Master's Thesis

# ITIL Configuration Management

Requirements analysis and prototype implementation

carried out at the

Information Systems Institute
Distributed Systems Group
Technical University of Vienna

under the guidance of
o.Univ.Prof. Dr. Schahram Dustdar
and
Dr. Manfred Siegl
as the contributing advisor responsible

by

Jürgen Langthaler
Stephensongasse 1/2/18, 1210 Vienna
Matr.Nr. 0025951

Vienna, September 9th, 2007 ____________________

*For my parents.*

# Abstract

The ITIL framework has been designed by experts and holds best practices to improve service management in a company. The two main processes *IT Service Support* and *IT Service Delivery*, which both consist of several sub processes, aim to manage provided services and to improve their quality. Configuration Management, which is part of the *IT Service Support* process, has a central function within the ITIL framework. All important parts of the IT infrastructure are held and managed in the Configuration Management Database.
Each subprocess of the ITIL framework makes different demands upon a Configuration Management. The challenge is to develop a Configuration Management system which meets all demands of the sub processes and all common requirements of a Configuration Management system.
During this master's thesis a requirements analysis for a Configuration Management system has been performed and, based on the results of the analysis, a Configuration Management system prototype has been developed.

# Zusammenfassung

Das ITIL Framework wurde von Experten zusammengestellt und beschreibt eine Reihe von Richtlinen um das Management von Diensten zu verbessern. Die zwei Hauptprozesse *IT Service Support* und *IT Service Delivery*, welche wiederum in Teilprozessse gegliedert sind, versuchen das Management von IT Services zu beschreiben und deren Qualität zuverbessern.
Configuration Management, welches Teil des Prozesses *IT Service Support* ist, stellt einen zentralen Bestandteil des ITIL Framework dar. Alle wichtigen Teile der IT Infrastruktur werden in der Configuration Management Database erfasst und verwaltet.
Jeder Teilprozess des ITIL Frameworks stellt unterschiedliche Anforderungen an das Configuration Management. Die Herausforderung ist es, alle Anforderungen der Teilprozesse zusammen mit gemeinsamen oder grundsätzlichen Ansprüchen innerhalb eines Systems erfüllen zu können.
Im Zuge dieser Diplomarbeit wurde eine Anforderungsanalyse für ein Configuration Management System erstellt und basierend auf den Ergebnissen der Analyse ein Prototyp eines Configuration Management Systems implementiert.

# Contents

# List of Figures

# List of Tables

Part I

# Introduction

## 1 Motivation

The effort of maintaining the hardware stocks of any company is increasing dramatically according to the size of the company and the number of customers being served. For being successful on the IT market it is absolutely necessary to find and solve every problem, which occurs in the complex system of a company's IT infrastructure quickly, and to keep downtimes as low as possible. After an error has been detected either by hardware sensors or a customer reporting a malfunction it is important to localise the cause of the problem and to act immediately. The internal workflow has to be defined exactly to keep the quality of the service measurable and traceable. The ITIL framework [oGC] provides a process model and best-practices to optimise internal workflow, on the other hand it also defines working procedures to improve quality. The data gathered by the different processes of ITIL allows the generation of statistical analyses.

## 2 Problem description

One of the main objectives of the Configuration Management process is to provide data for other ITIL processes. The data which is stored in the Configuration Management Database (CMDB) may vary greatly depending on the types of services provided by a company. Thus a Configuration Management system has to offer enough flexibility to hold arbitrary data and to be adaptable whenever the business model of the company changes.
Apart from flexibility each process of the ITIL framework makes different demands upon the Configuration Management and a Configuration Management system has to fulfil certain requirements which are common to all processes or required by the Configuration Management process itself.
The problem is to discover and analyse all requirements and build a Configuration Management system which is flexible enough and fulfils as many requirements as possible.

## 3 Objective

The objective of this master's thesis is to perform a requirements analysis which discovers all requirements on a Configuration Management system. The surrounding processes of a Configuration Management have to be taken into account to find out which demands each ITIL process has on a Configuration Management. The requirements analysis will be taken as base for a prototype implementation which is capable of holding arbitrary types of Configuration Items. The prototype has to follow the guidelines provided by ITIL.
The prototype should be able to store any hardware inventory with all necessary attributes in the Configuration Management Database. The information stored will not be limited by the hardware found in a company but also include staff who is responsible for the maintaining of the hardware as well as the relations of the Configuration Items to each other. Furthermore services provided by the company need to be added to the database with corresponding customer relations to the services.

## 4 Organisation of this thesis

The introduction is Part I.

Part II of this thesis will give an overview of the ITIL framework and explain the goals and tasks of the various processes.

Part III will describe the Configuration Management process in detail. It will explain the goals and activities of a Configuration Management.

Part IV will give the results of the requirements analysis which was performed to discover the requirements for a Configuration Management system. This part will analyse which demands other processes of the ITIL framework have on a Configuration Management system as well as common requirements.

Part V will first focus on the design considerations regarding the prototype implementation and then give an overview of the system architecture and techniques used during the implementation.

In part VI a verification of the thesis will be performed.
Part VII lists and describes some topics of future work which may emerge from this master's thesis.

Part VIII will summarise the results of the thesis.

Part IX is the appendix which will give details to the structure of the database used for the prototype implementation. Also an installation and configuration manual is given which will be useful for running the prototype system.

Part II

# The ITIL framework

The IT Infrastructure Library (ITIL) is a collection of guidelines and best practices to improve IT service management. It has been developed in the United Kingdom during the 1980's by the Central Computer Agency (CCTA) which is called Office of Government Commerce (OGC) nowadays. Members of computer centres, distributors and other specialists working as consultors and trainers contributed their knowledge to the ITIL framework.

ITIL provides a standardised process model which defines goals, workflow, input and output of each process. It helps to improve and measure quality of services on a continuous basis.

The ITIL framework consists of the following parts (see figure 1):

- **The Business Perspective:**

  The main task of the Business Perspective is to find a common understanding of IT services which are needed for business processes between the customers and the service providers [dw].

- **Service Support:**

  This part describes which actions are needed to maintain and support IT services. It provides the main contact point for users too [oGC05b].

- **Service Delivery:**

  Planning and delivering IT services are the main tasks of this part [iL00]. Also the requirements for providing a service for a customer are handled in this part.

- **ICT Infrastructure Management:**

  Information and Communication Technology (ICT) Infrastructure Management is responsible for planning the IT infrastructure. It is responsible for the management of the environment which is needed to provide a service as well as planning network services.

- **Planning to Implement Service Management:**

  This part provides information about how to implement the ITIL management processes in a company and how this project can be planned, introduced and maintained. A continuous monitoring and improving of the ITIL implementation is needed.

- **Application Management:**

  The management of the software lifecycle is the main task of this part [ZS05]. This includes the phases Requirements, Design, Build, Deploy and Optimise.
  In addition to the active lifecycle of software the discontinuation of a service has to be taken into account.

- **Security Management:**

  The main task of Security Management is to provide information in terms of security schooling and to protect the data from unauthorised access. Security Management is also responsible for monitoring active security policies and the level of security which has been defined in a Service Level Agreement (SLA).

ITIL describes the lifecycle of a service with the two processes *IT Service Support* and *IT Service Delivery.* The following sections will provide further details for those two processes and their sub-processes. A basic knowledge about the ITIL framework is important to understand which role Configuration Management has, which information it provides and which other processes use the data it provides.

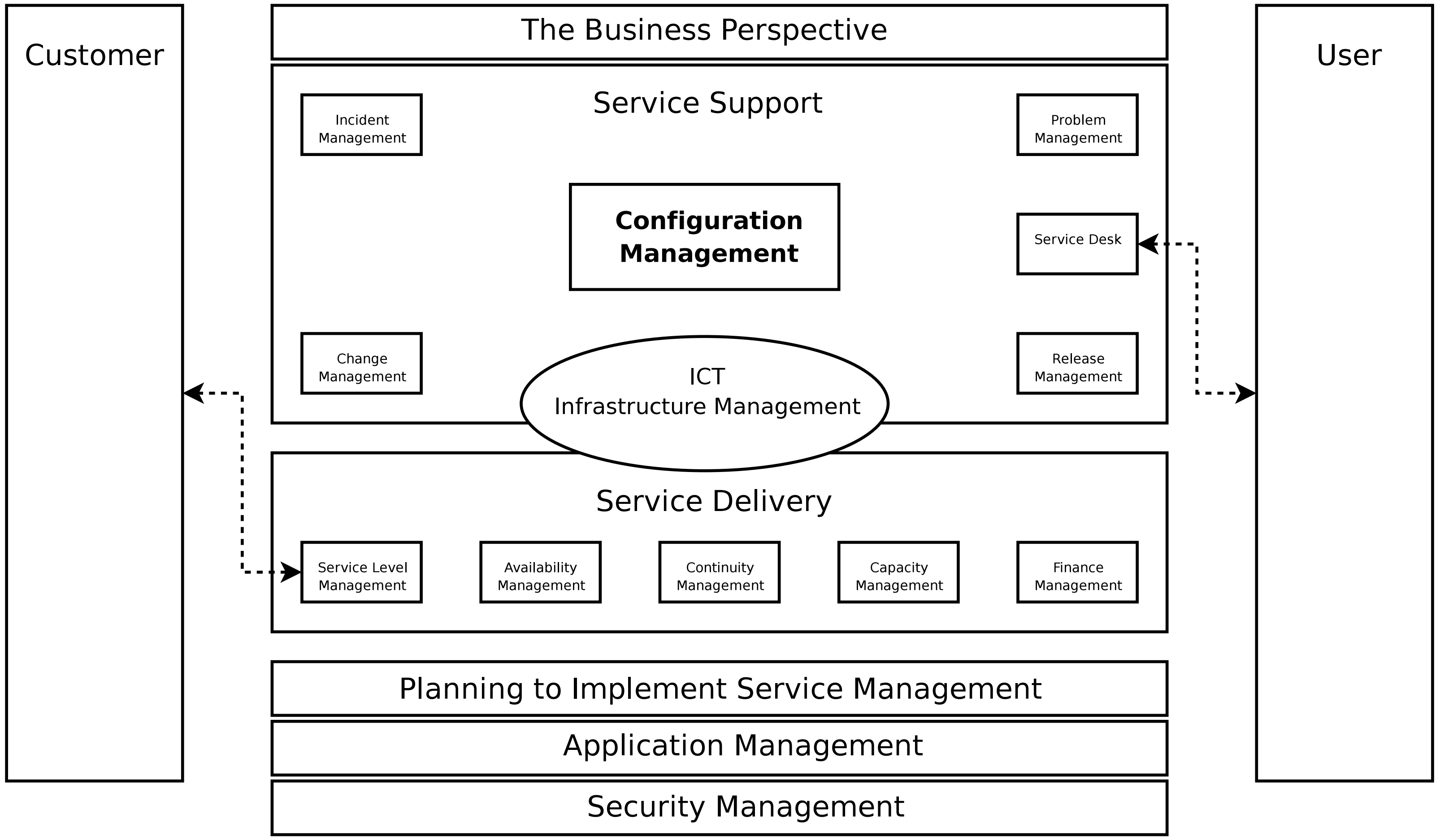


Figure 1: The ITIL library

# 5 IT Service Support

IT Service Support includes operational management processes, which deal with support tasks necessary to provide a service. The main task of the Service Support process is to ensure that the users of the services provided are able to access them. The Service Support part of the ITIL framework consists of several subprocesses whose description will follow now.

## 5.1 Single Point Of Contact

The *Service Desk* is responsible for answering customer calls and handling support requests. It is the Single Point Of Contact (SPOC) for customers and should provide fast solutions for known problems and useful information about frequently asked questions. The Service Desk is the main point for users to report incidents and service requests. The Service Desk is considered the first level support and is responsible for coordinating second- and third level support.

## 5.2 Incident Management

Incident Management is the basically reactive process of taking requests and messages from customers about problems with services. Each incident reported has to be documented. The aim is to solve any problem a user has as good as possible and to make services available again in short time. The Incident Management is responsible for the detection, recording and classification of an incident. The Incident Management uses workarounds and information about known errors created by the Problem Management. Incident Management acts on an effect centred basis, aiming to reduce or circumvent the impacts of a problem a user got.

## 5.3 Problem Management

Problem Management is responsible for solving problems fast and efficient. It covers all three phases of a problem:

- **Incident control:**

  The Problem Management helps the Service Desk to analyse and solve difficult or comprehensive problems. Different groups of specialists, who aim to solve a specific problem, are coordinated by the Problem Management.

- **Problem control:**

  Identify, analyse and document all causes of failures to prevent the recurrence of an error. All the impacts of a problem should be analysed and rated to assure adequate use of resources. Problem control should try to identify or predict possible failures to assure a high availability of services.

- **Error control:**

  Problem Management supports the Error Control by triggering change requests and filtering known errors to prevent multiple reporting.

The Problem Management provides workarounds and solutions for problems and is responsible for filing Requests for Change which are handled by the Change Management. To perform these tasks the Problem Management needs details about the incidents, reported by the Incident Management, and information about Configuration Items stored in the Configuration Management Database.

**Differences to Incident Management:**

Problem Management aims to solve the root causes of problems but Incident Management aims to reduce the impact of a problem or use a workaround to circumvent it. Another difference is that Problem Management tries to prevent problems before they arise working proactive and preventive, while Incident Management takes reactive actions after a problem has occurred.

## 5.4 Change Management

The function of Change Management is to assure that required changes are well prepared and implemented with respect to the process goals. The mission of Change Management is to evaluate the possible effects of a change on existing services, to create a schedule for the change and to ensure that the change is carried out in a time and cost effective manner. Each Request for Change (RfC) has to be authorised by the Change Advisory Board (CAB). The CAB is a council of experts which has to decide, whether an RfC will be rejected or implemented. After a change has been performed the Change Management has to verify that the change had the desired effect and didn't have negative side effects.

There are several reasons which can require a change in a service necessary:

- Solving a problem reported by Problem Management
- As a reaction to a customer complaint
- Installation or upgrade of system components
- Changes in the business requirements of customers
- New or changed laws
- Introduction of new products or services

Change Management should be active during all stages of development. It should create and handle change requests, plan changes and test the changes after they have been implemented to assure a high quality of the product and to reduce the number of error caused by a Change Request (CR). The sequence of changes applied to a service should be well documented by the Change Management.
Change Management and Configuration Management rely heavily on each other. The Change Management needs information about existing services, provided by the Configuration Management, to estimate the impact of a change on the Configuration Item itself and the infrastructure surrounding it. On the other hand the Configuration Management relies on the up to date information about Configuration Items provided by the Change Management. Each RfC has to be carried out in the CMDB and so Change Management helps to keep the database up to date.

## 5.5 Configuration Management

Configuration Management is an important source of information for the Service Support and Service Delivery processes. Developers as well as service providers and management staff benefit of a Configuration Management in a company. It provides up to date information about any resources necessary to provide a specific service and allows control of any IT property owned.
The Configuration Management is also responsible for keeping the data, stored in the Configuration Management Database, up to date. The accuracy of the information stored is vital for other ITIL processes, like the Change Management, which uses the CMDB as data source. Periodic audits and reviews of the Configuration Items should be scheduled.
A more detailed description of the Configuration Management process will be given in part III.

## 5.6 Release Management

One or more changes to an IT service are combined in a *release*. Various dependencies among different services and their software and hardware requirements lead together with other functional requirements to a new release of a service. Release Management ensures the successful planning, implementation and installation of different versions of an IT service. Release Management focuses on the production environment of services using defined processes and checks to ensure a successful deployment of a service.
Before a release is approved for production use it must be validated and the deployment to the production environment will be observed by the Release Management group.
The Release Management is responsible for the Definitive Software Library (DSL) and the Definitive Hardware Store (DHS) which are independent parts of the CMDB. Both, the DSL and the DHS, are repositories for either physical media or hardware parts. The metadata, which describes the physical entities, can be stored in the CMDB.
The DSL contains all physical master copies of software which is in live use. Also authenticity and license documentation should be included in the DSL. Each release should be well documented and added to the DSL. [oGC03] Backup copies of the DSL should be kept in an external location to prevent data loss.
Furthermore the Release Management is responsible for the DHS. The DHS contains certified backup hardware for systems which are in production use. All details about the hardware in the DHS has to be added to the CMDB, this includes hardware specifications, installation instructions as well as configuration details.

There are three different types of releases:

- **Delta release:**

  Only includes new or changed components since the last release. Delta releases usually require only a limited effort of implementation and testing.

- **Package release:**

  In a package release a bigger amount of changes is summarised to a package which is especially important if a release has an impact on other services. Package releases are commonly used for new installations of a service or major version upgrades.

- **Emergency release:**

  Emergency releases are required to solve critical or high priority problems and should be used with care because they increase the probability of errors. Emergency releases are performed outside any release cycle and should therefore only be done with authorisation of a change manager.

# 6 IT Service Delivery

IT Service Delivery deals with processes necessary to plan services on a medium- or long term scale [oGC05a]. Like the Service Support process it has several subprocesses. This chapter will give details about each subprocess of the Service Delivery process.

## 6.1 Service Level Management

Service Level Management (SLM) is a central function to control the quality of an IT service. Its main duty is to negotiate Service Level Agreements with customers and to ensure that the quality of a service is within the agreed boundaries. If the quality of a service is about to drop below the agreed level, Service Level Management is responsible for taking appropriate actions. Furthermore SLM is responsible for documenting any Service Level Requirements (SLR) which should be done in an early phase of development to guarantee that the resulting service will meet the requirements of a customer.
The implementation of Service Level Management in a company is important to make the value of maintenance costs measureable and to ensure that the Financial Management can set proper fees for a certain level of service.

## 6.2 Financial Management

Financial- or Cost Management is responsible for determining the costs of a specific service and for charging consumed services to customers as well as for creating invoices. Financial Management helps to make fees transparent and comprehensible. Knowing the costs of a service is important to make financial success planable. [Pla05], [fSidI]

|  | **Cost determination** | **Charging** |
|---|---|---|
| Annually planning | Determination of common fees for most important IT services | Creation of an allocation base for each service and preparation of a price list |
| Monthly tasks | Monitoring and target/actual comparison | Creation of invoices |

Table 1: Financial Management

Financial Management itself has three sub processes:

- **Budgeting:**

  Budgeting is responsible for planning future expenses and predicting revenues. It also allows the comparison of predicted and actual costs or revenues to improve future predictions.

- **IT Accounting:**

  Accounting is responsible for performing an analysis which service or which department is responsible for which costs. The results of this analysis may help to find areas where costs can be reduced.

- **Charging:**

  This process is responsible for charging customers for the services they have actually consumed.

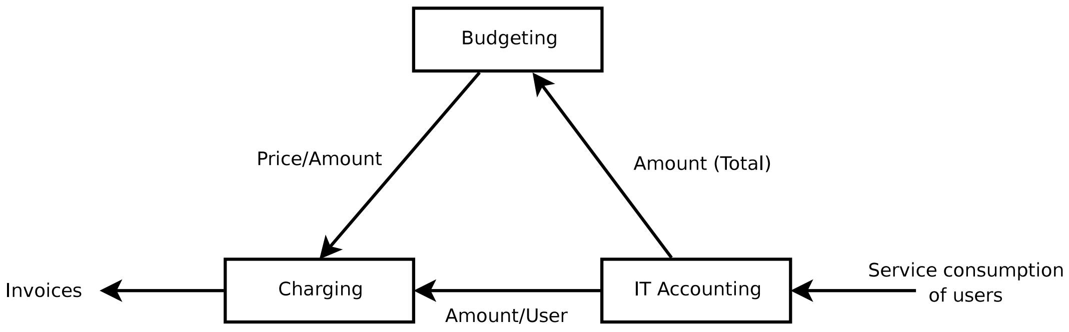


Figure 2: Financial Management

Figure 2 shows the relations among the three sub processes. The *IT Accounting* process keeps track of the consumed services for each user. The *Charging* process takes the amount of consumed services for each user and the prices for the services, provided by the *Budgeting* process, and creates an invoice for each user. The *Budgeting* process uses statistical forecasts of expected total consumption and defines prices for the services provided.

## 6.3 Capacity Management

Capacity Management ensures that customer requirements on throughput, response time and overall performance are met. On introduction of new services or changes of existing services Capacity Management is responsible for providing appropriate resources in time. By balancing load on existing resources, Capacity Management reduces the waste of available capacities. Apart from monitoring the current load, Capacity Management is responsible for creating reports representing current and average workload and forecasts which capacities will be needed in the future to make investments into new resources planable.
Capacity Management heavily relies on statistical methods to create forecasts of the workload. The prediction of required capacities, which will be needed in the future, is based on statistical analysis of the average workload and important to keep response times below a certain threshold defined in an SLA.

## 6.4 Availability Management

A high availability is needed to ensure that customers can access a service whenever they need it. But to achieve a high availability means that the failure rate has to be as low as possible and malfunctions have to be eliminated as soon as possible.
Availability Management is responsible for:

- **Reliability:**

  High reliability is the outcome of a high reliability of each part of the IT infrastructure, the flexability of a service provider and the quality of error prevention actions. Reliability is the ability of a system to maintain its functions during normal circumstances as well as hostile or unexpected circumstances.

- **Maintainability:**

  Maintainability is the ability to keep a service available, which includes following tasks:

  - Error prevention
  - Error recognition (Incidents)
  - Error analysis (Problems)
  - Error removal (Known problems)
  - Restore the service

- **Serviceability:**

  Serviceability are the benefits external service providers and suppliers guarantee to provide for a specific part of the IT infrastructure.

## 6.5 Service Continuity Management

Continuity Management is responsible for analysing different types of threats and catastrophes and for working out counter measures to minimise the impact on the IT infrastructure and/or data loss. Risk management techniques are used to estimate the probability and either to ignore a threat or to document actions which should be taken in case of emergency.

# Part III
# Configuration Management

The Configuration Management process has a very important role in the ITIL framework. It is the main source of information for other processes of Service Support and Service Delivery. The relations of the Configuration Management process are presented in figure 3. Change Management is responsible for validating and approving all changes. Each change performed needs to be documented in a Change Request. It is possible to integrate the Change Management process into Configuration Management. ITIL recommends that the Configuration Management at least keeps track of all changes and helps to evaluate the possible consequences of a change. The information stored in the CMDB should be accessible by all other Service Support processes. Incident-, Problem- and Release Management use the Configuration Management Database to retrieve information about Configuration Items and to store information about new incidents, problems and releases in it.
The Service Delivery processes also use the Configuration Management. The Service Level Management uses the CMDB to identify all Configuration Items, which are affected by a Service Level Agreement (SLA), and link the contract details to them. The Financial Management needs information about running services to charge customers accordingly especially if an SLA exists. Finally Continuity- and Availability Management need to identify Configuration Items which should be included in their analysis.

## 7 Process goals

Like the ITIL framework itself, Configuration Management aims to improve quality of IT services by providing a logical view of the IT infrastructure. The information provided by the Configuration Management is used by other ITIL processes like Incident Management or Release Management.
After the implementation of a Configuration Management in an organisation there should be defined policies and procedures for the identification of Configuration Items and how to perform changes in the CMDB. Security policies define who is allowed to perform which changes to the data recorded in the Configuration Management Database. Periodic audits and reviews, which ensure the accuracy of the information held in the database, are scheduled and held. [Gui]

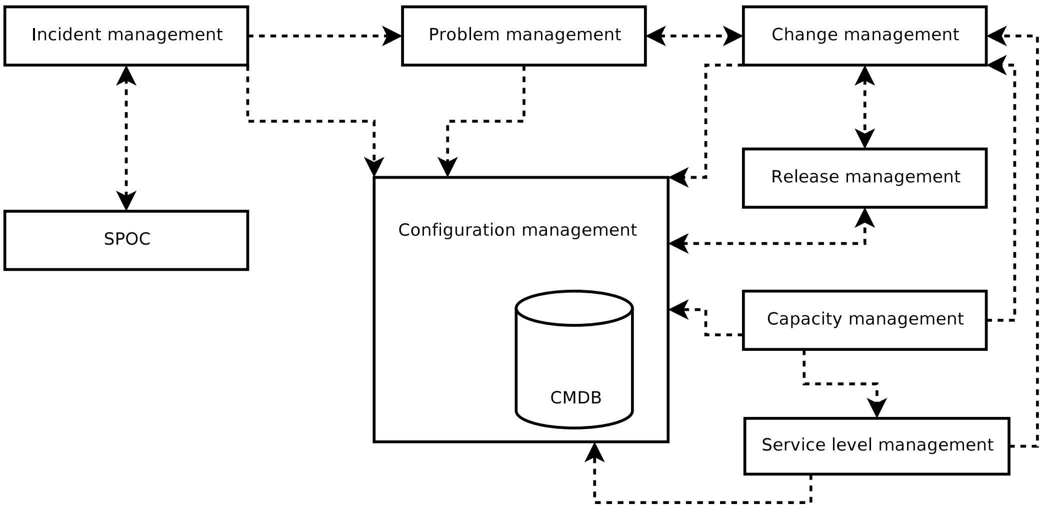


Figure 3: Relations of Configuration Management in ITIL

Configuration Management supports the Release Management by providing an overview where a specific release is used and which problems could arise by changing the release used by a Configuration Item (CI). Problem- and Incident Management use the CMDB to keep track of problems and provide up to date information about services for the customers. Implementing a Configuration Management allows the generation of problem and incident statistics and provides information needed by the management of the company.

# 8 Activities

The Configuration Management process performs the following activities to ensure the quality and consistency of the information stored in the Configuration Management Database:

## 8.1 Planning

First of all the scope, purpose, objectives, priorities, policies and procedures of the Configuration Management process have to be defined. A common labelling convention has to be designed and enforced. In this phase the types and attributes of Configuration Items have to be defined. This step is very important to keep the overhead of data maintenance as low as possible.

## 8.2 Identification

All Configuration Items (CI) of the IT infrastructure, necessary to provide a service, must be identified, labelled and added to the CMDB. For each type of resource the depth of documentation must be defined. For each attribute stored for any CI, a valid reason has to exist to legitimate the effort of gathering and maintaining the information hold in the database. The grade of detail, used for the Configuration Items in the CMDB, is very important for the success of a Configuration Management implementation. A too high grade of detail will increase the effort and costs needed to maintain the CMDB. A too low grade of detail will prevent an effective workflow because of missing data. Adding missing attributes to the CMDB can be time consuming and would produce high costs once a Configuration Management system is in productive use.

## 8.3 Control

To guarantee the quality of any information stored in the CMDB changes should only be possible with authorisation of the responsible department. All changes to the IT infrastructure are subject of the Change Management.

## 8.4 Status accounting

All changes to components should be documented and it should be possible to view or restore old versions of an entry and its relations to other entries. The process of restoring an old version of a Configuration Item can lead to possible inconsistency. See chapter 12.1.2 for more information on that topic. It should be traceable when a change has been made and who is responsible for it. Reports should provide information about the present and previous states of a Configuration Item.

## 8.5 Verification

Periodic verifications of the data stored in the CMDB assure the accuracy of the information provided. The physical existence and all attributes of the Configuration Items should be proved to be correct [Mar] and any changes performed have been applied according to active policies.

## 8.6 CMDB Backup and maintenance

Periodic backups of the database should be taken. The frequency of the backups and how long they are archived depends on how large the IT infras-

tructure is and how often changes to the database take place. A copy of the backup should be kept at an external location to avoid a total loss in case of emergency.
The CMDB should only contain records about Configuration Items which actually exist. If the database is used over a longer period of time redundant and outdated information may be present in the database. Regular checks should be performed to delete unneeded and outdated information from the database.

# 9 Configuration Management Database

The Configuration Management Database represents a logical model of the IT infrastructure. Each service and each hardware asset is recorded in the database and called Configuration Item (CI) in the terms of the ITIL library. Each CI has a type, a set of attributes and a set of relations to other Configuration Items [BGSS]. The CMDB also contains available documentation for the Configuration Items which may be present as binary files.
If possible, parts of the data gathering should be automated to reduce the work which is needed to update the information. Interfaces to service discovery and monitoring tools could be developed.

# 10 Benefits and possible problems

A Configuration Management offers a lot of possibilities and can improve the quality of the services provided but also problems can arise while implementing or using a Configuration Management. [oGC05b]

## 10.1 Benefits

- **Providing information about Configuration Items:**

  A Configuration Management provides up to date information about services and hardware which is used by other processes of the ITIL framework. Relations between the Configuration Items can be found as well as related documentation.

- **Control over valuables:**

  The CMDB represents the expected state of the present inventory. Configuration Management supports the IT management by providing responsibilities in case of theft and allows a comparison of the inventory present in the CMDB and the actual state.

- **Reduces the use of not authorised software:**

  All releases which are used to provide services are documented in the CMDB. Unauthorised software increases the complexity and support costs. Keeping track of releases reduces the rate of errors and helps to improve quality.

- **Helps to observe the law:**

  The Configuration Management keeps track of all software used in a company. Configuration Items which are found during an audit or because of a support request may have been operated illegally. Software which is not licensed can be found and dealt with more easily.

- **Allows problem forecasts and statistics:**

  By analysing the amount of problems which arise from a specific Configuration Item type or with software or hardware of a supplier statistics can be created which help to improve the quality of IT services. This analysis helps to prevent problems before they encounter.

## 10.2 Possible problems

- **Wrong level of detail:**

  The information which is stored in the CMDB and the attributes of the Configuration Items may be defined at a wrong level of detail. Too much information will increase the work which is needed to maintain the information. If not enough information is stored other processes will not be able to retrieve information they need. Furthermore it will not be possible to take full advantage of the control possibilities of a Configuration Management.

- **Inadequate analysis or design:**

  If the system and implementation are not prepared well the resulting system may not satisfy the needs of a company and thus either not be accepted by employees or useless for productive use.

- **Too tight time schedule:**

  The Configuration Management might be seen as a bottleneck if the IT management doesn't consider the time which is needed by the Configuration Management to perform its tasks. While planning changes or releases, the time needed for a Configuration Management should be taken into account.

- **The Configuration Management is ignored by individuals or groups:**

  Employees may see the process of a Configuration Management as too bureaucratic or strict and decide to pass it over to avoid the time and effort which is needed. This problem is difficult to solve especially if individuals refuse to accept the newly introduced workflow. The management should try to motivate their personnel to use the Configuration Management by pointing out the benefits of the process.

- **The chosen tools are not flexible enough:**

  The Configuration Management system, which has been chosen, may not be flexible enough to support new requirements or Configuration Item types whenever the business model of the company changes or new services are introduced.

- **Standalone Configuration Management:**

  If the Configuration Management is introduced without the support of Change- and Release Management, it is not as effective as intended and it will not be able to provide the expected benefits.

Part IV

# Requirements analysis

In a company implementing the ITIL framework the Configuration Management has a central role in the Service Support and Service Delivery processes. It is responsible for providing an up to date representation of the IT infrastructure with all its constraints together with all required and useful information. Depending on which services the company provides different information is required and a lot of different types of visualisation may be useful to represent the data needed in a certain process or situation. Also each ITIL process needs different data from the CMDB.
The Configuration Items and their attributes stored in the CMDB have to be well chosen to fit the requirements of the company, so the information provided will be useful for a user of the Configuration Management. Too many attributes will increase the work which is needed to maintain the data and keep it up to date. Choosing the wrong or too few attributes will reduce the benefits of a Configuration Management.
This part will analyse which functions a Configuration Management system has to provide and which data needs to be stored in the CMDB.

# 11 Requirements of other ITIL processes

Each subprocess of the Service Support and Service Delivery processes has different requirements upon the Configuration Management. This includes the Configuration Items which are needed and also their attributes. Apart from stored data, which a process may need, Configuration Management has to perform different tasks which a Configuration Management system has to provide different functions [Jäg05] for.

## 11.1 Incident Management

As the Incident Management team at the Service Desk is the Single Point Of Contact for customers, the Configuration Management has to provide fast access to all information about a customer, which services he is accessing and any known problems associated with those services. A collection of frequently asked questions, known problems and related solutions should be available about each service to solve common problems quickly. The collection of known problems and associated solutions should be easily searchable.

Furthermore user manuals for each service should be available.

**Required data:**

- Stored incidents
- Customers information
- General information about services
- Basic configuration details needed to access certain services
- Contact information for second level support
- Solutions or workarounds for common or known problems
- User manuals
- Information about current and planned downtimes
- State information (actual-, expected- and last measured state)

**Required functions:**

- *Incident description:*

  When an incident is reported all information required to reproduce the malfunction has to be collected. The name and contact information of all involved people should be included in the incident description.

- *Incident classification:*

  After an incident has been reported the ITIL process model defines that the incident has to be classified. It should be possible to set the priority of an incident.

- *Search for incidents:*

  User should be able to find all current and past incidents which are equal or similar to the search parameters. It should be possible to find all incidents for a specific Configuration Item.

- *Link incidents:*

  Whenever two different incident reports describe the same malfunction of the same Configuration Item, the incident reports should be linked together.

- *Incident course documentation:*

  All actions taken to eliminate a failure should be documented and linked to the incident report. The Service Desk staff should be able the give information to the users about which actions have been taken to solve a problem and to hand out information about the current progress.

- *Incident statistic:*

  It should be possible to generate a list of incidents which occurred during a certain period and to generate this report automatically.

## 11.2 Problem Management

As Problem Management is responsible for finding the cause of errors and malfunctions it has to be able to access detailed technical documentation about a service and which connections to other services exist. For advanced knowledge about a service, contact information for known experts and/or developers has to be available to arrange a team of specialists when needed. If physical access to a hardware resource is needed to solve a problem it is essential to know the exact location of the resource. Therefore detailed information about the whereabouts of a hardware resource needs to be available. In case of hardware errors the period of guarantee and any associated service contracts for a hardware resource have to be available. Contact information to manufacturers, vendors or service contract partners of a resource have to be available, too.

**Required data:**

- Detailed technical documentation
- Connections to other services
- Information about system administrators, experts and/or developers
- Physical location and connections of a resource
- Hardware specifications
- Warranty periods and service contracts
- Contact information to manufacturers, vendors and responsible service contract partners

- Location of available backup hardware

**Required functions:**

- *Problem description:*

  All information related to the problem and persons concerned by the problem should be documented. The problem description should include which sources of information have been used during the identification of the problem.

- *Problem analysis:*

  To simplify the analysis of a problem all information, which could be useful, should be provided. This includes stored incident reports for a Configuration Item, hardware and software specifications as well as all Configuration Items which are related to the current resource.

- *Problem classification:*

  A priority state should be assigned to the problem. An analysis which risks arise with the problem and which benefits can be expected when the problem is solved.

- *Assign problems:*

  A problem should be assigned to a person who takes responsibility for the coordination and the planning of a solution of the problem.

- *Problem course documentation:*

  All actions, taken to solve a problem, have to be filed and all new information, which is gathered while the problem has not been solved, has to be added to the problem record.

- *Problem solutions:*

  All possible solutions for a problem have to be collected and documented. The software should allow the discussion about different solutions. Of all proposed solutions one will be elected and marked as the final solution.

## 11.3 Change Management

Change Management needs to plan changes to the IT infrastructure and running services while ensuring that modifications are performed in a controlled manner. A list of RfCs has to be available. Each RfC should have a priority, a state of progress and an associated schedule. As the mapping of changes to the environment to the CMDB takes time and effort the user interface needs to be well designed and easy to handle. A well designed user interface will reduce the time needed to take over changes to the IT infrastructure into the CMDB.

**Required data:**

- List of change requests
- Fallback instructions
- Service specifications
- Ability to perform changes to the representation of the IT infrastructure in the CMDB
- Contact information of the service managers and administrators

**Required functions:**

- *Accept a RfC:*

  A request for change has to be audited by the Change Management department. Only if the RfC is accepted the demanded changes will be performed. The software should prevent a further processing of a rejected RfC. If an RfC is accepted, it should be assigned to a coordinator who is responsible for further actions during the handling of the RfC.

- *Classify a RfC:*

  The change coordinator has to assign the RfC a priority and to describe the possible effects which the change involves.

- *Plan changes:*

  The actions required to perform the demanded changes have to be planned. It should be possible to define milestones which have to be reached by a certain date. All resources, which are needed to perform the changes, have to be listed.

- *Progress information and documentation:*

  While the planned actions for executing the RfC are being performed, the state of the RfC should be updated regularly. The progress of the actions has to be documented.

- *Search for an RfC:*

  It should be possible to search all stored requests for change by different parameters like the affected Configuration Item, date or assigned coordinator.

## 11.4 Release Management

Release Management works together with Change- and Configuration Management to keep the CMDB up to date. It uses two independent parts of the CMDB, the *Definitive Software Library (DSL)* and the *Definitive Hardware Store (DHS)* (See chapter 5.6). In the DSL the contents of a release is stored and documented and the DHS holds backup hardware for critical components. Both, the DSL and the DHS, should either be located externally or backups should be stored at an external location.

**Required data:**

- Ability to locate available releases
- State of a release (i.e. *Testing* or *Stable*)
- Hardware and software dependencies of a release
- List of resources where a specific release is in productive use
- Installation instructions
- Configuration details

**Required functions:**

- *Release administration:*

  Add new releases of a service or change its attributes. Upgrade services to a new release and associate the new release version with the Configuration Item.

- *Release documentation:*

  While a release is in development, the current state and progress has to be updated regularly. All problems, related to a specific release, have to be documented and linked to the release version. The release documentation has to include the reasons why the release is required.

- *Release history:*

  Older release versions of a service should be available as long as needed. A version can be marked as deprecated to prevent further usage.

## 11.5 Capacity Management

Capacity Management needs to be able to gather data about the current and average workload of the Configuration Items where appropriate. The data will be used to create reports, statistics and forecasts to make available and needed resources planable.

**Required data:**

- Current and average workload as well as statistical forecasts of workload
- Minimum performance and throughput defined in a SLA
- Hardware specifications to estimate expected performance

**Required functions:**

- *Workload gathering:*

  Check Configuration Items for their current workload to generate statistics. Any shortages should be reported to allow further investigation.

- *Workload history:*

  Each time the workload is measured the results should be kept in history to allow the generation of long time statistics which show the load of a system in a defineable timeframe.

# 12 Common Requirements

Not only other ITIL processes like Incident- or Problem Management have demands on a Configuration Management system, also the Configuration Management process itself defines requirements which have to be met. One of these requirements is a data history for Configuration Items which is needed to make changes to a CI traceable. [oGC05b] An access control will be needed for security reasons and to create different roles which are needed to ensure that changes are only performed by authorised staff. A requirement, which emerged from the analysis of the ITIL processes, is that binary data will have to be stored in the Configuration Management Database to provide documentation and user manuals for the Configuration Items. Also the performance of the system has to be taken into account. Short response times are very important for the Service Desk to access the information which is needed to handle a service request as fast as possible.

## 12.1 History

To trace different versions and changes of Configuration Items a history of changes has to be kept. For each operation on a Configuration Item the time the operation was performed, should be stored as well as the user, who performed a specific action.

### 12.1.1 Data models

Technically different methods can be used to store history data in a database:

- **History in the same table:**

  History data is stored in the same table as most recent data. This method has the drawback that in case of many changes performed history data grows fast. *SELECT* statements to the table may be slower due to the large amount of history.

- **History in another table:**

  Another table, a history table, is used to store history data. This will keep the number of rows in the primary table, used for up to date data, smaller but has the drawback that, on an *UPDATE* or *DELETE* statement, the row has to be moved from the primary table to the history table.

Both methods may be used in a Configuration Management system. It depends on the performance requirements or the number of expected *SELECT*, *UPDATE* or *DELETE* operations which method will be used. Also the number of rows kept in history can be limited to a certain amount to keep the size of history data smaller. In general, the time how long history data is kept, depends on the requirements of the company using the Configuration Management system.

### 12.1.2 Consistency

Whenever the possibility to restore older versions of a Configuration Item and its connections to other Configuration Items is needed, there is a possibility that the resulting data is inconsistent. The old version of the CI, which needs to be restored, may reference to Configuration Items which no longer exist or connections between those two types of Configuration Items are no longer allowed.
The Configuration Management system has to prevent potential inconsistency by either preventing actions that would lead to an inconsistent state or taking appropriate actions to keep the database consistent.

## 12.2 Data storage

The CMDB holds all data which is needed to manage Configuration Items. The Configuration Management Database has to offer a possibility to store binary data, support transactions to ensure the consistency of the data stored and a appropriate backup strategy has to be defined and supported by the database application.

### 12.2.1 Binary data

Binary data can either be stored in a central database, which may be a relational database, or in the operating system's filesystem separated from the database system which is used. Each approach has some benefits and drawbacks:

- **Central database:**

  Binary data is stored in using the Binary Large Object (BLOB) type of relational databases. This can be error prone, increase the size of the database quickly and therefore lead to performance drawbacks. On the other side this makes it much easier to keep the database consistent. To improve performance with blobs they could be stored in a separate table. Also the size of data which can be stored may be limited either by the database system or by physical limits. Most databases try to keep BLOBs, like most other data, in memory, which limits the size of available space due to limited memory. For the Postgre SQL database which was used for the implementation of the prototype, the limitations can be found at [Posb].

- **Hybrid database:**

  Binary files are stored in the operating system's filesystem and only the location of the file is stored as a reference in the database. This keeps the primary database smaller but makes it much more difficult to keep the data consistent. For each row inserted or deleted a filesystem operation will have to be executed to store or delete the file. These two actions will have to be executed as transaction to ensure consistency. This approach will allow a physical distribution between database server and file server as the files could be stored on a remote filesystem.

Another limiting factor for the size of binary objects is the amount of memory available for the Java Virtual Machine (JVM). To store the data in the database or a file it has to be kept in memory first, represented by a byte array in terms of Java. Therefore the amount of available heap space for the JVM limits the size of binary objects which can be stored. The size of the heap space can be changed by command line parameters (See chapter 16.2) to the Java Virtual Machine but is also limited by the amount of physical and virtual memory available to the operating system.
The limits of the JVM heap space could be circumvented with some effort by either using a streaming solution to handle binary data or extending the JDBC database driver.

### 12.2.2 Consistency

For a functional and efficient Configuration Management it is essential to keep the CMDB consistent. The application has to assure that only valid and consistent data is accepted and stored to the database at first place. The database has to support transactions and take care of reference integrity. But not all constraints can be checked by a relational database. For constraints which can't be mapped to a database layout and binary data in a hybrid database it is more difficult to keep the database consistent. Therefore tools should be provided to perform consistency checks. The consistency check should first find all errors and then try to repair errors automatically. If the automated repair fails, the error needs to be escalated and the responsible administrator needs to be consulted for further actions.

### 12.2.3 Backup

Regular backups following a defined backup policy should be standard in every company and strongly recommended by the ITIL framework. Backups of the CMDB will include the data stored in the relational database and in case of a hybrid database the files stored in the filesystem. In any case an incremental backup strategy with occasional full backups is recommended to limit the amount of data transferred and stored. Backups will most likely be performed automatically and can either be performed locally by a backup process or remotely by a backup server:

- **Local backup:**

  A process which is periodically run on the local machine creates a backup of the installation and moves the backup archives either to a persistent media or to a central backup server. This approach distributes the load of taking backups. All local resources like the database and the file system can easily be accessed.

- **Remote backup:**

  A process at a remote backup server requests a SQL dump from the local database and transfers files to create a backup. Most databases support remote SQL dumps thus allowing the backup server to create a backup of the relational database remotely. In case of a hybrid database, the *rsync* [Rsy] application is one possibility to synchronise files remotely. Network filesystems can also be used to access remote files. The backup server can schedule backups on his own and will take the workload of backup creation. The machine, of which a backup is created, is only stressed by disk access and network load emerging from the file transfer to the backup server

The ITIL framework also recommends to keep copies of backups at an external location to ensure that the restoration of the database is possible even after a complete loss of the production environment and the primary backups.

## 12.3 Performance requirements

The Configuration Management system interfaces with various other processes of the ITIL framework like the Incident Management or the Release Management. ITIL processes will request data from the CMDB which is provided by the Configuration Management system. Some requests performed by the i.e. Capacity Management, will be carried out by an automated process which generates reports, and thus have non critical requirements to performance. On the other hand the requests carried out by the Incident Management have critical performance requirements since the main task is providing customer support. The information, which is required by the Service Desk, has to be available in a very short time.

### 12.3.1 High performance requirements

These tasks have high performance requirements, since they are mostly used by the Service Desk and the Incident Management. The response times have to be as low as possible to minimise the time customers have to wait for the requested information.

- State information about services
- Customer information
- Known problems and frequently asked questions
- Detailed information about services
- Relations of the Configuration Item

### 12.3.2 Normal performance requirements

Tasks which have normal performance requirements are not time critical but still have to be carried out fast enough to allow an interactive workflow with the system. The main users of the tasks are Problem Management, Change Management, Release Management and the Configuration Management itself.

- Management of Configuration Items
- Binary content (i.e.: documentation)
- Management of known problems and change requests
- Management of releases

### 12.3.3 Low performance requirements

These tasks may have a response time which doesn't allow interactive work because the amount of data which is processed during a request is either too big or complex. Users should be warned that the operation, they are about to perform, may take a long time to complete.

- Reports which require iteration through all services or Configuration Items of a certain type
- Stock lists

## 12.4 Access control

To restrict access to different areas or tasks an access control is necessary. This is needed for security reasons on the one hand and to enforce an ITIL compliant workflow on a permission basis on the other hand. Users will have to login with their username and password to identify themselves.
For each user of the system the real name and contact information should be stored to associate real persons with their user names. For automatic password recovery an email address should be stored and validated. Each user will be member of a group. Group membership defines a default set of permissions for a user to determine whether the user is allowed to perform an action or the access is denied.
The system should provide a predefined set of groups and permissions which is appropriate for most cases but allow the definition of new groups and their permissions.

## 12.5 Management of Configuration Items

To manage the data hold in the CMDB, an administration interface is needed. The Configuration Items stored in the database and their attributes should be editable. Access to the management interface should be limited by the access control to follow active security policies and prevent unauthorised changes.
As the type of Configuration Items and their attributes, which need to be stored in the CMDB, depend on the type of services a company provides, the Configuration Management system has to offer a high level of flexibility. To offer this flexibility it should be possible to define new types of Configuration Items. For each CI type it should be possible to define new attributes which all Configuration Items of this type have to possess. The system has to ensure that changes to existing CI types do not break the consistency of existing Configuration Items and their attributes.
The CMDB is also responsible for storing the relations among Configuration Items. It should be possible to define links among Configuration Items. Relations of Configuration Items should be possible at any level of detail needed. For each CI type it should be restrictable which other CI types are linkable to Configuration Items having this type.

# 13 Use cases

Users of the Configuration Management system will have to perform certain tasks on a regular basis. These tasks should be easy to perform and follow the ITIL guidelines. The user interface to perform those tasks should be well designed to allow a fast and efficient workflow. In the following section several use cases, which originate from different ITIL processes and represent functions which will be used regularly, will be described.

## 13.1 Configuration Item search

Before any action can be taken or any information is passed out to a customer the Service Desk members have to relate the customer to a Configuration Item. Therefore a powerful and flexible search is needed to find the desired Configuration Item easily. The search function should offer different possibilities to find a Configuration Item. The search interface should be as accurate as possible to minimise the number of results returned.

1. The user enters available information about a Configuration Item.
2. The system provides a list of Configuration Items which match the provided information.

## 13.2 Configuration Item overview

Many requests will require a summary for a certain Configuration Item, providing all state information and shortcuts to commonly used tasks. All contacts, incidents and problems related to the Configuration Item should be accessable. The Configuration Item overview should be the starting point for every action the Service Desk performs.

1. The user searches for a Configuration Item and selects the desired one from the list.
2. The system Configuration Item overview is displayed.

## 13.3 Incident registration

The registration of an incident should be possible from the Configuration Item overview. The form collects all data required to understand the cause of the problem. The incident has to be classified with a certain priority by the Incident Management. The reporting customer of the incident has to be filed and the incident has to be linked to a Configuration Item. If necessary the incident can be linked to other incidents. In a second step the incident has to be assigned to an agent who is responsible for further handling of the incident.

1. The system Configuration Item overview is displayed.
2. The user selects "Report new incident".
3. The Incident registration form is displayed.
4. The user enters all required information provided by the customer.
5. The system saves the incident for further processing.

## 13.4 Problem report creation

Whenever an issue is passed to the Problem Management for further investigation a problem report has to be created. It has to include all information which is available about the problem. All incidents, Configuration Items and persons related to problem have to be linked to the problem report. Known workarounds or possible solutions for the problem have to be attached to it.

1. The user selects "New problem report".
2. The form to enter a new problem report is displayed.
3. The user enters all required information, attaches affected Configuration Items and releases. Related incidents are linked to the problem.

## 13.5 Request for Change

Changes of the configuration should only be performed on behalf of an RfC to follow the guidelines of the ITIL framework. The RfC has to contain a detailed description what has to be done to perform the changes and has to include the reasons for the change. All affected Configuration Items have to be linked to the RfC. Incidents or problems, which may be the root cause for the RfC, should be linked to the RfC.

1. The user selects "New RfC".
2. The form to enter a new RfC is displayed.
3. The user enters all required information, attaches affected Configuration Items. Related incidents or problems are linked to the RfC.

## 13.6 Release search

To keep track of releases used in the current configuration the Release Management needs a possibility to find all Configuration Items which use a specific release. This allows the Release Management to identify all Configuration Items which may need to be upgraded whenever a new release version is available and ready to be deployed into production environment.

1. The user selects a release for which the Configuration Items should be found.

2. The user selects "Find Configuration Items using this release".

3. The system lists all Configuration Items which use the selected release.

Part V

# Implementation

Given the requirements on a Configuration Management system a prototype has been implemented which is intended to prove that the requirements analysis was complete and decisions made concerning design were right. The system provides the ability to acquire the data needed by other ITIL processes and support the workflow proposed by the ITIL framework.

# 14 Design considerations

The basic structure of the system has been given in advance by Dr. Manfred Siegl, but many aspects of the system and implementation details are based on the results of the requirements analysis.

## 14.1 Preset basic conditions and requirements

- Client - Server oriented
- The server is a webservice implemented in Java
- A light weight client as user frontend
- Adobe Flash [Inca] for visualisations communicating directly with the webservice

Apart from these requests the system design will be determined by whatever fits best to meet the requirements discovered during the requirements analysis.

Due to some severe limitations of webservices regarding session handling and push based communication, the server will not be implemented as a webservice. As the HTTP protocol is used by the clients to call functions of the webservice, there is no unsolicited communication from the webservice to the clients possible. This is a major disadvantage because clients would have to use polling to keep their data up to date.
Another disadvantage of webservices is their stateless nature. For each request another HTTP session is created and keeping track of authenticated clients is only possible if the application simulates sessions. Apart from the difficult session handling, TCP connections are established and closed for

each request, which leads to a certain performance overhead. But this performance drawback is neither desired nor needed.

Both limitations interfere with the requirements on the Configuration Management system. Sessions are needed for user authentication and access control. Unsolicited messages are required to provide an up to date state of Configuration Items.

In agreement with Dr. Manfred Siegl the server will be implemented as a standalone application server in Java communicating with Extensible Markup Language (XML) messages with the clients. The former requirement that the server should be implemented as a webservice has been dropped due to their limitations interfering with the requirements of the system. The server is implemented in Java as required.

Note that not all use cases mentioned in chapter 13 have been implemented because the prototype implementation does not include all data needed for the use cases. The prototype implementation focuses on the Configuration Management process alone. Interfaces to other processes are either subject to future work of have only been implemented in a simple fashion.

## 14.2 Database

As database server the PostgreSQL [Posa] database has been chosen, because it meets all requirements (See chapter 12.2) and its license allows corporate usage.
A central approach has been used for the database, which means that binary data will be stored inside the relational database. A central database is easier to handle, consistency can be easily ensured using transactions provided by the database system and the JDBC API supports the handling of binary data.

# 15 System architecture

The system has a distributed nature and has four main actors communicating with each other and providing data for another actor. Figure 4 shows the connections between the actors and which protocol they use to communicate with each other. The following section will describe each actor and its role in the system.

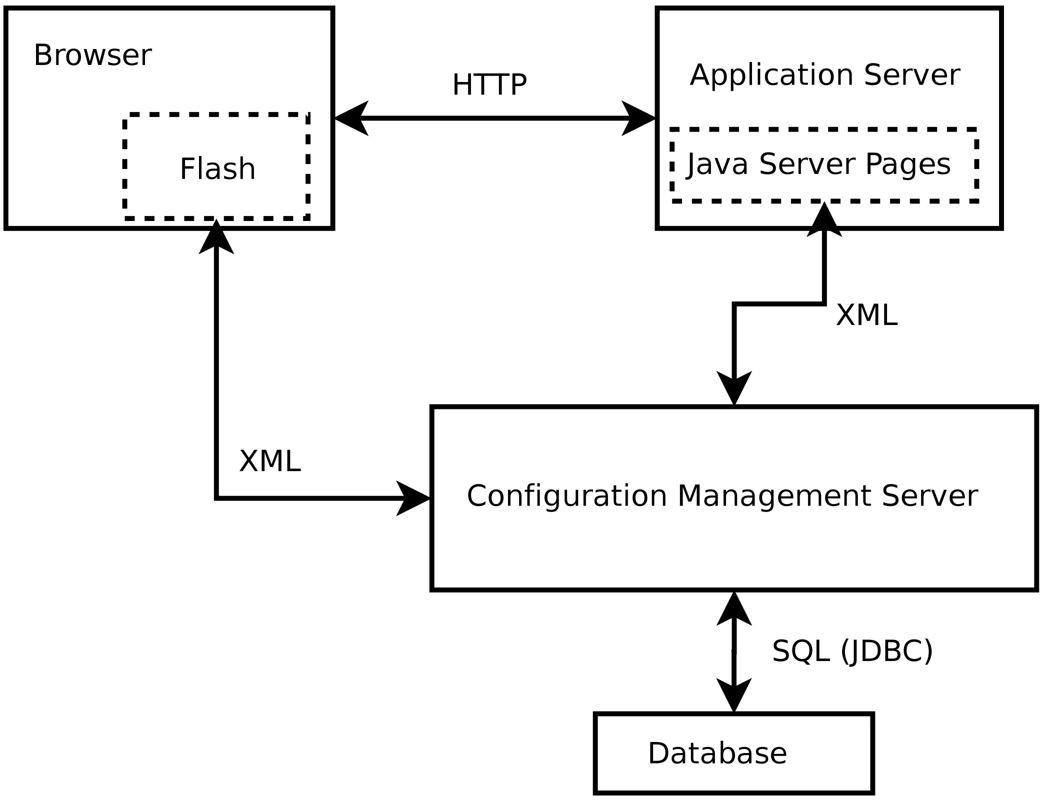


Figure 4: System architecture

## 15.1 Server

The Configuration Management server provides a set of commands the client can call.
The server is the main part of the Configuration Management system because it provides access to all data stored. The underlying relational database is abstracted by the Configuration Management server. It is responsible for data consistency and integrity.
Each client connected to the server can enlist itself to receive notifications whenever data is changed, added or deleted, which allows push based updates on the client side. For each client a session is created which handles requests in its own thread. The session can be terminated by the client by sending a logout request or by the server if the client hasn't sent at least a keep alive message within a certain time. The TCP socket is closed and all active subscriptions are invalidated.

## 15.2 Database

The database, which is accessed by the Configuration Management server, is a Postgre SQL based relational database. It holds all data required by the Configuration Management, which also includes binary data stored in the Configuration Management Database. Figure 5 shows the Entity-Relationship Model (ER) of the relational database.
The database allows the definition of dynamic attributes for the *citem* table. For each *citemtype* additional attributes can be defined within the *attribute* table. The actual values of the attributes for each *citem* are stored in the *attributevalue* table. This allows a very flexible data model which can be changed at runtime. The Configuration Management server is responsible for hiding this data structure from the users and ensuring data consistency. To access the database the Java Database Connectivity (JDBC) API is used.

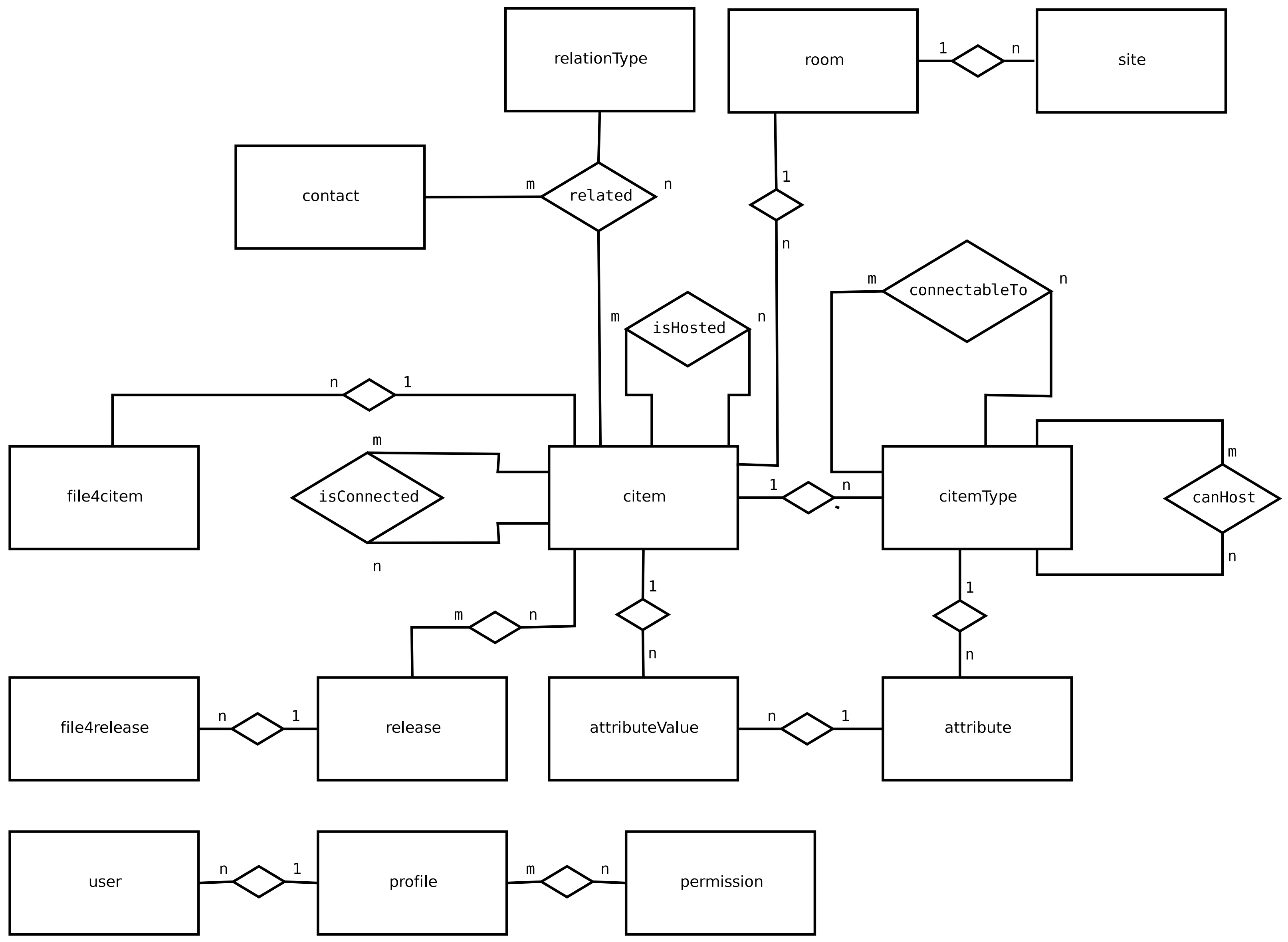


Figure 5: ER of the database

## 15.3 Application server

The Java Server Pages, which are accessed by the client's web browser, need an application server to run. Various application servers exist and are able to host Java Server Pages. The Sun System Application Server [Corc] has been chosen, because it is the reference implementation of an application server provided by the initial developer of Java. All other application servers like Glassfish [Coma], Tomcat [Fou] or JBoss [Comb] should be compatible and able to run the developed Configuration Management system with no or minor changes to the build system or source code. The Configuration Management system has been tested on the application servers Glassfish and Tomcat and the system runs without problems or modifications.

### 15.3.1 Java Server Pages

The Java Server Pages implement the main user interface of the system. Users access them through a web browser. The users have to authenticate themselves before they can access the system. For each session a permanent TCP socket to the Configuration Management server is created. This socket is used for communication with the Configuration Management server using an XML based protocol. All data requested by the user will be fetched from the server.

## 15.4 Client

The user interacts with the system by the means of a web based interface. Java Server Pages (JSP) [Corb] will be used to implement the required light weight client. The Java Server Pages use the Configuration Management server as data source, so no direct database connection will be needed.

### 15.4.1 Adobe Flash

An Adobe Flash application is used to realise permanent and direct communication between the Configuration Management server and the client. Adobe Flash supports XML communication with the *XMLSocket* class [Incb]. The Adobe Flash applications connect directly to the Configuration Management server and subscribe to be notified whenever certain data changes. Changes made by other clients are pushed from the server to the Adobe Flash application and provide immediate feedback of changes for the user. Also this approach is used for status updates of Configuration Items.

# 16 Technical details

This section will describe some aspects of the implementation in detail. Programmatic concepts are presented as well as class diagrams and the XML protocol used for communication.

## 16.1 Relations of Configuration Items

Two Configuration Items can be related to each other. There are two types of relations in the system: *Connections* and *Hosting*. The type of a Configuration Item determines to which other items it can be related to. For each Configuration Item type, a list of Configuration Item types can be defined. Only those Configuration Items, which are of a type contained in that list, can be related to the Configuration Item.
It is in the responsibility of the Configuration Management staff members to decide at which level of detail connections among Configuration Items should be recorded. In reality the connection between i.e. two servers doesn't consist of a single Cat.5 cable. The connection may consist of several cables with or without switches in between. Also network interfaces may not be present in the Configuration Management Database but are mandatory for a network connection in reality. The system may hide such details because of the level of detail which has been chosen.

### 16.1.1 Connections

Connections define bidirectional relations among Configuration Items. If Configuration Item $A$ is connected to Configuration Item $B$ then $B$ is also connected to $A$. Therefore the possible connections, which a certain Configuration Item type defines, are bidirectional too. The connections define a graph with Configuration Items as nodes and the connections as edges. A Configuration Item can have an arbitrary number of connections, but only one connection to each item and it cannot be connected to itself.

### 16.1.2 Hosting

The relation of hosting defines a dependency of one Configuration Item to another. If the item, which hosts other Configuration Items, fails it is almost certain that the hosted items will fail too. This relation is intended for software services running on hardware. The hosting relation defines a directed link from one Configuration Item to another.

## 16.2 Binary data

Binary data can be linked to Configuration Items and releases. To store the binary data in the database the Large Object API [Posc] of the Postgre SQL database driver is used. The maximum size of files, which can be stored, is mainly limited by the amount of memory available for the Java Virtual Machine because the system needs to allocate heap space for the binary data. The amount of memory available for the JVM can be changed by the *-Xmx* command line parameter of the Java Runtime. The Postgre SQL database has its own size limits which can be found at [Posb].
The management of binary data has been kept simple and, for the use in a production system, may require further development.

## 16.3 Implementation details

This section will describe the source code of the implementation using class diagrams and present the programming concepts used during the implementation of the Configuration Management system prototype.
The source code is organised in three parts. The source code for the Configuration Management server, the servlet source code and the common code library.

### 16.3.1 Server

The Configuration Management server has a multithreaded architecture. It communicates with its clients using an XML protocol (See chapter 16.4). The XML messages are exchanged between the server and its clients using TCP connections. The server is stateless, apart from session handling.
The class diagram in figure 6 shows the classes, which are responsible for handling client requests. A *ServerSocket* is listening on the network interface and port specified in the configuration files, which are accessed through the *SettingsManager*. For each connected client a new instance of the *SessionThread* class is created, which takes all client requests. The *SessionManager* has knowledge of all active sessions and is responsible for shutting down timed out sessions. The maximum number of sessions is only limited by the available memory.
All requests received on the socket are handled by the *MessageHandler* and a response is generated. The *MessageHandler* is responsible for checking whether the user has authenticated yet or not. The *MessageHandler* tries to find the responsible handler class for each request. If a handler class can be found, the request is passed to it. Otherwise an exception message is sent

as response. The state diagram in figure 8 shows the different states during message processing.
Handler classes for messages are part of the *table engine* and extend the *AbstractTable* class. The *AbstractTable* processes the messages and automatically generates SQL statements from the request message received. The *AbstractTable* class is capable of processing add, change and list operations but can be extended by child classes to support other operations or custom logic.
If the request can be processed successfully and the request data is valid, a response is sent to the client. Otherwise an exception message is returned.
Clients can also subscribe themselves to receive notifications whenever a certain table changes. The *AbstractTable* class is responsible for the generation of notification messages and sending them to all subscribers.
The class diagram in figure 7 shows the classes of the table engine. Only a subset of the existing child classes of the *AbstractTable* class is shown for readability but in general, for each table in the database, a child class of *AbstractTable* exists.
The concept of the table engine and its automated message processing and SQL statement generation has been used to handle simple requests automatically and increase code reuse and maintainability of the code. The concept allows the programmer to override each message processing method and provides routines to validate request message data.

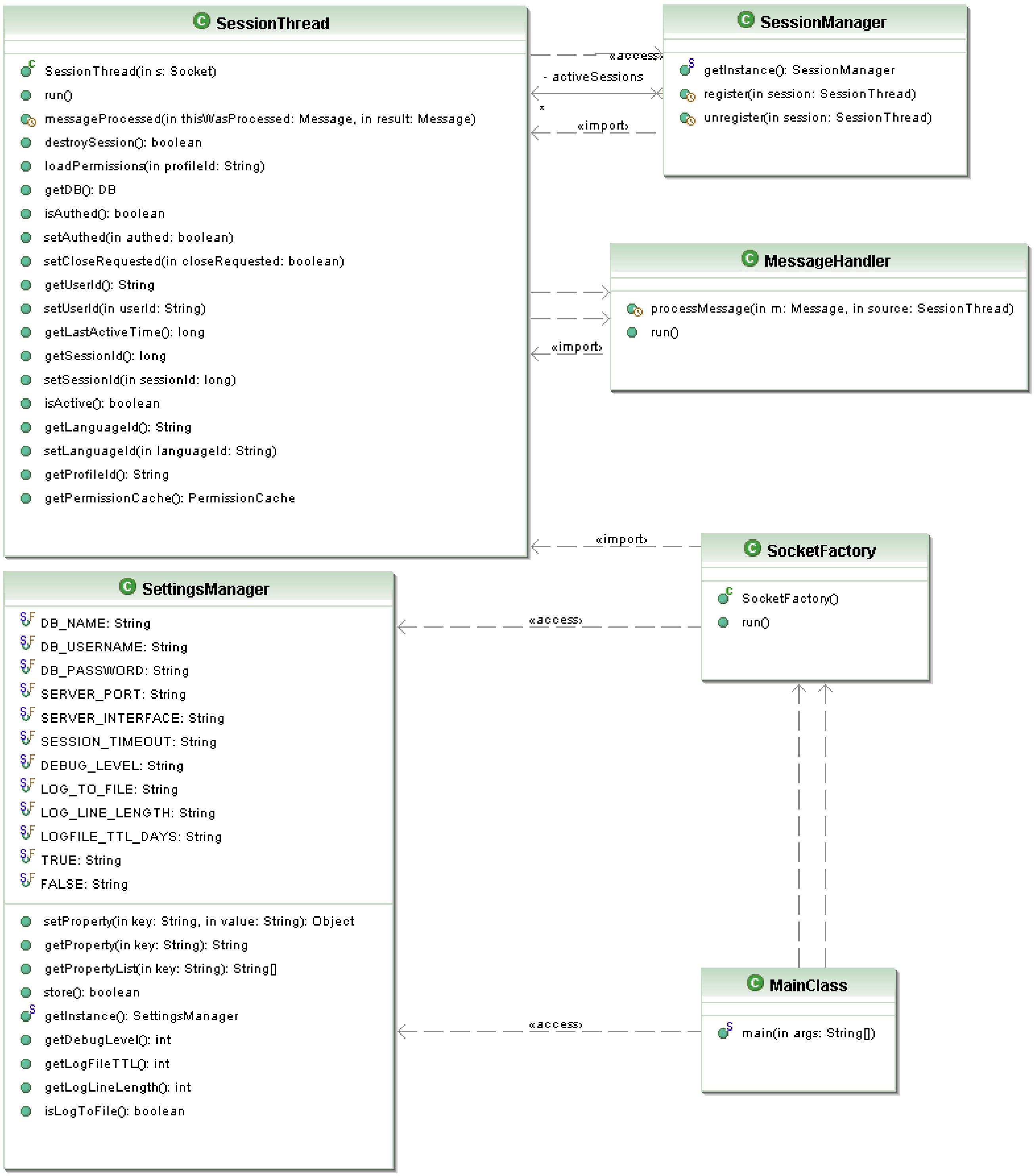

SessionThread
SessionThread(in s: Socket)
run()
messageProcessed(in thisWasProcessed: Message, in result: Message)
destroySession(): boolean
loadPermissions(in profileId: String)
getDB(): DB
isAuthed(): boolean
setAuthed(in authed: boolean)
setCloseRequested(in closeRequested: boolean)
getUserId(): String
setUserId(in userId: String)
getLastActiveTime(): long
getSessionId(): long
setSessionId(in sessionId: long)
isActive(): boolean
getLanguageId(): String
setLanguageId(in languageId: String)
getProfileId(): String
getPermissionCache(): PermissionCache
«access»
- activeSessions
*
«import»
SessionManager
getInstance(): SessionManager
register(in session: SessionThread)
unregister(in session: SessionThread)
MessageHandler
processMessage(in m: Message, in source: SessionThread)
run()
«import»
«import»
SocketFactory
SocketFactory()
run()
«access»
SettingsManager
DB_NAME: String
DB_USERNAME: String
DB_PASSWORD: String
SERVER_PORT: String
SERVER_INTERFACE: String
SESSION_TIMEOUT: String
DEBUG_LEVEL: String
LOG_TO_FILE: String
LOG_LINE_LENGTH: String
LOGFILE_TTL_DAYS: String
TRUE: String
FALSE: String
setProperty(in key: String, in value: String): Object
getProperty(in key: String): String
getPropertyList(in key: String): String[]
store(): boolean
getInstance(): SettingsManager
getDebugLevel(): int
getLogFileTTL(): int
getLogLineLength(): int
isLogToFile(): boolean
«access»
MainClass
main(in args: String[])

Figure 6: Class diagram of the server socket and session classes

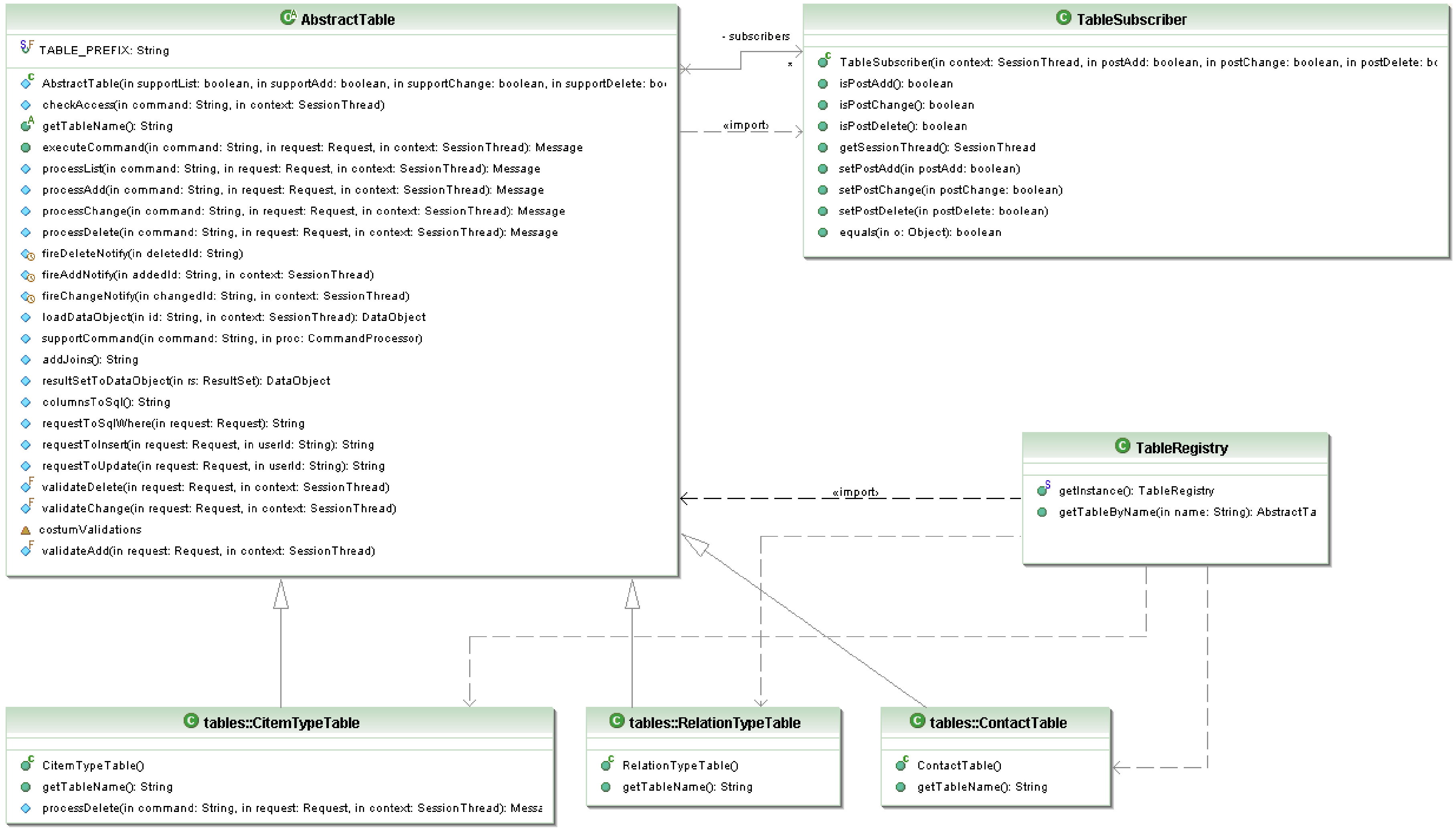

AbstractTable
TABLE_PREFIX: String
AbstractTable(in supportList: boolean, in supportAdd: boolean, in supportChange: boolean, in supportDelete: bo
checkAccess(in command: String, in context: SessionThread)
getTableName(): String
executeCommand(in command: String, in request: Request, in context: SessionThread): Message
processList(in command: String, in request: Request, in context: SessionThread): Message
processAdd(in command: String, in request: Request, in context: SessionThread): Message
processChange(in command: String, in request: Request, in context: SessionThread): Message
processDelete(in command: String, in request: Request, in context: SessionThread): Message
fireDeleteNotify(in deletedId: String)
fireAddNotify(in addedId: String, in context: SessionThread)
fireChangeNotify(in changedId: String, in context: SessionThread)
loadDataObject(in id: String, in context: SessionThread): DataObject
supportCommand(in command: String, in proc: CommandProcessor)
addJoins(): String
resultSetToDataObject(in rs: ResultSet): DataObject
columnsToSql(): String
requestToSqlWhere(in request: Request): String
requestToInsert(in request: Request, in userId: String): String
requestToUpdate(in request: Request, in userId: String): String
validateDelete(in request: Request, in context: SessionThread)
validateChange(in request: Request, in context: SessionThread)
costumValidations
validateAdd(in request: Request, in context: SessionThread)
- subscribers
*
«import»
TableSubscriber
TableSubscriber(in context: SessionThread, in postAdd: boolean, in postChange: boolean, in postDelete: bo
isPostAdd(): boolean
isPostChange(): boolean
isPostDelete(): boolean
getSessionThread(): SessionThread
setPostAdd(in postAdd: boolean)
setPostChange(in postChange: boolean)
setPostDelete(in postDelete: boolean)
equals(in o: Object): boolean
TableRegistry
getInstance(): TableRegistry
getTableByName(in name: String): AbstractTa
«import»
tables::CitemTypeTable
CitemTypeTable()
getTableName(): String
processDelete(in command: String, in request: Request, in context: SessionThread): Messa
tables::RelationTypeTable
RelationTypeTable()
getTableName(): String
tables::ContactTable
ContactTable()
getTableName(): String

Figure 7: Class diagram of the server table engine

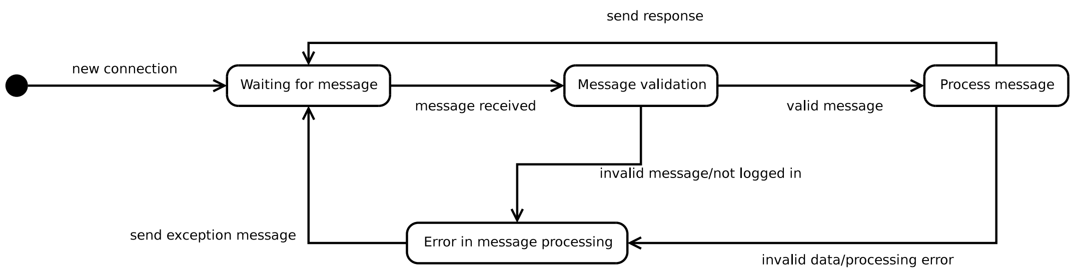


Figure 8: State diagram of the server message handling

### 16.3.2 Servlet

The class diagram in figure 9 shows the basic structure of the classes responsible for generating the user interface and connecting to the Configuration Management server. The *CMServlet* class handles all HTTP requests and chooses the Java Server Page (JSP) responsible for the request. The *Client* class holds all active connections to the Configuration Management server. Each connection is handled by an instance of the *ClientSocket* class.
HTML output is automatically generated by the *SimpleView* view class and its child classes. The *SimpleView* class is capable of generating a HTML user interface for arbitrary data objects. The *BrowseableView*, which extends the *SimpleView* class, adds the functionality to browse the displayed list of data objects, which is useful whenever a large number of objects is displayed. The *SearchableView* enables the user to search for a specific data object and extends the *BrowseableView* class.
The automation of HTML code generation has the advantage that simple standard user interfaces can be handled at a central spot. This increases code reuse and maintainability whenever the HTML code needs to be changed. The automated generation of HTML code has the disadvantage of decreased readability of the source code but overall the advantages of increased maintainability and code reuse outweight this tradeoff. Future additions to the system may only require a single line of code instead of the creation of a whole JSP page. Changes to the HTML code can be done in a central location for all existing views.

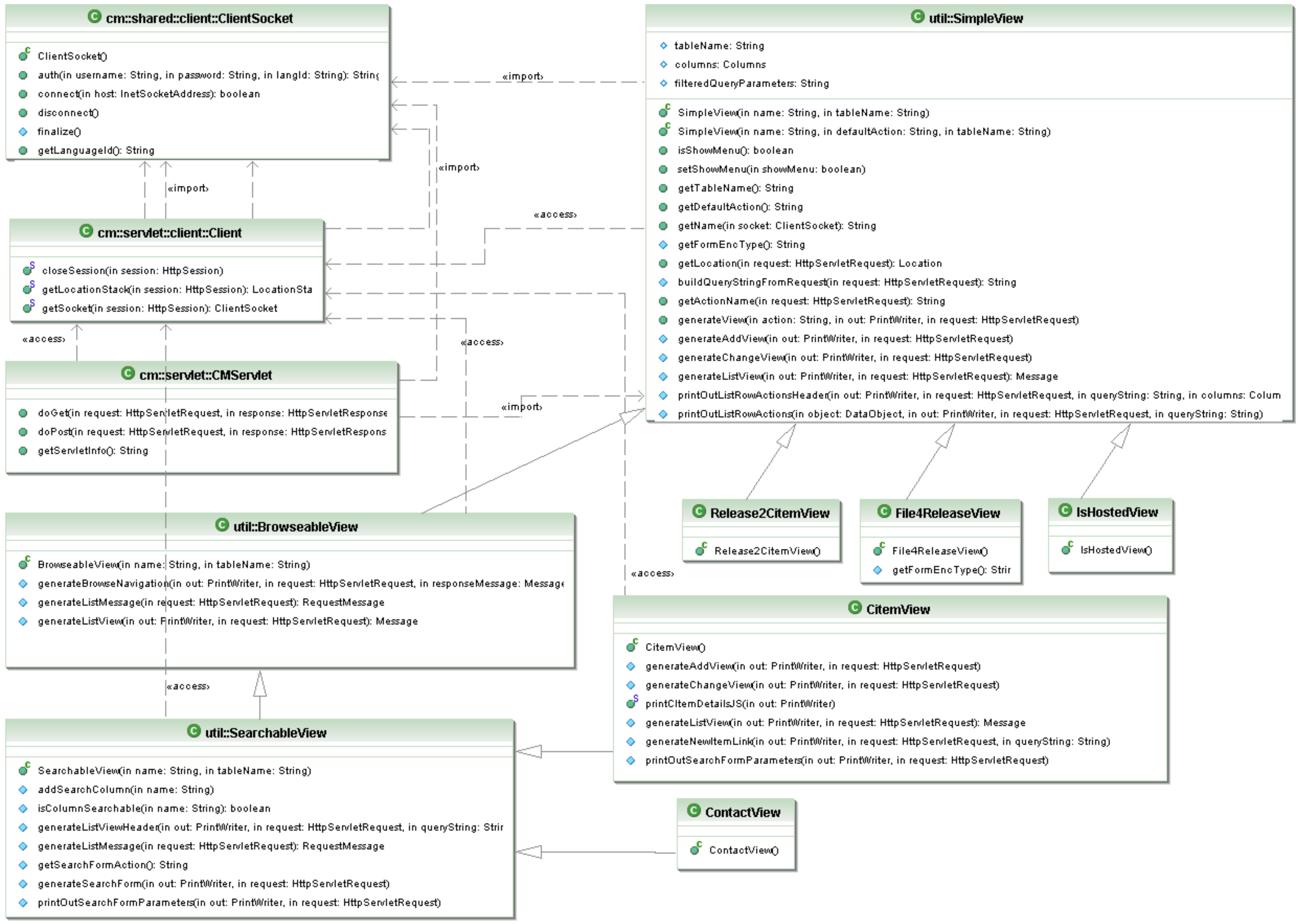

cm::shared::client::ClientSocket
ClientSocket()
auth(in username: String, in password: String, in langId: String): String
connect(in host: InetSocketAddress): boolean
disconnect()
finalize()
getLanguageId(): String
util::SimpleView
tableName: String
columns: Columns
filteredQueryParameters: String
SimpleView(in name: String, in tableName: String)
SimpleView(in name: String, in defaultAction: String, in tableName: String)
isShowMenu(): boolean
setShowMenu(in showMenu: boolean)
getTableName(): String
getDefaultAction(): String
getName(in socket: ClientSocket): String
getFormEncType(): String
getLocation(in request: HttpServletRequest): Location
buildQueryStringFromRequest(in request: HttpServletRequest): String
getActionName(in request: HttpServletRequest): String
generateView(in action: String, in out: PrintWriter, in request: HttpServletRequest)
generateAddView(in out: PrintWriter, in request: HttpServletRequest)
generateChangeView(in out: PrintWriter, in request: HttpServletRequest)
generateListView(in out: PrintWriter, in request: HttpServletRequest): Message
printOutListRowActionsHeader(in out: PrintWriter, in request: HttpServletRequest, in queryString: String, in columns: Colum
printOutListRowActions(in object: DataObject, in out: PrintWriter, in request: HttpServletRequest, in queryString: String)
«import»
«import»
«import»
«access»
cm::servlet::client::Client
closeSession(in session: HttpSession)
getLocationStack(in session: HttpSession): LocationSta
getSocket(in session: HttpSession): ClientSocket
«access»
«access»
cm::servlet::CMServlet
doGet(in request: HttpServletRequest, in response: HttpServletResponse
doPost(in request: HttpServletRequest, in response: HttpServletRespons
getServletInfo(): String
«import»
Release2CitemView
Release2CitemView()
File4ReleaseView
File4ReleaseView()
getFormEncType(): Strir
IsHostedView
IsHostedView()
util::BrowseableView
BrowseableView(in name: String, in tableName: String)
generateBrowseNavigation(in out: PrintWriter, in request: HttpServletRequest, in responseMessage: Message
generateListMessage(in request: HttpServletRequest): RequestMessage
generateListView(in out: PrintWriter, in request: HttpServletRequest): Message
«access»
CitemView
CitemView()
generateAddView(in out: PrintWriter, in request: HttpServletRequest)
generateChangeView(in out: PrintWriter, in request: HttpServletRequest)
printCItemDetailsJS(in out: PrintWriter)
generateListView(in out: PrintWriter, in request: HttpServletRequest): Message
generateNewItemLink(in out: PrintWriter, in request: HttpServletRequest, in queryString: String)
printOutSearchFormParameters(in out: PrintWriter, in request: HttpServletRequest)
«access»
util::SearchableView
SearchableView(in name: String, in tableName: String)
addSearchColumn(in name: String)
isColumnSearchable(in name: String): boolean
generateListViewHeader(in out: PrintWriter, in request: HttpServletRequest, in queryString: Strir
generateListMessage(in request: HttpServletRequest): RequestMessage
getSearchFormAction(): String
generateSearchForm(in out: PrintWriter, in request: HttpServletRequest)
printOutSearchFormParameters(in out: PrintWriter, in request: HttpServletRequest)
ContactView
ContactView()

Figure 9: Class diagram of the servlet view classes

### 16.3.3 Common code library

Both, the Configuration Management server and the servlet, use a common code library. The library contains classes which both, the server and the servlet, use. Apart from utility classes, the common code library contains the implementation for the logging system and the system to access multilingual strings. The library also holds critical classes, which are needed for the XML communication and database definitions.

The class diagram in figure 10 shows the classes, which are used to generate and parse XML messages. The *SAXModelBuilder* class is responsible for message parsing. Each element, which can be part of an XML message the server can process, extends the *XMLElement* class. The outer container of each XML message, the *Message* class, has several child classes, like the *RequestMessage* class, which provides convenience methods for message handling.

The class diagram in figure 11 shows the classes used for column definitions. For each table in the database one of those classes defines, which columns exist and which data type they hold. Furthermore this classes have an important role in the generation of the user interface as well as the generation of SQL statements in the table engine. The column classes provide properties and methods for each datatype defining how the user interface will look like and which syntax is required for the SQL database. Each column class, like the *CitemColumns* class, extends the *Columns* class. The *Columns* class holds a set of *TableColumn* instances. For readability of the class diagram only four column classes have been included in the diagram.

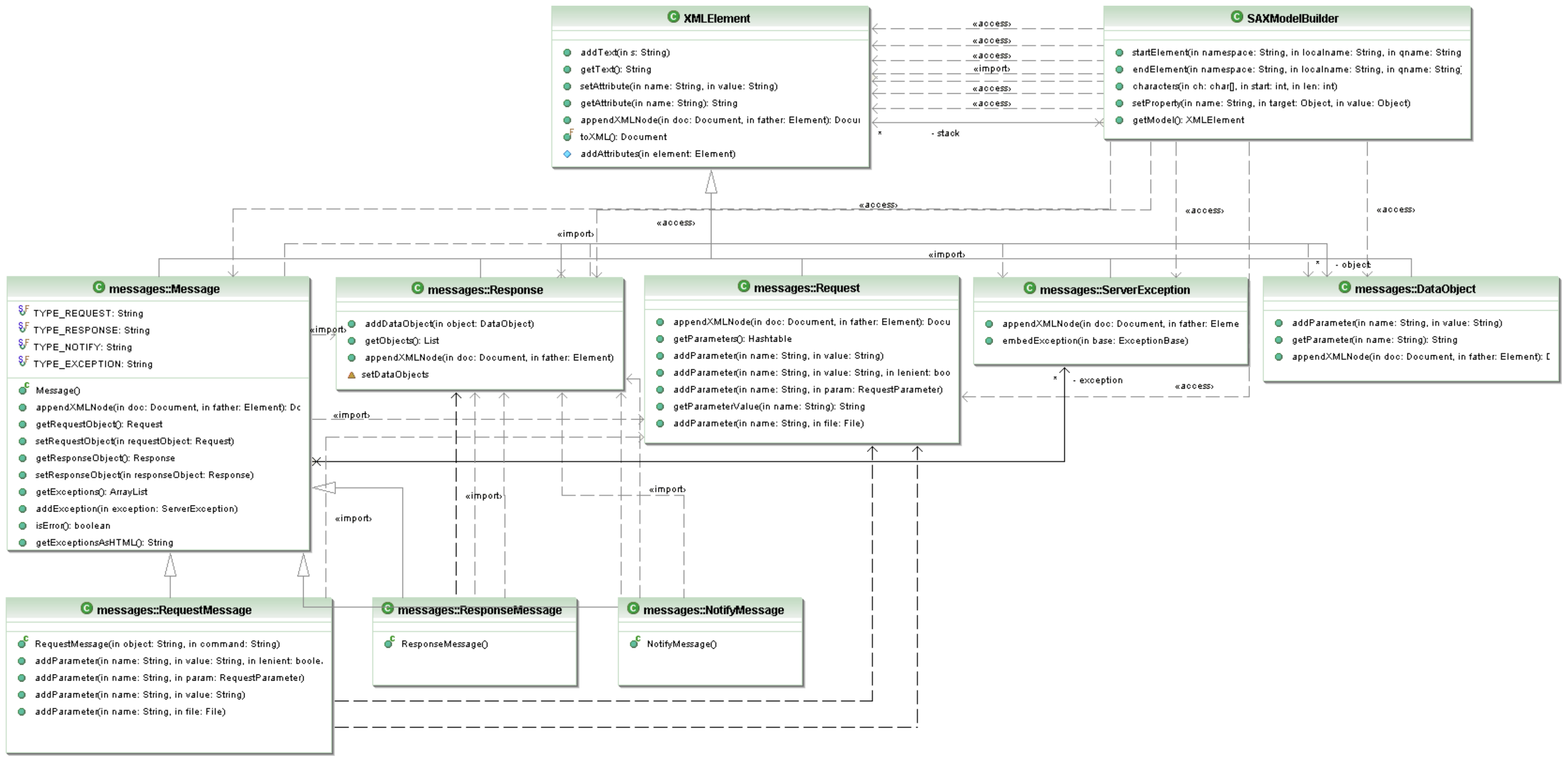

XMLElement
addText(in s: String)
getText(): String
setAttribute(in name: String, in value: String)
getAttribute(in name: String): String
appendXMLNode(in doc: Document, in father: Element): Docu
toXML(): Document
addAttributes(in element: Element)
SAXModelBuilder
startElement(in namespace: String, in localname: String, in qname: String
endElement(in namespace: String, in localname: String, in qname: String
characters(in ch: char[], in start: int, in len: int)
setProperty(in name: String, in target: Object, in value: Object)
getModel(): XMLElement
«access»
«import»
- stack
*
messages::Message
TYPE_REQUEST: String
TYPE_RESPONSE: String
TYPE_NOTIFY: String
TYPE_EXCEPTION: String
Message()
appendXMLNode(in doc: Document, in father: Element): Dc
getRequestObject(): Request
setRequestObject(in requestObject: Request)
getResponseObject(): Response
setResponseObject(in responseObject: Response)
getExceptions(): ArrayList
addException(in exception: ServerException)
isError(): boolean
getExceptionsAsHTML(): String
messages::Response
addDataObject(in object: DataObject)
getObjects(): List
appendXMLNode(in doc: Document, in father: Element)
setDataObjects
messages::Request
appendXMLNode(in doc: Document, in father: Element): Docu
getParameters(): Hashtable
addParameter(in name: String, in value: String)
addParameter(in name: String, in value: String, in lenient: boo
addParameter(in name: String, in param: RequestParameter)
getParameterValue(in name: String): String
addParameter(in name: String, in file: File)
messages::ServerException
appendXMLNode(in doc: Document, in father: Eleme
embedException(in base: ExceptionBase)
- exception
messages::DataObject
addParameter(in name: String, in value: String)
getParameter(in name: String): String
appendXMLNode(in doc: Document, in father: Element): D
- object
messages::RequestMessage
RequestMessage(in object: String, in command: String)
addParameter(in name: String, in value: String, in lenient: boole
addParameter(in name: String, in param: RequestParameter)
addParameter(in name: String, in value: String)
addParameter(in name: String, in file: File)
messages::ResponseMessage
ResponseMessage()
messages::NotifyMessage
NotifyMessage()

Figure 10: Class diagram of the XML message classes

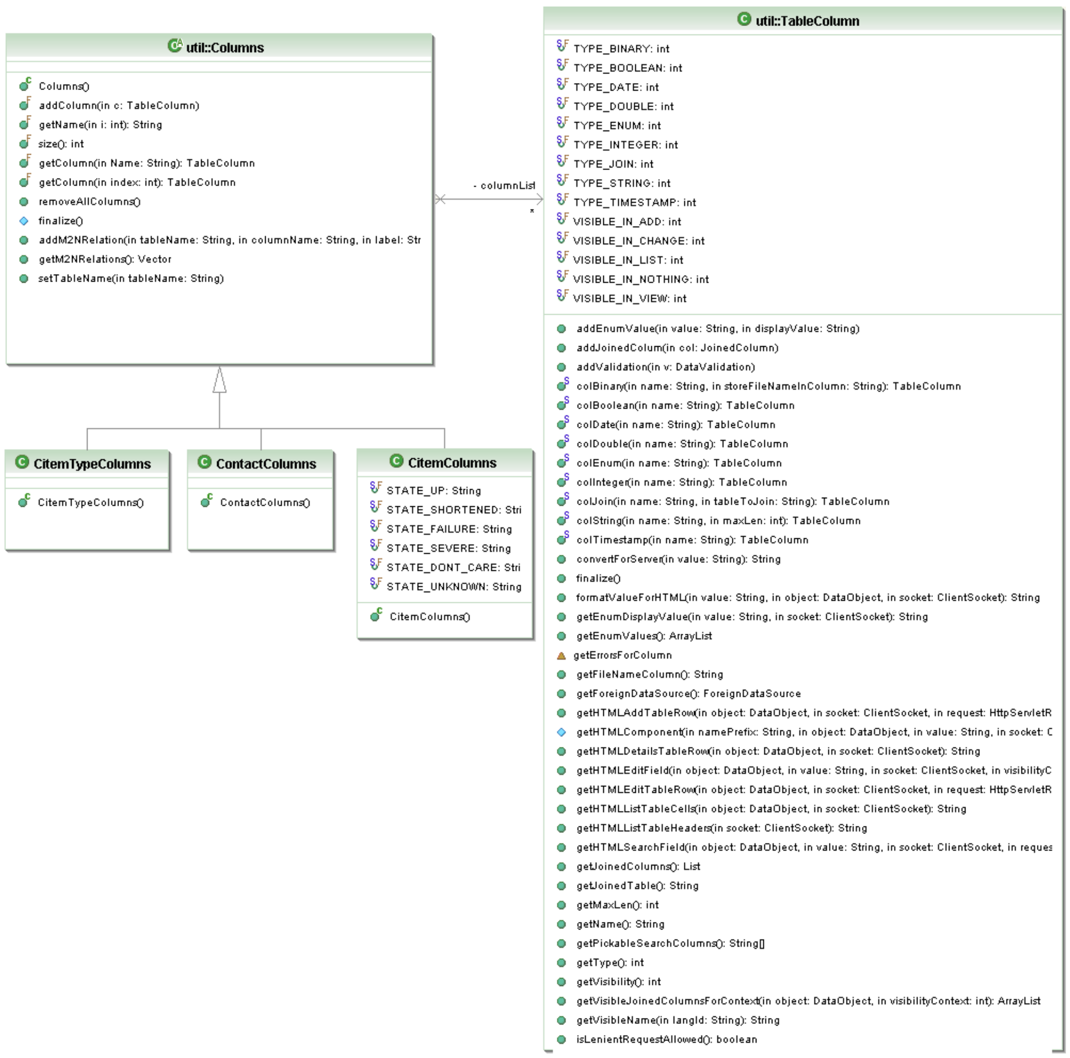

util::Columns
Columns()
addColumn(in c: TableColumn)
getName(in i: int): String
size(): int
getColumn(in Name: String): TableColumn
getColumn(in index: int): TableColumn
removeAllColumns()
finalize()
addM2NRelation(in tableName: String, in columnName: String, in label: Str
getM2NRelations(): Vector
setTableName(in tableName: String)
- columnList
*
CitemTypeColumns
CitemTypeColumns()
ContactColumns
ContactColumns()
CitemColumns
STATE_UP: String
STATE_SHORTENED: Stri
STATE_FAILURE: String
STATE_SEVERE: String
STATE_DONT_CARE: Stri
STATE_UNKNOWN: String
CitemColumns()
util::TableColumn
TYPE_BINARY: int
TYPE_BOOLEAN: int
TYPE_DATE: int
TYPE_DOUBLE: int
TYPE_ENUM: int
TYPE_INTEGER: int
TYPE_JOIN: int
TYPE_STRING: int
TYPE_TIMESTAMP: int
VISIBLE_IN_ADD: int
VISIBLE_IN_CHANGE: int
VISIBLE_IN_LIST: int
VISIBLE_IN_NOTHING: int
VISIBLE_IN_VIEW: int
addEnumValue(in value: String, in displayValue: String)
addJoinedColum(in col: JoinedColumn)
addValidation(in v: DataValidation)
colBinary(in name: String, in storeFileNameInColumn: String): TableColumn
colBoolean(in name: String): TableColumn
colDate(in name: String): TableColumn
colDouble(in name: String): TableColumn
colEnum(in name: String): TableColumn
colInteger(in name: String): TableColumn
colJoin(in name: String, in tableToJoin: String): TableColumn
colString(in name: String, in maxLen: int): TableColumn
colTimestamp(in name: String): TableColumn
convertForServer(in value: String): String
finalize()
formatValueForHTML(in value: String, in object: DataObject, in socket: ClientSocket): String
getEnumDisplayValue(in value: String, in socket: ClientSocket): String
getEnumValues(): ArrayList
getErrorsForColumn
getFileNameColumn(): String
getForeignDataSource(): ForeignDataSource
getHTMLAddTableRow(in object: DataObject, in socket: ClientSocket, in request: HttpServletR
getHTMLComponent(in namePrefix: String, in object: DataObject, in value: String, in socket: C
getHTMLDetailsTableRow(in object: DataObject, in socket: ClientSocket): String
getHTMLEditField(in object: DataObject, in value: String, in socket: ClientSocket, in visibilityC
getHTMLEditTableRow(in object: DataObject, in socket: ClientSocket, in request: HttpServletR
getHTMLListTableCells(in object: DataObject, in socket: ClientSocket): String
getHTMLListTableHeaders(in socket: ClientSocket): String
getHTMLSearchField(in object: DataObject, in value: String, in socket: ClientSocket, in reques
getJoinedColumns(): List
getJoinedTable(): String
getMaxLen(): int
getName(): String
getPickableSearchColumns(): String[]
getType(): int
getVisibility(): int
getVisibleJoinedColumnsForContext(in object: DataObject, in visibilityContext: int): ArrayList
getVisibleName(in langId: String): String
isLenientRequestAllowed(): boolean

Figure 11: Class diagram of the column classes

## 16.4 The XML protocol

The protocol used for communication between the Configuration Management server and its clients is based on XML. All communication is UTF-8 encoded. The protocol defines that, for each message received by the server, a reply has to be sent. The protocol defines four message types: *request*, *response*, *exception* and *notify*. Each message starts with the XML definition for version and encoding.

All messages are encapsulated in a *Message* XML element:

```
<?xml version="1.0" encoding="UTF-8"?>
<Message type="request">
...
</Message>
```

A request message consists of a *Request* element with the two attributes *command* and *object*. The *object* attribute decides which object class is affected and the *command* attribute specifies which operation should be performed by the server. Valid values for the *command* attribute are: *list*, *add*, *change*, *delete*, *subscribe* and *unsubscribe*. All objects have to support the *list*, *subscribe* and *unsubscribe* commands. A certain object may or may not support the *add*, *change* or *delete* commands. Objects can also support other commands to support extended features.
A *Request* element can have zero or more *Value* child elements. A *Value* element has a *name* attribute and a text value. *Value* elements serve as parameters for requests.

```
<?xml version="1.0" encoding="UTF-8"?>
<Message type="request">
     <Request command="list" object="citem">
         <Value name="citemtypeid">1</Value>
     </Request>
</Message>
```

For each request either a response message or an exception is returned. A response message contrains a *Response* element with zero or more attributes depending on the operation performed. *Response* elements can contain zero or more *DataObject* elements. The *Value* child elements of a *DataObject* element follow the same rules as those in *Request* elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<Message type="response">
    <Response>
        <DataObject>
            <Value name="cuserid:users:name">Admin</Value>
            <Value name="useable">t</Value>
            <Value name="name">Administratoren</Value>
            <Value name="cuserid">1</Value>
            <Value name="ctime">2006-01-01 00:00:00</Value>
            <Value name="comment">Administratoren Profil</Value>
            <Value name="id">1</Value>
        </DataObject>
    </Response>
</Message>
```

An exception message contains one or more *ServerException* elements. Each element has the mandatory attributes *type*, *source* and *message*. Depending on the type of the exception more attributes may be present.

```
<?xml version="1.0" encoding="UTF-8"?>
<Message type="exception">
    <ServerException column="name"
                     message="value is required and cannot be null"
                     source="profile"
                     type="InvalidDataException" value=""/>
</Message>
```

Messages with the type *notify* are only sent to clients who have previously enlisted themselves with a *subscribe* request to receive updates for the given object class. The *Response* elements has at least the two attributes *action* and *object*. The *action* attribute notifies of the action which has been performed on the object. In case of a *delete* action the *Response* object also has an *id* parameter which contains the unique key of the deleted object. In case of *add* or *change* actions the *Response* element contains exactly one *DataObject* element which contains the updated values of the added or changed object.

```
<?xml version="1.0" encoding="UTF-8"?>
<Message type="notify">
    <Response action="change" object="citem">
        <DataObject>
            <Value name="name">Fileserver</Value>
            <Value name="id">5</Value>
            <Value name="cuserid">1</Value>
            <Value name="citemtypeid:citemtype:name">Server</Value>
            <Value name="laststate">f</Value>
            <Value name="citemtypeid">1</Value>
            <Value name="currentstate">u</Value>
            <Value name="comment">Shared fileserver</Value>
            <Value name="muserid">1</Value>
            <Value name="ctime">2006-08-29 20:12:12.104092</Value>
            <Value name="muserid:users:name">Admin</Value>
            <Value name="expectedstate">u</Value>
            <Value name="mtime">2006-10-04 13:47:44.247645</Value>
            <Value name="useable">t</Value>
            <Value name="cuserid:users:name">Admin</Value>
        </DataObject>
    </Response>
</Message>
```

## 16.5 System Configuration

To function properly, the Configuration Management server and the servlet need some configuration details. This includes database configuration or debug level switches. Java Properties [Cora] files are used to store the parameters. There is a configuration file for the Configuration Management server and another one for the servlet.
If a configuration file is missing, it will be created by the server or the servlet. The configuration file is generated using default values, which will most likely not fit and prevent the server or the servlet from working properly.

### 16.5.1 Configuration Management Server

The server uses a configuration file named 'server-settings.ini'. It has to be located next to the executable file of the Configuration Management server for the application to find it. The configuration file contains settings for database access, socket binding and session timeout.

### 16.5.2 Servlet Configuration

The servlet uses a configuration file named 'cmservlet-settings.ini'. It is located in the 'config' section of the Sun application server [Corc]. The file contains parameters for how to find the Configuration Management server. The servlet also uses some parameters, which are in the responsibility of the web application server to handle, such as session timeout. These values are set when deploying the servlet package to the application server and can be found in the XML configuration file 'sun-web.xml' located in the 'metadata/servlet' directory of the source package.

## 16.6 Access Control and Security

The Configuration Management system is intended for use in an internal network. Access to the system should not be possible from the world wide web. Potential attackers would therefore need access to the internal network the Configuration Management server is located in.
It is recommended to execute the Configuration Management server under an account with limited access rights and not as super user to minimise the damage caused by potential attackers.
To counter the probability of an successful attack a basic access control and security system has been implemented. All users have a profile assigned which determines which sections of the system the user has access to and which operations he is allowed to perform. A user has to authenticate himself before he can use the system.
The access control is implemented in two levels. First the user interface hides all areas and functions which a user is not allowed to access. This prevents normal users from performing tasks which they are not allowed to carry out. Experienced attackers may bypass the user interface and communicate with the Configuration Management server directly. The application server allows an authenticated user only to perform operations which he is allowed to thus limiting the damage an attacker can cause. Intruders, communicating directly with the Configuration Management server, will only be able to carry out tasks they could perform anyway using the user interface.
Currently the network communication between the Configuration Management server and its clients is not encrypted. Encrypting the network communication would add security to the system and part of the future work on the Configuration Management system.

# Part VI
# Verification

In this part, a real world scenario will be presented and a Configuration Management will be applied to the scenario. The Configuration Management system used in this scenario will be the prototype implementation, which has been developed during this master's thesis.
Please note that all names apart from my own, domain names and network addresses have been changed, to protect the customers privacy and for security reasons.

# 17 The scenario

The scenario takes place in a company, *The Company*, with an internal network consisting of several workstations and servers, either for external customers or internal use. Apart from the internal network, two customers will be described.

## 17.1 The internal network

The internal network of *The Company* contains several workstations. Some are running a Windows based Operating System (OS) and others using a Linux based OS. Table 2 shows all workstations including their IP address and OS.

The internal network also contains several servers used for application development and network management. Table 3 shows all servers and their purpose.

## 17.2 Customer 1

Customer 1 has two sites located in Vienna, *Site N* and *Site S*. *The Company* is maintaining a server at each site. Between those two servers, a secure tunnel is established over the internet using the vtun [Vtu] application. The tunnel is used to route internal traffic from *Site N* to *Site S* and vice versa. Table 4 shows the two servers.

| **Hostname** | **IP** | **OS** | **Description** |
|---|---|---|---|
| geek.local.at | 192.168.0.45 | Gentoo Linux | Workstation of Jürgen Langthaler |
| af.local.at | 192.168.0.30 | Windows XP | Workstation of Armin Herzog |
| develop3.local.at | 192.168.0.44 | Windows XP | Workstation of Peter Eckler |
| box.local.at | 192.168.0.48 | Debian Linux | Workstation of Matthias Bruckner |
| pewh.local.at | 192.168.0.55 | Debian Linux | Workstation of Peter Eldermann |
| office.local.at | 192.168.0.31 | Windows XP | Workstation of Evelyn Herzog |

Table 2: Workstations in the internal network

| **Hostname** | **IP** | **OS** | **Description** |
|---|---|---|---|
| defender.company.at | 200.100.180.1 | Slackware Linux | Router and firewall |
| mail.company.at | 200.100.180.2 | Slackware Linux | Mailserver |
| hotel.local.at | 192.168.0.18 | Slackware Linux | *HotelManagement* development server |
| www.company.at | 200.100.180.5 | Slackware Linux | Webserver |

Table 3: Servers in the internal network

### 17.3 Customer 2

Customer 2 is running the *HotelManagement* software developed by *The Company*. The *HotelManagement* software consists of an application server, referred as *HotelManagement engine*, and a GUI client application. The application server of the *HotelManagement* software is running on a dedicated server machine, while the GUI client is running on a workstation. The dedicated server is also hosting a Tomcat [Fou] application server to provide a webservice. Table 5 shows the servers and workstations located at the site of Customer 2.

## 18 Configuration Management in this scenario

In this section a Configuration Management will be applied to the scenario, which has been presented in the previous section. The ITIL framework de-

| Hostname | IP | OS | Description |
|---|---|---|---|
| gw-n.customer1.at | 200.100.180.66 | Slackware Linux | vtun server at *Site N* |
| gw-s.customer1.at | 200.100.180.106 | Slackware Linux | vtun server at *Site S* |

Table 4: Servers of customer 1

| Hostname | IP | OS | Description |
|---|---|---|---|
| hotel.customer2.at | 192.168.1.10 | Debian Linux | *HotelManagement* and Tomcat server |
| ws.customer2.at | 192.168.1.50 | Windows XP | Workstation running the *HotelManagement* GUI client |

Table 5: Servers and workstations of customer 2

fines two activities before a Configuration Management can be introduced: Planning and Identification.

## 18.1 Planning

The planing phase includes the definition of the scope, purpose, objectives, priorities, policies and procedures of the Configuration Management. Also the types and attributes of all Configuration Item types are defined in this phase.

### 18.1.1 Scope

The scope of the Configuration Management, in this scenario, is the internal network of *The Company* and the Configuration Items located at the sites of the two customers.

### 18.1.2 Purpose

The purpose of the Configuration Management at *The Company* is to install a central database for application and service versions and customer contact information. All important version numbers and contact information will be added to the CMDB.

### 18.1.3 Objectives

The objectives of the Configuration Management at *The Company* are to improve service quality by easier version control and faster response times.

### 18.1.4 Configuration Item types

All Configuration Item types and their attributes have to be defined for each type of Configuration Item, which can be found in the scenario. All Configuration Items have a common attribute *name*, which will be used identify them. The name can be chosen freely, but has to follow a naming scheme according to the companies policies. Following Configuration Item types have been identified:

#### Hardware

- **Server:** Intended for computers running as a server and hosting other services.
  *Attributes:* Vendor, CPU architecture, Number of CPUs, CPU clock speed (MHz), Operating system, OS version

- **Workstation:** Computers, which are used by employees or customers, for everyday use.
  *Attributes:* Operating system, OS version

- **Printer:** Desktop printers.
  *Attributes:* Vendor, Model, Colour printer, Laser printer, LPT interface, USB interface

- **IP Network interface:** A network interface providing IP network connectivity.
  *Attributes:* IP, Vendor, Chipset

- **Switch:** A IP network switch.
  *Attributes:* Vendor, Number of ports

- **RJ45 wall outlet:** A RJ45 wall outlet, which can be used for network connectivity.
  *Attributes:* Label

#### Services

- **Webserver:** A webserver software providing access to HTML pages.
  *Attributes:* Vendor, Version, Port, URL

- **Sendmail:** Sendmail application providing mail services.
  *Attributes:* Version, Port

- **CVS:** CVS server providing repositories for version control.
  *Attributes:* Version

- **Tomcat:** Tomcat application server for hosting webservices and web applications.
  *Attributes:* Version, Port

- **SSH:** Secure shell for remote access.
  *Attributes:* Version

- **DHCP Server:** DHCP server providing dynamic IP addresses.
  *Attributes:* Vendor, Version, DHCP pool start, DHCP pool end

- **DNS Server:** DNS server to resolve domain names to IP addresses.
  *Attributes:* Vendor, Version

- **vtund:** VTun service for secure IP tunnels.
  *Attributes:* Version, Port

- **Database:** Database server providing SQL databases.
  *Attributes:* Version, Port, Type

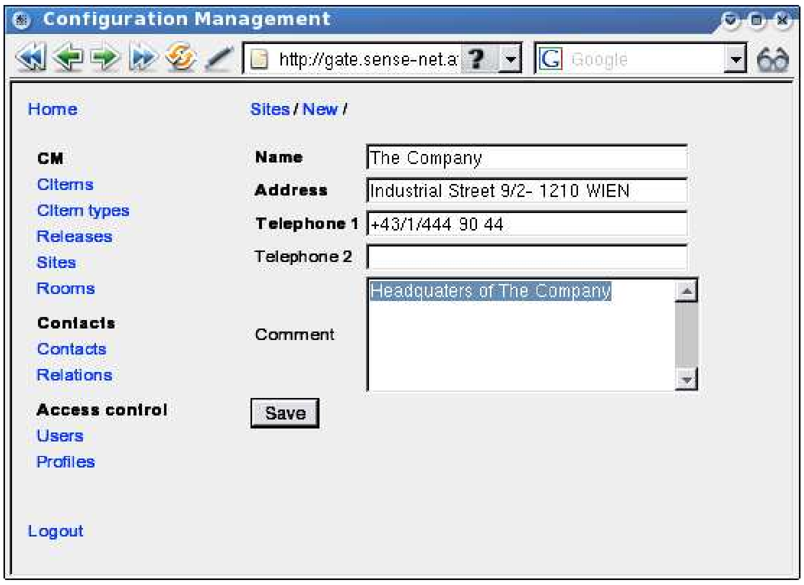
- **HotelManagement engine:** Server side application of the *HotelManagement* software.
  *Attributes:* Version, Port

- **HotelManagement GUI:** Client side GUI application of the *HotelManagement* software.
  *Attributes:* Version

- **HotelManagement WebWS:** A Webservice, which provides access to the *HotelManagement engine* using SOAP messages.
  *Attributes:* Version, WSDL-URL

## 18.2 Identification

During the identification phase all Configuration Items have to be located, assigned a name according to the naming scheme and added to the CMDB. The identification is a time consuming process, which needs to by done by hand and therefore a fair amount of time should be planned for this phase.

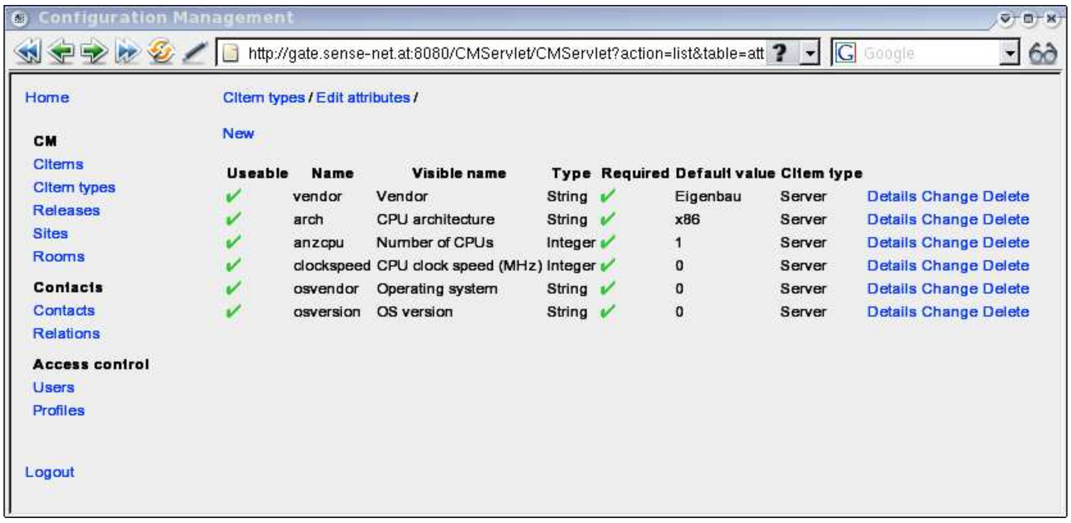


| | |
|---|---|
| Name | The Company |
| Address | Industrial Street 9/2- 1210 WIEN |
| Telephone 1 | +43/1/444 90 44 |
| Telephone 2 | |
| Comment | Headquaters of The Company |

Figure 12: Creating a new site

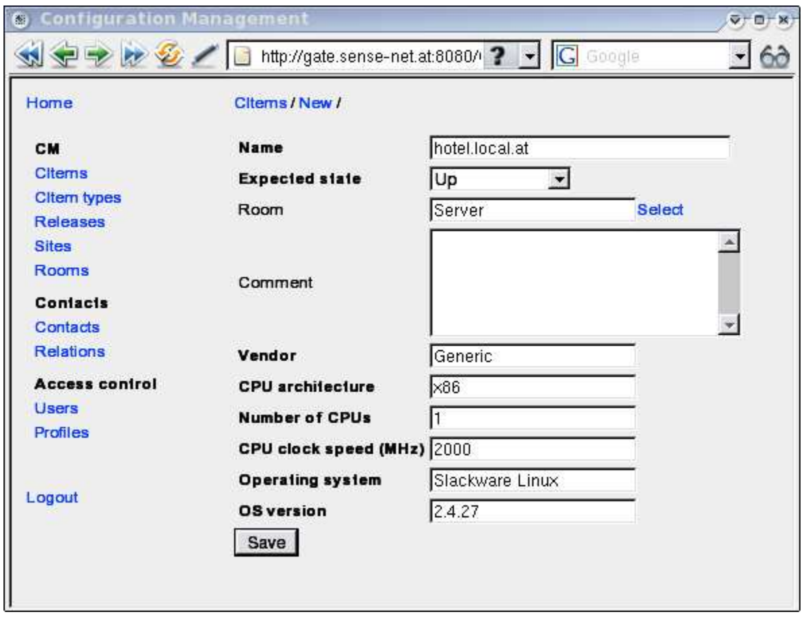
As a future work this manual process could be partially replaced by automated methods using bar code scanners or Radio Frequency Identification (RFID) tags.
It is important that the information gathered during the identification is as accurate and complete as possible. Inaccurate, wrong or missing information will severely limit the benefits of the Configuration Management.
This phase needs to be prepared well as several problems may arise. It needs to be taken care of that all information, which needs to be entered into the CMDB, is accessible. Staff that is responsible for providing information, which isn't directly accessible, has to be in house and reachable. Secured areas, which require access control, need to be either made accessible for the Configuration Management staff or an employee with adequate permissions needs to gather the information for the Configuration Management.

First off all the sites and rooms for *The Company* and the customers have to be created in the Configuration Management system. Figure 12 shows the user interface for the creation of a new site. Furthermore all Configuration Item types and their attributes have to be created before the Configuration Items can be added. Figure 13 shows the dynamic attributes of the *Server* Configuration Management type. If all Configuration Item types have been created, the Configuration Items can be added to the CMDB. Figure 14 shows the creation of a new CI of the type *Server*.



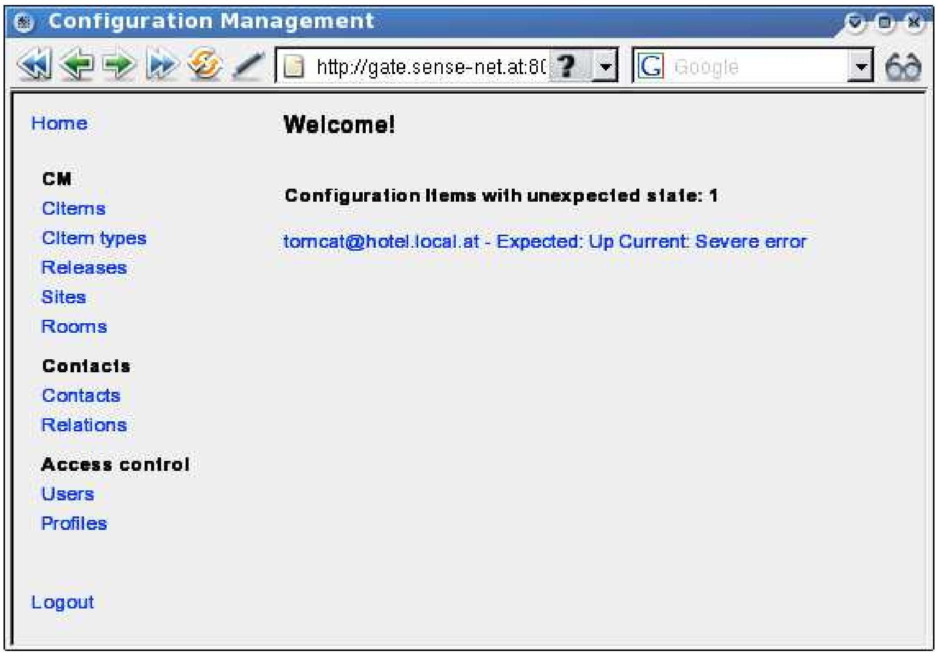
| Useable | Name | Visible name | Type | Required | Default value | CItem type | |
|---|---|---|---|---|---|---|---|
| ✓ | vendor | Vendor | String | ✓ | Eigenbau | Server | Details Change Delete |
| ✓ | arch | CPU architecture | String | ✓ | x86 | Server | Details Change Delete |
| ✓ | anzcpu | Number of CPUs | Integer | ✓ | 1 | Server | Details Change Delete |
| ✓ | clockspeed | CPU clock speed (MHz) | Integer | ✓ | 0 | Server | Details Change Delete |
| ✓ | osvendor | Operating system | String | ✓ | 0 | Server | Details Change Delete |
| ✓ | osversion | OS version | String | ✓ | 0 | Server | Details Change Delete |

Figure 13: Managing attributes of a Configuration Item type



| | |
|---|---|
| Name | hotel.local.at |
| Expected state | Up |
| Room | Server |
| Comment | |
| Vendor | Generic |
| CPU architecture | x86 |
| Number of CPUs | 1 |
| CPU clock speed (MHz) | 2000 |
| Operating system | Slackware Linux |
| OS version | 2.4.27 |

Figure 14: Creating a new Configuration Item


Configuration Management

http://gate.sense-net.at:8(

Home

**CM**

CItems

CItem types

Releases

Sites

Rooms

**Contacts**

Contacts

Relations

**Access control**

Users

Profiles

Logout

**Welcome!**

**Configuration Items with unexpected state: 1**

tomcat@hotel.local.at - Expected: Up Current: Severe error

Figure 15: Home screen of the Configuration Management system

## 19 Representation using the prototype

The presented scenario has been added to the CMDB of the prototype implementation. Adequate Configuration Item types and their attributes have been created and all Configuration Items found in the scenario have been created. The relations among the Configuration Items have been mapped using the *Connection* or *Hosting* relations, which the prototype is capable of. Contacts for customers and employees have been created and related to the Configuration Items.
Figure 15 shows the home screen of the Configuration Management system, which is shown right after a user has logged in. Whenever Configuration Items are in an unexpected state, those Configuration Items are listed on the home screen. On the screenshot you can see that the *tomcat* service currently isn't functional.

### 19.1 Local network

Figure 16 shows some Configuration Items in the local network. All Configuration Items, which are shown in the diagram, are located in the server room of *The Company*.
The diagram shows the router, *defender.company.at*, which connects the local

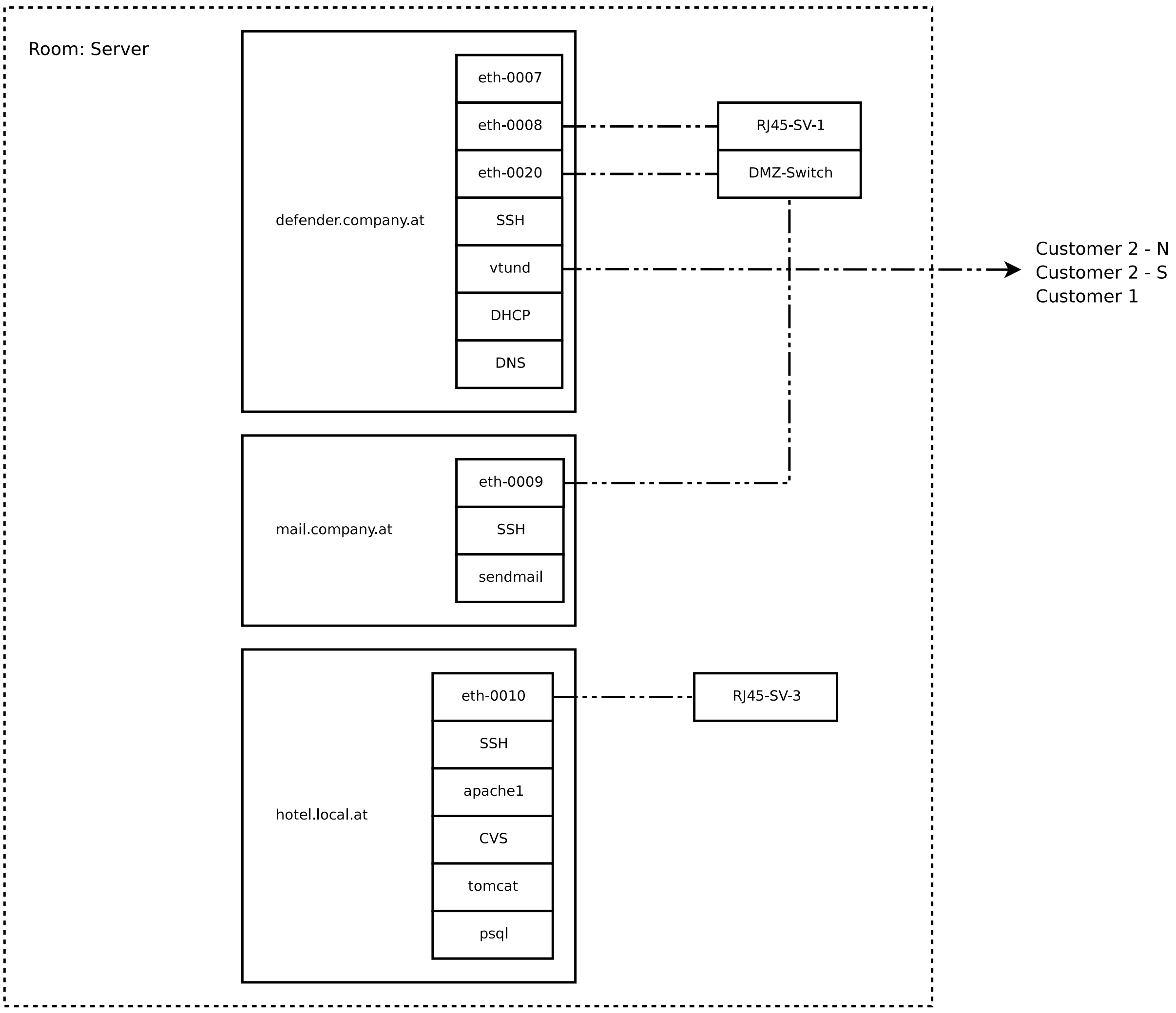
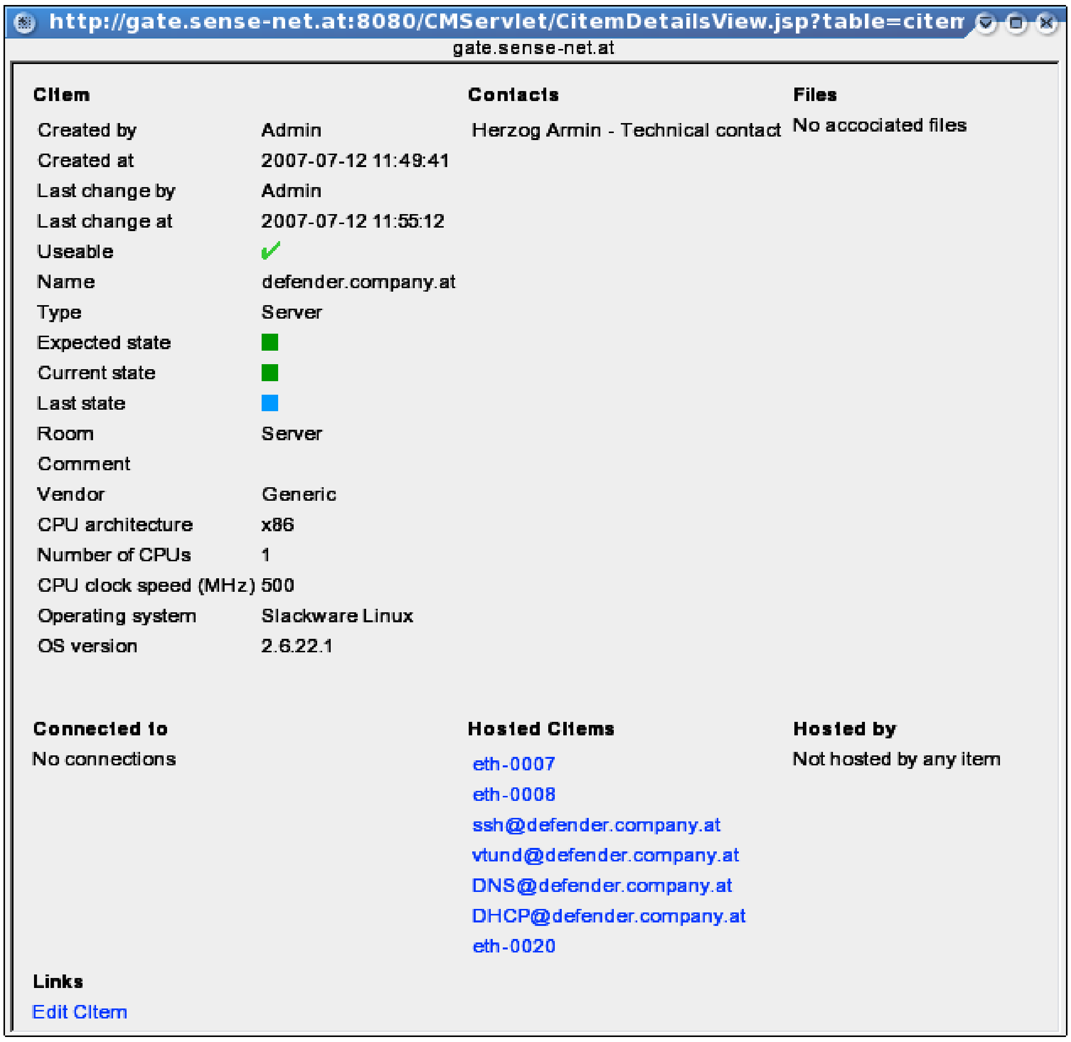


Figure 16: The server room in the local network

network to the worldwide internet. Secure tunnels to customers are established using the *vtun* application.
Apart from the mail server, *mail.company.at*, the development server for the *HotelManagement* software, *hotel.local.at* is shown.
Figure 17 shows a screenshot of the Configuration Item overview. This function provides a summary for a Configuration Item and provides links to hosted or connected items, associated contacts and files. Furthermore the Configuration Item overview provides state information which is updated in realtime whenever the state of the Configuration Item changes.


http://gate.sense-net.at:8080/CMServlet/CitemDetailsView.jsp?table=citem

gate.sense-net.at

**CItem**

| | |
|---|---|
| Created by | Admin |
| Created at | 2007-07-12 11:49:41 |
| Last change by | Admin |
| Last change at | 2007-07-12 11:55:12 |
| Useable | ✓ |
| Name | defender.company.at |
| Type | Server |
| Expected state | 🟩 |
| Current state | 🟩 |
| Last state | 🟦 |
| Room | Server |
| Comment | |
| Vendor | Generic |
| CPU architecture | x86 |
| Number of CPUs | 1 |
| CPU clock speed (MHz) | 500 |
| Operating system | Slackware Linux |
| OS version | 2.6.22.1 |

**Contacts**

Herzog Armin - Technical contact

**Files**

No accociated files

**Connected to**

No connections

**Hosted CItems**

eth-0007
eth-0008
ssh@defender.company.at
vtund@defender.company.at
DNS@defender.company.at
DHCP@defender.company.at
eth-0020

**Hosted by**

Not hosted by any item

**Links**

Edit CItem

Figure 17: Screenshot of the CItem overview

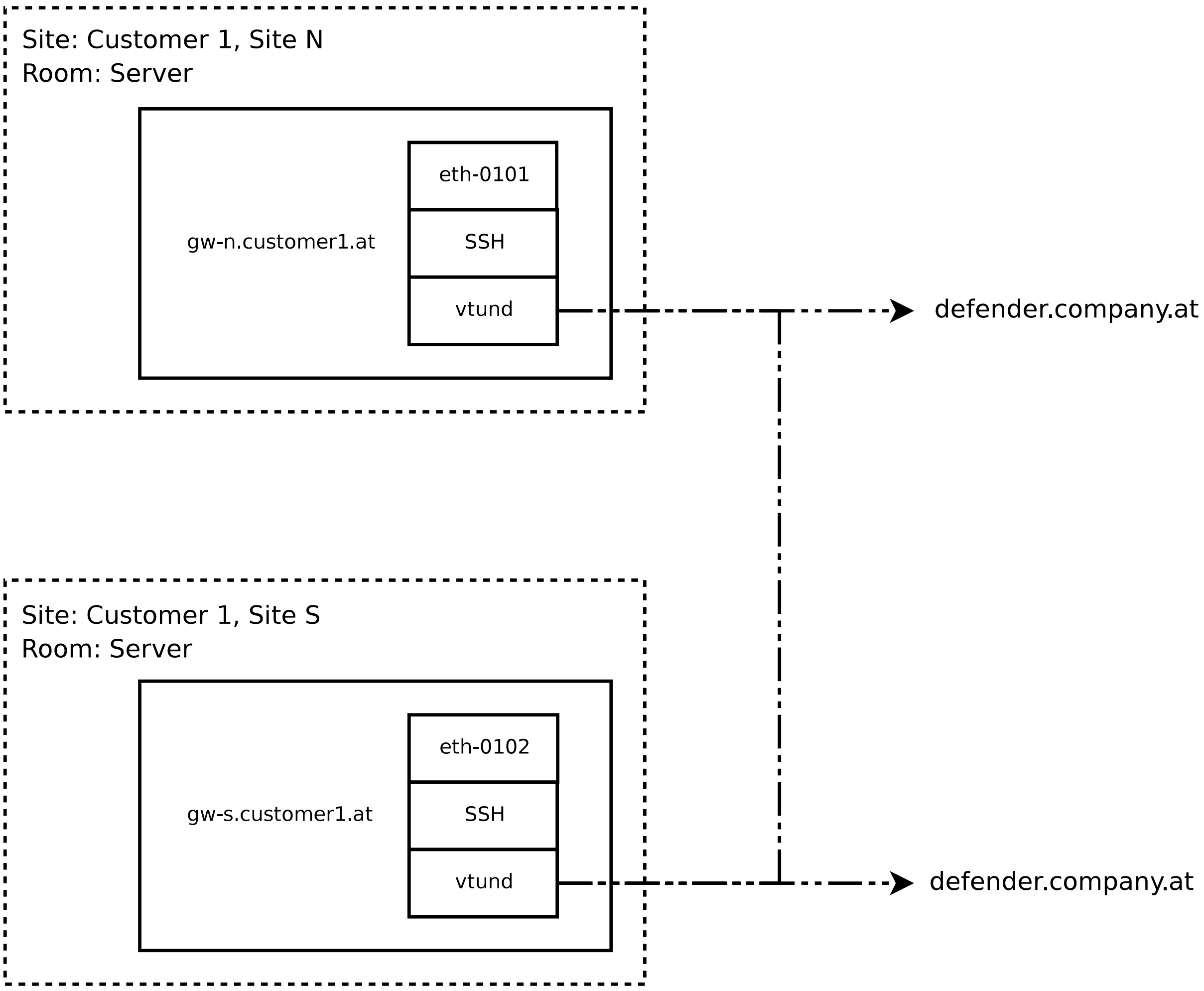


Figure 18: Configuration Items of customer 1

## 19.2 Customer 1

*The Company* is maintaining two servers for customer 1. Each server is located at a different site. The two servers connect the local networks of the two locations of customer 1 with a secure tunnel over the internet. Apart from the tunnel between the two locations, each server establishes a tunnel to *The Company* for maintenance reasons. The Configuration Items of customer 1 are shown in figure 18.

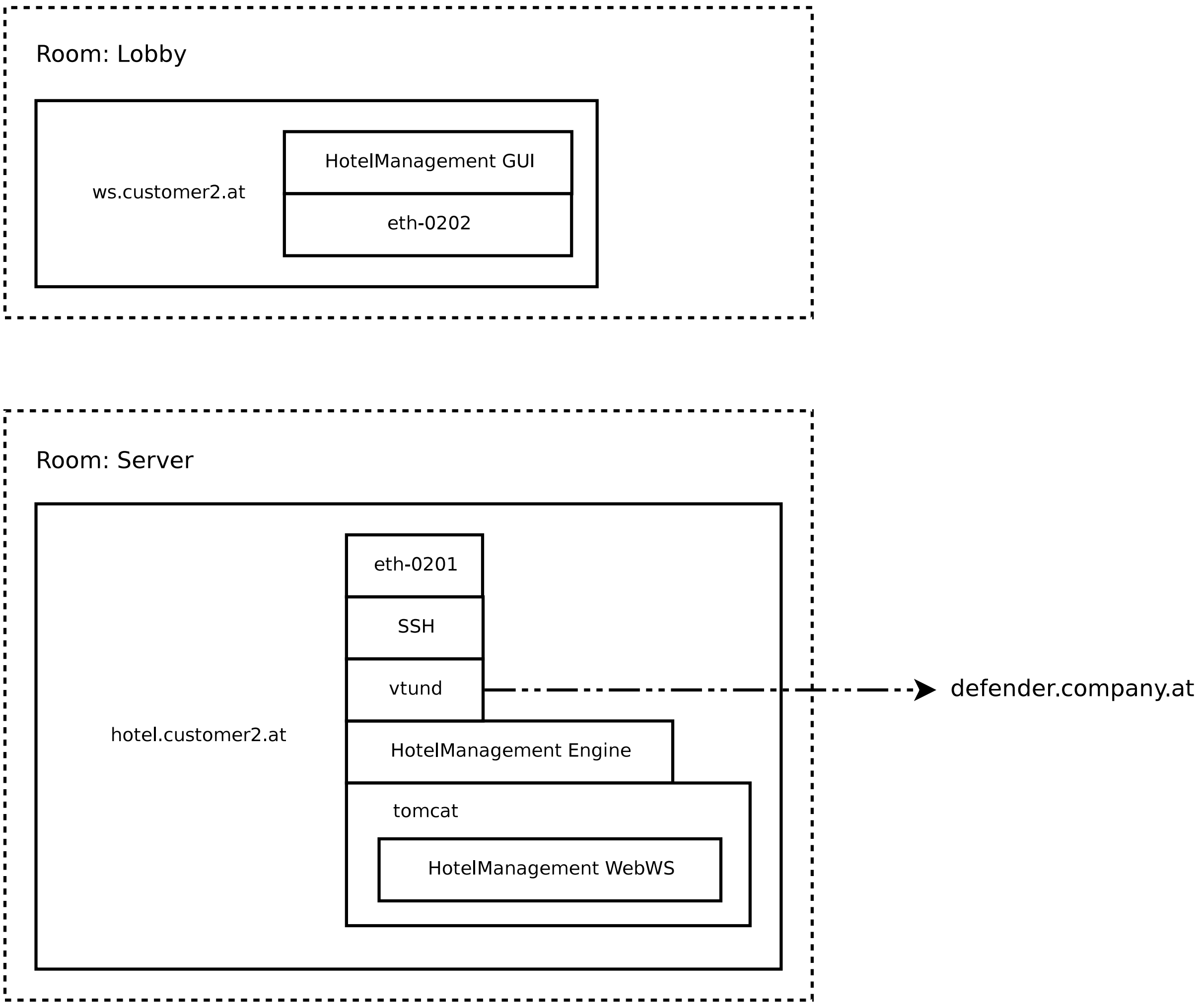
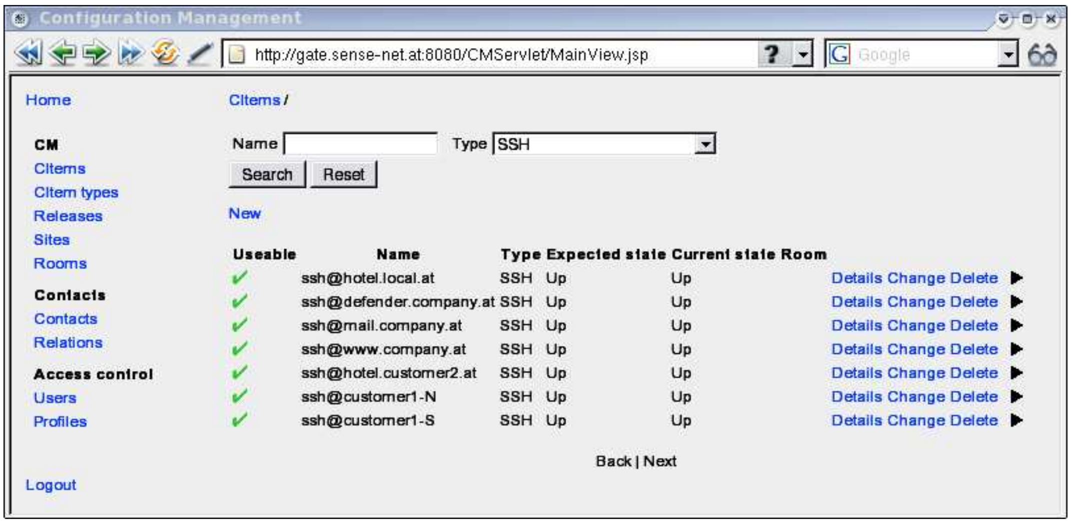


Figure 19: Configuration Items of customer 2

## 19.3 Customer 2

Figure 19 shows the Configuration Items for customer 2. At this location the *HotelManagement* software is installed. The *HotelManagement* GUI client is running on *ws.customer2.at*. The client application connects to the *Hotel-Management engine* service running on *hotel.customer2.at*. This server is also hosting the *Tomcat* application server to provide the *HotelManagement WebWS* webservice.



| Useable | Name | Type | Expected state | Current state | Room | |
|---|---|---|---|---|---|---|
| ✓ | ssh@hotel.local.at | SSH | Up | Up | | Details Change Delete ▶ |
| ✓ | ssh@defender.company.at | SSH | Up | Up | | Details Change Delete ▶ |
| ✓ | ssh@mail.company.at | SSH | Up | Up | | Details Change Delete ▶ |
| ✓ | ssh@www.company.at | SSH | Up | Up | | Details Change Delete ▶ |
| ✓ | ssh@hotel.customer2.at | SSH | Up | Up | | Details Change Delete ▶ |
| ✓ | ssh@customer1-N | SSH | Up | Up | | Details Change Delete ▶ |
| ✓ | ssh@customer1-S | SSH | Up | Up | | Details Change Delete ▶ |

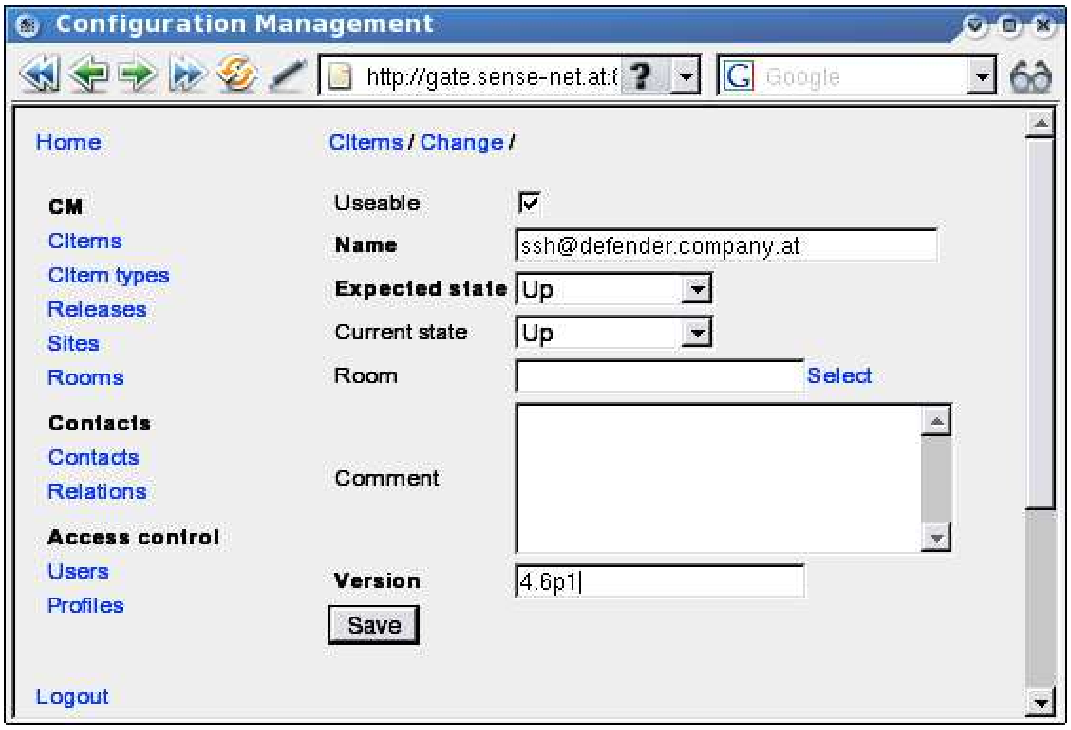
Figure 20: Configuration Items search showing SSH services

# 20 Management of Configuration Items

The previous section has shown that the scenario can be represented by the Configuration Management system prototype. This section will give examples for management task which can be performed using the prototype implementation. The CMDB serves as data source for other ITIL processes and therefore this section will describe tasks, which will be needed by other ITIL processes.

## 20.1 Change Management

A task, which is performed on a regular basis, is the update of the service *OpenSSH* on servers which are important for security. After the changes have been carried out on the services, the infomation stored in the CMDB need to be updated too to keep the database up to date.
Figure 20 shows all secure shell services currently present in the CMDB. The Configuration Management system allows the user to search for Configuration Items of a specific type easily. The *SSH* Configuration Item type has the attribute *version* which will need to be changed to reflect the *OpenSSH* update in the CMDB. Figure 21 shows the editing of the *OpenSSH* service running on *defender.company.at*.
Kernel versions of servers, which are important for security, are updated on a regular basis too. The changes to the CMDB can be carried out similar to the updates of *OpenSSH* versions.


Configuration Management

http://gate.sense-net.at:(

Home

CM
CItems
CItem types
Releases
Sites
Rooms

Contacts
Contacts
Relations

Access control
Users
Profiles

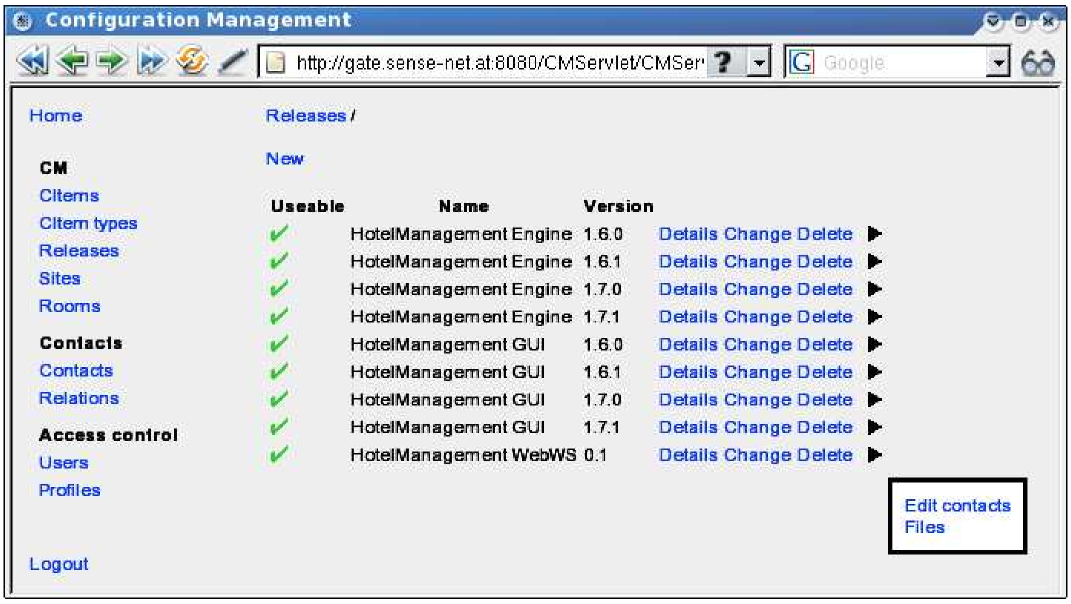
Logout

CItems / Change /

| | |
|---|---|
| Useable | ✓ |
| **Name** | ssh@defender.company.at |
| **Expected state** | Up |
| Current state | Up |
| Room | Select |
| Comment | |
| **Version** | 4.6p1 |

Save

Figure 21: Editing the attributes of an SSH service


Configuration Management

http://gate.sense-net.at:8080/CMServlet/CMSer

Home

CM
CItems
CItem types
Releases
Sites
Rooms

Contacts
Contacts
Relations

Access control
Users
Profiles

Logout

Releases /

New

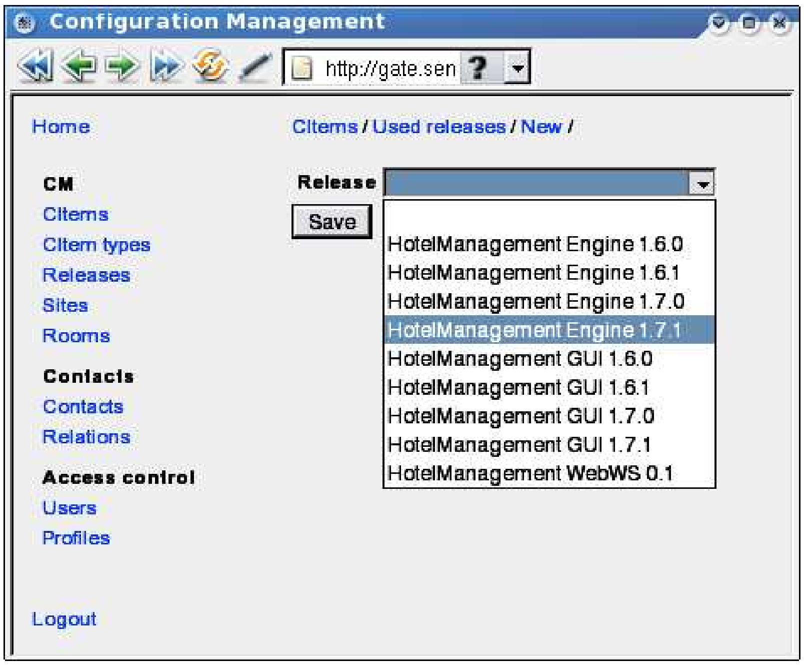
| Useable | Name | Version | |
|---|---|---|---|
| ✓ | HotelManagement Engine | 1.6.0 | Details Change Delete ▶ |
| ✓ | HotelManagement Engine | 1.6.1 | Details Change Delete ▶ |
| ✓ | HotelManagement Engine | 1.7.0 | Details Change Delete ▶ |
| ✓ | HotelManagement Engine | 1.7.1 | Details Change Delete ▶ |
| ✓ | HotelManagement GUI | 1.6.0 | Details Change Delete ▶ |
| ✓ | HotelManagement GUI | 1.6.1 | Details Change Delete ▶ |
| ✓ | HotelManagement GUI | 1.7.0 | Details Change Delete ▶ |
| ✓ | HotelManagement GUI | 1.7.1 | Details Change Delete ▶ |
| ✓ | HotelManagement WebWS | 0.1 | Details Change Delete ▶ |

Edit contacts
Files

Figure 22: Releases currently present in the CMDB

## 20.2 Release Management

The Configuration Management system is capable of managing releases. In the scenario there are different releases of the *HotelManagement* software present. The Release Management is responsible for maintaining the releases. Figure 22 shows the releases currently present in the CMDB. Each release has a set of files associated with it, which are needed to install the release. Figure 23 shows how a new release is associated with a Configuration Item.



HotelManagement Engine 1.6.0
HotelManagement Engine 1.6.1
HotelManagement Engine 1.7.0
HotelManagement Engine 1.7.1
HotelManagement GUI 1.6.0
HotelManagement GUI 1.6.1
HotelManagement GUI 1.7.0
HotelManagement GUI 1.7.1
HotelManagement WebWS 0.1

Figure 23: Attaching a release to a Configuration Item

Part VII

# Future work

In the following part the future work which may emerge from this master's thesis will be presented.

## 21 Extending the implementation for production use

As implementing a complete and productive useable and maintainable Configuration Management system would exceed the scope of this master's thesis, only a prototype of a Configuration Management system has been implemented. For a reasonable useage of the system a full implementation is required. The system in its current state can only serve as a proof of concept for the requirements analysis made and as a starting point for a future implementation.
For a Configuration Management system which may be used in a production environment, important features will have to be added. One of those is a history for Configuration Items providing traceability of changes, which is strongly recommended by the ITIL framework.
Some other aspects of the system could be improved. The binary storage system could either be extended or replaced by an interface to an external file storage system. This would allow the system to offer easier and more detailed file management.
One of the benefits of the ITIL framework is the ability to generate statistics and forecasts based on the data which is collected by the various processes. A suiteable method for report generation has to be added.

## 22 Implementation of other ITIL processes

Configuration Management is only a part of the ITIL framework. Although it is a very important part it has a limited use without being used together with the other ITIL processes.
ITIL defines that the Release Management and Change Management closely work together with the Configuration Management. Therefore the current handling of releases in the system has either to be extended or replaced by an interface to an external application. The integration of an RfC tracking system together with a history for Configuration Items is needed to provide

reasonable results from a Change Management.
Problem Management as well as Incident Management and the Service Desk need a ticketing system which could be connected to the Configuration Management Database. When designing a system used by the Service Desk, high usability requirements have to be met to allow unskilled users to work with it. Designing and implementing such a system is a difficult task and clearly out of the scope of this thesis.

## 23 Configuration Item monitoring

It is important for a Configuration Management to keep the information about the Configuration Items up to date. To reduce the workload of information gathering and data updates, parts of those processes could be automated. Interfacing an existing hardware- or software monitoring application would be one solution to monitor the state of Configuration Items automatically. Especially in large environments this feature is very important, as up to date information about Configuration Items is necessary and updating the state information by hand is not an option, if the number of Configuration Items is large.

# Part VIII
# Conclusion

## 24 The ITIL framework

The ITIL framework is a collection of guidelines and best practices and can be applied to a broad variety of service providers. It offers processes, role definitions and workflow descriptions but the actual implementation of a Configuration Management or any other ITIL process may vary in different companies. The way a Configuration Management will look like is highly dependant on which type of services are provided by a company and which level of detail will be recorded in the Configuration Management database. Building a Configuration Management system, which is applicable for the majority of companies, therefore is a difficult if not impossible goal. The Configuration Management process offers high benefits but there are also pitfalls which need to be taken into account. To make a company benefit from a Configuration Management system the system needs to meet the demands of the company and to provide at least the information which is needed.

Building a generic Configuration Management system is an objective which can only be achieved if the system is able to offer a high level of flexibility. The types of Configuration Items, which attributes should be stored, are not known in advance for a generic system and may change as the business model of a company changes. A generic system has to offer enough flexibility to follow the new business model without loosing functionality and usability.

Not only new types of services provided by a company bring new demands on a Configuration Management system. Also the ITIL process itself can be the reason to change the CMDB. ITIL aims to improve quality by repetitive audits and reviews and thus may reveal new demands on the Configuration Management by its nature. Therefore the Configuration Management system may have to be changed after it has been initially designed to meet the new requirements which have been discovered.

Introducing Configuration Management without the other Service Support processes will serverly limit the benefits which come with a Configuration Management. One of the main objectives of the Configuration Management process is to serve other processes like Incident Management or Change Management as data source which isn't possible if only a Configuration Management is implemented. The improvements which can be achieved, result from all Service Support processes working together rather than from each single

process alone.
Introducing the ITIL processes and the tools needed for them can't fully replace the current infrastructure, which is present in a company, in a revolutionary step. The introduction has to be well prepared and carried out step by step using an evolutionary approach. Interfacing existing tools like monitoring software, ticketing systems or other supportive tools, if present and applicable, is important for the acceptance of the newly introduced process model on both management and employee side. On a long term scale the ITIL framework offers great possibilities to improve the quality of the services provided by a company. Practical experience shows: ITIL cannot be introduced over night and without proper preparation.

# 25 Implementation

The implemented prototype in its current state can only be considered as proof of concept implementation supporting the requirements analysis performed. To serve as a complete and useable tool in a production environment it has to be extended and adapted to the company's additional needs.
Apart from the fact that the prototype in its current state isn't ready for production use, the prototype provides a reasonable base to develop a flexible and full featured Configuration Management system.
While planning the implementation a webservice based approach for the server side part was taken into account. This possibility has been dropped because of the disadvantages, which would have been implied by a webservice. Webservices have become more popular recently and are considered a modern technique providing a standardised interface. A HTTP based webservice is by design based on requests and each TCP connection established for a request is usually closed right after the request has been processed. This introduces an unnecessary performance overhead, especially, when many subsequent requests are made. Apart from this performance overhead I discovered another disadvantage. The plan was to allow Adobe Flash applications, running on the client side, establish a permanent connection to the Configuration Management server and to receive notifications pushed by the server to the clients, whenever data changes. This would allow the client to display up to date information without the need to poll at regular intervals. Unfortunately a webservice based approach would have made this impossible because push based notifications from a HTTP based webservice to its clients are not possible due to the nature of the HTTP protocol used for communication.
In section 14.1 four requirements have been given, which the system has to

fulfil.
The first requirement was that the system has to be client - server oriented. This requirement has been fulfilled. The system consists of a Configuration Management server and one or more clients connecting to the server.
The second requirement was that the server should be implemented as a webservice in Java. This requirement could not be fulfilled because of the reasons given above.
The third requirement was that the system should have a light weight client as user frontend. This requirement has been fulfilled by implementing the user frontend as a web application. The only requirements a client has to meet are a HTML browser which is capeable of Java Script and has the Adobe Flash extension installed. Those requirements are met by most web browsers.
The fourth requirement was that Adobe Flash should be used for visualisations which communicate directly with the webservice. This requirement could also be fulfilled. Instead of the webservice, the Adobe Flash applications directly connect to the Configuration Management server, which replaced the webservice in the system.

# 26 Verification

The mapping of a real world scenario in the prototype implementation has shown that the prototype is basically capable of storing a complex scenario in the CMDB. All relations among the Configuration Items could be represented in the CMDB. The scenario has also shown, that some minor improvements could be applied to the prototype implementation to ease the handling of the user interface. Small workflow improvements and convenience functions could speed up the process of Configuration Item management.
Advanced visualisations, providing an overview of the relations among Configuration Items, would improve the handling further and make the prototype easier to use and relations easier to understand.

# Part IX
# Appendix

## A Database description

This appendix will provide a description of the database used for the prototype implementation. It will describe and explain all attributes of the database tables.

### A.1 Common attributes

All tables of the database have a common set of attributes which keep track of the creating and modification time and which user performed those actions. Furthermore every database table has a single primary key and a flag which determines whether new references to it are allowed or not.

| | |
|---|---|
| **id** | Unique identifier of the type *serial*. |
| **ctime** | Creation time. |
| **cuserid** | Reference to the user who initially created the row. |
| **mtime** | Modification time. |
| **muserid** | Reference to the user who performed the last change on this row. |
| **useable** | Default: *true*; Whenever this flag is set to *false* no new references to this row are allowed by the application server. Existing references are left untouched. |

Table 6: Common attributes

### A.2 Table descriptions

This section will give a description for each table in the database. Tables used for M to N relations will be omitted unless they are needed for further understanding.

| | |
|---|---|
| **Table** | permission |
| **Description** | Holds the permissions which are defined in the system. |
| **Constraints** | The *code* attribute must be unique throughout the table. |
| **code** | Code of the permission. |

Table 7: Description for the table *permission*

| | |
|---|---|
| **Table** | profile |
| **Description** | Holds all profiles which are defined in the system. |
| **Constraints** | The *name* attribute must be unique throughout the table. |
| **name** | Name of the profile. |
| **comment** | Descriptive comment for the profile. |

Table 8: Description for the table *profile*

| | |
|---|---|
| **Table** | users |
| **Description** | Holds all users which may logon to the system. |
| **Constraints** | The *username* attribute must be unique throughout the table. |
| **username** | Username which is needed to logon to the system. |
| **password** | Password of the user. |
| **name** | Real name of the user. |
| **email** | Email address of the user. |
| **tel** | Telephone numer of the user. |
| **comment** | Descriptive comment. |
| **profileid** | Reference to the table *profile*. Defines which profile the user is part of. |

Table 9: Description for the table *users*

| | |
|---|---|
| **Table** | site |
| **Description** | Sites which are present in the system. Rooms in which Configuration Items are placed have to be located at a site. |
| **name** | Name of the site. |
| **address** | Address of the site location. |
| **phone1** | Phone number under which the site can be contacted. |
| **phone1** | Alternative phone number. |
| **comment** | Descriptive comment. |

Table 10: Description for the table *site*

| | |
|---|---|
| **Table** | room |
| **Description** | Rooms in which Configuration Items may be placed. |
| **name** | Name of the room. |
| **phone1** | Phone number under which this room can be contacted. |
| **phone2** | Alternative phone number. |
| **comment** | Descriptive comment. |

Table 11: Description for the table *room*

| **Table** | citemtype |
|---|---|
| **Description** | Holds all Configuration Item types. The type of a CI defines which attributes it has. |
| **name** | Name of the type. |
| **comment** | Descriptive comment. |

Table 12: Description for the table *citemtype*

| **Table** | attribute |
|---|---|
| **Description** | Dynamic attributes which can be defined for each CI type. |
| **name** | Internal name of the attribute. |
| **displayname** | Name which will be displayed to the users of the system. |
| **type** | Type of the values which are expected by this attribute. |
| **required** | Whether this attribute is required or not. |
| **defaultvalue** | Default value which will be assigned to all existing Configuration Items if the value is required. |
| **citemtypeid** | Reference to the CI type this attribute belongs to. |

Table 13: Description for the table *attribute*

| **Table** | attributevalue |
|---|---|
| **Description** | This table holds the actual values of the attributes which are defined by the CI type for each Configuration Item. |
| **value** | The value which is stored for an attribute of a Configuration Item. |
| **attributeid** | Reference to the attribute this value belongs to. |
| **citemid** | Reference to the CI this value belongs to. |

Table 14: Description for the table *attributevalue*

| **Table** | canhost |
|---|---|
| **Description** | Defines which types of Configuration Items can be hosted by the ones with CI type referenced with the attribute *citemtypeid*. |
| **citemtypeid** | Reference to the CI type for which the hostable types are defined. |
| **canhostcitemtypeid** | Reference to the CI type of which Configuration Items can be hosted by Configuration Items with the type referenced with *citemtypeid*. |

Table 15: Description for the table *canhost*

| **Table** | connectableto |
|---|---|
| **Description** | Defines which Configuration Items can connect to ones with the CI type referenced with the attribute *citemtypeid* |
| **citemtypeid** | Reference to the CI type for which the possible connections are defined. |
| **connectabletocitemtypeid** | Reference to the CI type of which Configuration Items can be connected to Configuration Items with the type referenced with *citemtypeid*. |

Table 16: Description for the table *connectableto*

| **Table** | citem |
|---|---|
| **Description** | Holds all Configuration Items which are present in the system. This table only holds the attributes which are common for all Configuration Items. |
| **name** | Name of the Configuration Item. |
| **comment** | Descriptive comment. |
| **roomid** | Reference to the room in which this CI is located. |
| **citemtypeid** | Reference to the type of which this CI is. |
| **expectedstate** | State in which this CI is expected in normal operation. |
| **currentstate** | Current state of the CI. |
| **laststate** | State which the CI was in before the current state. |

Table 17: Description for the table *citem*

| **Table** | isconnected |
|---|---|
| **Description** | Holds all connection definitions for all Configuration Items. |
| **citemid** | Reference to the CI for which this connection is stored. |
| **connectedtocitemid** | Reference to the CI which the CI referenced with *citemid* is connected to. |

Table 18: Description for the table *isconnected*

| **Table** | ishosted |
|---|---|
| **Description** | Holds all hosting relations for all Configuration Items. |
| **citemid** | Reference to the CI for which this hosting relation is stored. |
| **hostedcitemid** | Reference to the CI which is hosted by the CI referenced with *citemid*. |

Table 19: Description for the table *ishosted*

| **Table** | contact |
|---|---|
| **Description** | All contacts which are stored in the system. This may include internal personell as well as customers. |
| **lastname** | Last name of the contact. |
| **firstname** | First name of the contact. |
| **street** | Street of the contacts address. |
| **zipcode** | Zip code in which the contact is located. |
| **city** | City the contact resides in. |
| **phone1** | Phone number of the contact. |
| **phone2** | Alternative phone number. |
| **fax** | Fax number. |
| **email** | Email address of the contact. |
| **comment** | Space for all information which doesn't fit elsewhere. |

Table 20: Description for the table *contact*

| | |
|---|---|
| **Table** | relationtype |
| **Description** | Types of relations which may be used to link contacts to Configuration Items or releases. |
| **name** | Name of the relation type. This name will be displayed to users. |
| **comment** | Descriptive comment. |

Table 21: Description for the table *relationtype*

| | |
|---|---|
| **Table** | relatedtocitem |
| **Description** | This table holds all relations which are defined between contacts and Configuration Items. |
| **contactid** | Reference to the contact who is related to a CI. |
| **citemid** | Reference to the CI which the contact is linked to. |
| **relationtypeid** | Reference to the relationtype the the relation is of. |

Table 22: Description for the table *relatedtocitem*

| | |
|---|---|
| **Table** | file4citem |
| **Description** | This table links binary data to Configuration Items. |
| **Constraints** | This table doesn't acutally contain the binary data but a reference (*data*) to it. |
| **citemid** | Reference to the CI for which this file is stored. |
| **data** | OID under which the binary data can be referenced. |
| **filename** | Filename the file had when it was stored. |
| **comment** | Descriptive comment. |

Table 23: Description for the table *file4citem*

| | |
|---|---|
| **Table** | release |
| **Description** | Releases which are defined in the system. |
| **name** | Name of the release. |
| **version** | Version of the release. |
| **comment** | Descriptive comment. |

Table 24: Description for the table *release*

| **Table** | relatedtorelease |
|---|---|
| **Description** | This table holds all relations which are defined between contacts and releases. |
| **contactid** | Reference to the contact who is related to a release. |
| **citemid** | Reference to the release which the contact is linked to. |
| **relationtypeid** | Reference to the relationtype the the relation is of. |

Table 25: Description for the table *relatedtorelease*

| **Table** | file4release |
|---|---|
| **Description** | This table links binary data to releases. |
| **Constraints** | This table doesn't acutally contain the binary data but a reference (*data*) to it. |
| **citemid** | Reference to the relase for which this file is stored. |
| **data** | OID under which the binary data can be referenced. |
| **filename** | Filename the file had when it was stored. |
| **comment** | Descriptive comment. |

Table 26: Description for the table *file4release*

# B Installation and configuration

This section of the appendix will focus on how to set up the prototype implementation and how to configure the system to be working in your local environment.

## B.1 Installation

This section will guide you through the steps of the installation and give background information about how to compile the source code, deploy the servlet package to the application server and set up the Postgre SQL database.

### B.1.1 Software requirements

Following software is required to compile the source code successfully and to run the prototype:

- **Sun JDK version 5.0 or newer**

  A Java compiler is required to build the source code. Also the JVM is needed to run the resulting binary packages.

- **Apache ant**

  The ant tool, developed by the Apache Ant Project, is needed to use the build script provided. I used the version 1.6.2 during development.

- **Postgre SQL version 8.x**

  The Postgre SQL database is used by the CM server as data backend. I tested the versions 8.0.4 and 8.0.13 but any other version based on the 8.x branch will most likely work too.

- **Sun Java System Application Server Platform Edition 9.0**

  The application server provided by the Sun Corp. was used to host the servlet. The application server is part of the Java EE 5 SDK. Also an open source branch of this application server exists. The project is called Glassfish [Coma] and basically provides the same functionality as the Sun Java System Application server and can be used as an alternative. Other application servers may work too but haven't been tested.

For the installation and configuration of the software given above please see the manuals provided by the vendors.

### B.1.2 Compiling the source code

To compile the sources and generate a package which is deployable to the application server the Apache ant [Pro] tool is used. It is independent of a specific development environment or IDE and thus provides a reuseable method for building a deployable binary package. Ant is used to build a JAR file of the Configuration Management server and a deployable WAR file containing the Java Server Pages.
If ant is called without any target parameter the *all* target is executed. This target builds the server, the servlet and the shared library, which is used by the CM server and the servlet. Finally the servlet package is deployed to the application server. The path, where the servlet package is deployed to, can be adjusted by editing the *build.xml* file and changing the *deploypath* property. Apart from the *all* target, the following high level targets are available:

- ***shared***

  This target builds only the shared library. The resulting jar file can be found in the *lib* subdirectory of the source root.

- ***server***

  This target builds the CM server and all its requirements. A jar file named *CMServer.jar* is created in the source root directory.

- ***servlet***

  By calling this target the servlet package is built and the deployable *CMServlet.war* file is created in the source root directory. After the package has been built successfully, the file is deployed to the application server.

- ***clean***

  This target cleans all created binary files for server, servlet and the shared library.

### B.1.3 Database setup

As soon as the Postgre SQL server is running you may now import the *create_tables.sql* script located in the *db* subdirectory of the source root. The script will create a new database with the name *cmdb* and create all required tables. You may want to change the user, which is set as owner of the database, by editing the script.

### B.1.4 Configuration Management server

To run the CM server the *runServer.sh* script is provided. It initialises the JVM with all parameters required and starts *CMServer.jar* file, which can be found in the source root directory, after the server source has been compiled.

### B.1.5 Servlet installation

The only steps required to install the servlet are compiling the source code and deploying the resulting war file to the application server. After building the source code ant will try to deploy the servlet package automatically. The path, where the servlet package is deployed to, can be adjusted by editing the *build.xml* and changing the *deploypath* property.

### B.1.6 The 'dbinit' tool

This tool reads data from CSV files, connects to the CM server and tries to add the rows into the database. Currently this tool is used to import the permissions for the access control of the CM server but it can easily be adapted to import any type of data.
The tool takes a set of command line parameters:

```
dbinit <IP> <Port> <username> <password> <data>
```

- **IP**

  The IP of the CM server, into which the data should be imported.

- **Port**

  The port the CM server is listening on.

- **Username**

  The username which should be used for the connection. Usually the *admin* superuser will be needed here, especially if permissions are not present in the system yet.

- **Password**

  The password to the users account given in the *username* parameter.

- **data**

  The type of data which should be imported. The tool expects a *'data'.csv* file to be present in the *data* subdirectory.

To run the tool the script *run.sh* can be used.

## B.2 Configuration

The Configuration Management server and the servlet each have their own configuration file, which contains values needed to customise the applications to a local environment. The configuration files use the Java Properties format [Cora]. Whenever a settings file is missing the CM server or the servlet will create a new one with default values, which will most likely not fit your needs.

### B.2.1 Configuration Management server

The CM server properties are stored in the file *server-settings.ini*. It allows you to adjust following settings:

- **DBName**

  Name of the database which the server will use. The default value is *cmdb*.

- **DBUsername**

  Postgre SQL user which will be used to connect to the database. The user should have sufficient rights to insert, update and delete data within the database.

- **DBPassword**

  Password which will be used while connecting to the database.

- **ServerInterface**

  The network interface the server should bind to. The default value is *127.0.0.1* which will only allow connections from localhost.

- **ServerPort**

  The TCP port the server will bind to. The default value is *4444*.

- **SessionTimeout**

  The time in seconds after which an inactive session will be closed. The default value is *3600*.

### B.2.2 Servlet configuration

The servlet basically needs information about how to connect to the CM server:

- **ServerHost**

  The hostname or IP address of the machine the CM server is running on. Default is *localhost*.

- **ServerPort**

  The port which the CM server is listening to. The default value is *4444*.

### B.2.3 Error logging

Both, server and servlet use the same logging concept and their configuration files also contain the following properties which are needed to control the logging system.
The applications can either write their output to a file or to standard output/error. If logging to files is enabled, a new log file named *YYYY-MM-DD.log* will be created for each day. The log files can be found in the *cm-logs* subdirectory.

- **DebugLevel**

  This property can be used to limit the amount of information which is logged. Currently four log levels exist: *Error* (5), *Warning* (10), *Notice* (15) and *Info* (20). Higher values will increase the amount of information written to the logs, lower values will decrease it. The default and recommended value is *Notice* (15).

- **LogToFile**

  If this property is set to *1* (default), the logging output will be written to a log file instead to standard output/error. If the property is set to *0* the application output will be written to standard output/error.

- **LogFileTTLDays**

  Number of days the log files are kept. Each time the applications are started they try to delete outdated logfiles. If the value is set to *-1*, no log files will be deleted.

- **LogLineLength**

  This limits the number of characters a log entry line can have. This value is useful to speed up logging when not logging to a file. Setting the value to *-1* will disable log line truncation.

# Nomenclature

| | |
|---|---|
| BLOB ....... | Binary Large Object |
| CAB ......... | Change Advisory Board |
| CCTA ....... | Central Computer Agency |
| CI ........... | Configuration Item |
| CMDB ....... | Configuration Management Database |
| CR ........... | Change Request |
| DHS ......... | Definitive Hardware Store |
| DSL ......... | Definitive Software Library |
| ER ........... | Entity-Relationship Model |
| ICT .......... | Information and Communication Technology |
| ITIL ......... | IT Infrastructure Library |
| JSP .......... | Java Server Pages |
| JVM ......... | Java Virtual Machine |
| OGC ......... | Office of Government Commerce |
| OS ........... | Operating System |
| RfC .......... | Request for Change |
| RFID ........ | Radio Frequency Identification |
| SLA ......... | Service Level Agreement |
| SLM ......... | Service Level Management |
| SLR ......... | Service Level Requirements |
| SPOC ........ | Single Point Of Contact |
| XML ......... | Extensible Markup Language |

# References

[BGSS] Michael Brenner, Markus Garschhammer, Martin Sailer, and Thomas Schaaf. CMDB - Yet Another MIB? http://www.nm.ifi.lmu.de/pub/Publikationen/bssg06/.

[Coma] Glassfish Community. Glassfish Community. https://glassfish.dev.java.net/.

[Comb] JBoss Community. Jboss. http://labs.jboss.com/.

[Cora] Sun Corp. Java Properties class. http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html.

[Corb] Sun Corp. Java server pages technology. http://java.sun.com/products/jsp/.

[Corc] Sun Corp. Java System Application Server. http://www.sun.com/software/products/appsrvr/index.xml.

[dw] dv werk. http://www.dv-werk.de/.

[Fou] Apache Software Foundation. Tomcat. http://tomcat.apache.org/.

[fSidI] Bundesamt für Sicherheit in der Informationstechnik. IT Service Management nach ITIL. http://www.bsi.bund.de/literat/studien/ITinf/itil.pdf.

[Gui] ITIL Open Guide. ITIL Open Guide. http://www.itlibrary.org/.

[iL00] Norwich itSMF Ltd. *IT Service Management - Die IT Infrastructure Library.* ISBN 3-9521449-0-8, 2000.

[Inca] Adobe Systems Incorporated. Adobe Flash. http://www.adobe.com/de/products/flash/.

[Incb] Adobe Systems Incorporated. Adobe Flash XMLSocket class. http://livedocs.adobe.com/fms/2/docs/00000839.html.

[Jäg05] Jens Jäger. Entwicklung eines Toolkonzeptes für die Unterstützung der ITIL Service Support Prozesse. Master's thesis, Institut für Informatik, TU München, 2005.

[Mar] Oliver Martin. Configuration Management - A CA IT Service Management Process Map. http://cview.bitpipe.com/detail/RES/1153342070_328.html.

[oGC] Office of Government Commerce. ITIL IT Service Management. http://www.itil.co.uk/.

[oGC03] Office of Government Commerce. *Software Asset Management*. London : TSO, 2003.

[oGC05a] Office of Government Commerce. *Service Delivery*. London : TSO, 2005.

[oGC05b] Office of Government Commerce. *Service Support*. London : TSO, 2005.

[Pla05] Mag. Andreas Plamberger. IT Service Management nach ITIL. http://www.big.tuwien.ac.at/teaching/offer/ss05/usi1/usi1itil.pdf, 2005.

[Posa] PostreSQL. PostgreSQL. http://www.postgresql.org/.

[Posb] PostreSQL. PostgreSQL about page. http://www.postgresql.org/about/.

[Posc] PostreSQL. PostgreSQL documentation. http://www.postgresql.org/docs/7.4/static/jdbc-binary-data.html.

[Pro] Apache Ant Project. Apache Ant. http://ant.apache.org/.

[Rsy] Rsync. Rsync homepage. http://samba.anu.edu.au/rsync/.

[Vtu] Vtun. Vtun homepage. http://vtun.sourceforge.net/.

[ZS05] Andrea Ziegler-Skopecek. Qualitätsverbesserung von Software Entwicklungsprojekten in der IT Branche. Master's thesis, Wirtschaftsuniversität Wien, 2005.