and more applicable training would have been possible.

Due to a quite aggressive schedule, additional functionality of the tool had to be dropped in the end. It did not allow for too many setbacks too, issues such as illness were not accounted for. This resulted in a limited time for the resolving of problems and bugs, and less time for guidance of the staff.

During the early stages of the project, it was already questionable if the desired application could be put to real use. Investigation learned that the amount of enquiries the HD facility received on a daily basis did not justify the implementation of a tool, and paper registration would have been sufficient. The concepts introduced by ITIL are mainly applicable to larger organizations, and can be regarded as overkill for smaller ones. Especially configuration management requires a very thorough administration of data, and will fail if this done on an ad hoc basis.

The conclusions are listed briefly below:

1. The specified tool has been developed on time and meets its functional requirements.
2. The IRM staff is able to manage the system independently, but cannot extend its functionality.
3. Interests of the stakeholders conflicted in several ways, where exchange of knowledge on the development process suffered.
4. In the current situation, it is questionable if the organization is yet fully equipped to cope with all of the concepts introduced by the project.

7.2 Recommendations

Based upon the conclusions described in paragraph 7.1, several recommendations can be made. They are described in this paragraph.

The application has the aim to improve the provided services, as well as the perception of its users. This aim will have to be verified, and could be done by e.g. questionnaires. The results will help the management and implementation process of the application.

When the current application has been put to full use, and the organization has matured enough, it is recommended to implement the ITIL processes problem and change management. They will further improve the services provided by the IRM.

If the amount of enquiries to he HD has risen, it would be beneficial to limit this. A client database, which verifies the user and its right for support, could filter the enquiries. A web-based enquiry form would fasten the enquiry process.

When the developed tool is being used more often, it would be beneficial to implement some of the design issues that have been dropped due to time constraints. It will allow for better concurrent use, and give more management insight.

List of tables and figures

References

[01] Bruegge, B. and Dutoit, A. H., *Object-oriented software engineering – Conquering Complex and Changing Systems*, Prentice-Hall International, Inc., New Jersey, 2000.

[02] Carroll, J.M. (ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*, Wiley, New York, 1995.

[03] CCTA, *IT Infrastructure Library - Configuration Management,* Ninth impression, The Stationery Office, Norwich, 1998.

[04] CCTA, *IT Infrastructure Library - Help Desk*, Fifth impression, HMSO, 1995.

[05] CCTA, *IT Infrastructure Library - Problem Management*, Ninth impression, The Stationery Office, Norwich, 1998.

[06] CCTA, *IT Infrastructure Library – IT Service Management Case Studies*, HMSO, 1996.

[07] Celko, J., *SQL for Smarties, Advanced SQL programming*, 2nd edition, Morgan Kaufman Publishers, San Francisco California.

[08] CUSAT/TU Delft, *MHO/CUSAT/DUT/INFOCUS Formalization of ICT management at CUSAT – Technical Mission Report 06 May – 04 June, 2000*, 2000.

[09] CUSAT/TU Delft, *MHO/CUSAT/DUT/INFOCUS Formalization of ICT management at CUSAT – Technical Mission Report 14 October – 04 November, 2000*, 2000.

[10] Cochin University of Science and Technology, *Masterplan for the implementation of the IS/IT policy – Information Resource Management*, 1997.

[11] CUSAT/TU Delft, *MHO Cooperation project - Project proposal on Information Policy Development and Implementation (INFOCUS)*, 1997.

[12] CUSAT/TU Delft, *MHO/CUSAT/DUT/INFOCUS Information Policy Development and Implementation - Progress Report 1999*, 2000.

[13] Dumay, M.J. and Groeneweg, R.P., *CUSAT IT Support Management – Project Proposal*, 2000.

[14] Flanagan, D., *Java Foundation Classes in a Nutshell – A Desktop Quick Reference*, O'Reilly & Associates, Inc., Sebastopol, 1999.

[15] Flanagan, D., *Java in a Nutshell – A Desktop Quick Reference*, 3rd edition, O'Reilly & Associates, Inc., Sebastopol, 1999.

[16] Grimaldi, R.P., *Discrete and Combinatorial Mathematics*, Addison-Wesley.

[17] Hughes, B. and Cotterell, M., Software Project Management, 2nd edition, McGraw-Hill, Berkshire, 1999.

[18] Kesarlal, J.A., *Change Management tussen meerdere partijen*, graduation report, Faculteit der Technische Wiskunde en Informatica, TU Delft, 1996.

[19] Looijen, M., *Beheer van Informatiesystemen,* 4th edition, Kluwer Bedrijfsinformatie b.v., Deventer, 1999.

[20] Monson-Haefel, R., *Enterprise JavaBeans*, 2nd edition, O'Reilly & Associates, Inc., Sebastopol, 2000.

[21] Rambaran Mishre, S.P., *Migratie naar Client/Server met ITIL*, graduation report, Faculteit der Technische Wiskunde en Informatica, TU Delft, 1997.

[22] SUNY Computer Officers Association (COA), *Help Desk Software Survey May 1999*, See: http://www.oswego.edu/~sturr/HelpDesk.html, 1999.