CHALMERS

Supporting a Transition from Manual to Automated Functional Testing

How the design and implementation of an application/database/web-interface trio can help a department with automating their functional testing.

Master of Science Thesis

ELIN WEBER
ERIK STERNERSON

Department of Computer Science and Engineering
Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, September 2009

Supporting a Transition from Manual to Automated Functional Testing
How the design and implementation of an application/database/web-interface trio can help a department with automating their functional testing.
ELIN C WEBER
ERIK J STERNERSON

Examiners: Martin Fabian[1] and Sven-Arne Andreasson[2]
[1]Department of Signals and Systems
[2]Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
Photo taken by user [kilodelta] on http://flickr.com
Photo available under the Creative Commons:
Attribution-Noncommercial-Share Alike 2.0 Generic license

Department of Computer Science and Engineering
Department of Signals and Systems
Göteborg, Sweden, September 2009

ABSTRACT

Functional testing is a very important part of the verification work done before releasing a technical product to market. The results of the functional tests can be used both to prove that the product fulfills the requirements, but also to ensure that errors that were discovered in an earlier version of the product have not reappeared. Functional testing can either be performed manually, automatically or as a mix of both. We present our experiences from our work on designing and implementing a system intended to support a transition from a mostly manual testing work-flow to a partly automated one, focusing on a smooth transition between the two states. The system developed is tailored suit both manual and automated testing on several products, configurations and test targets in parallel which differs heavily from a scenario where one product is tested in one configuration on one target. Our work consists of designing and implementing: A test scheduler capable of executing several tests in parallel on different targets, a web interface allowing users to search for suitable test targets, with support for both manual and automated testing, and a database storing information for the scheduler and the web interface.

SAMMANFATTNING

Funktionell testning är en mycket viktig del av verifikationsarbetet som görs innan en produkt släpps på marknaden. Resultaten från den funktionella testningen kan användas både för att bevisa att produkten uppfyller kraven, men även för att försäkra sig om att brister som upptäckts i en tidigare version av produkten inte dykt upp igen. Funktionell testning kan antingen utföras manuellt, automatiskt eller som en mix av båda. Vi presenterar våra erfarenheter från vår del av arbetet med design och implementering av ett system med avsikt att stödja en övergång från ett mestadels manuellt testflöde till ett delvis automatiskt sådant, med fokus på en smidig övergång mellan de båda stadierna. Det utvecklade systemet är anpassat för att passa både manuell och automatisk testning på flera produkter, konfigurationer och testobjekt samtidigt, vilket skiljer sig stort från ett scenario där en produkt med en konfiguration testas på ett testobjekt. Vår del av arbetet består av att designa och implementera: En schemaläggare för test som kan exekvera flera test samtidigt på olika testobjekt, ett webbgränssnitt som ger användare möjligheten att söka för passande testobjekt, med stöd för både manuell och automatisk testning, samt en databas som håller information som används för schemaläggaren och webbgränssnittet.

PREFACE

This report was written as part of a 20 week Master's Thesis project. The project was carried out at the Integration and Verification department of the Mobile Broadband Modules unit at Ericsson, Sweden. The report is a cross-disciplinary work by two students at the Chalmers University of Technology; Elin Weber from the Systems, Control and Mechatronics Master's Program and Erik Sternerson from the Software Engineering and Technology Master's Program.

The authors would like to thank their examiners/mentors at Chalmers University of Technology, Martin Fabian and Sven-Arne Andreasson for their guidance and support.

This report describes the planning, design and implementation of a system to assist the Integration and Verification department with transitioning from a mostly manual functional test execution to a mostly automated one. This effort would not have been possible without the fantastic work made by co-workers, whose names are withheld for the sake of confidentiality.

The division of work between the members of this project (Weber and Sternerson) has generally been an even split on most tasks, except as noted below.

Elin Weber has done the majority of the work on the following tasks: design of the database, implementation of the administrative functionality of the web interface and implementation of the test automation logic of the test coordinator.

Erik Sternerson has done the majority of the work on the following tasks: system architecture of the web interface and coordinator, implementation of the coordinator, apart from the test control logic, and implementation of the supporting systems.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

This chapter describes the background, scope and problem definition for the project. The chapter also contains related work and methodology, as well as guidelines for reading the report.

1.1 Background

The market for mobile broadband has increased dramatically during the last few years, and there are no signs that the rate of increase will slow down in the near future. One result of this increase is that at Ericsson, a worldwide company developing telecom solutions, the Mobile Broadband Modules (MBM) unit of Ericsson has expanded from tens of engineers two years ago to over one hundred engineers today. As the amount of development increases, so does the amount of integration and verification.

Prior to this project, the functional verification of the broadband modules was done mostly manually. A test engineer was given a set of instructions and a list of expected results, with the task to match these results to the actual outcome. In the "early days", when development targeted a handful of computer models with an even smaller number of operating systems, this approach was sufficient. However, as the matrix of *Computer models to test* X *Operating systems to test* grew; it became increasingly apparent that performing the testing manually would eventually become both overly complicated and time-consuming.

Because of the problem mentioned above, a three-tier solution was proposed:

- Tier 1 concerned the planning and management of functional tests and their results.
- Tier 2 concerned the coordination of functional tests and test assets.
- Tier 3 concerned the execution of functional tests.

While Tier 1 was handled internally, it was decided that Tier 2 and Tier 3 would be good candidates for Master's Thesis projects.

1.2 Scope

This section presents the scope of the thesis by describing the purpose of the project and the delimitations.

1.2.1 Purpose

The purpose of this thesis project is to ease the transition from manual testing to automated testing. If the purpose is fulfilled, the company will be able to run more functional tests automatically. Of the three tiers mentioned above, this thesis aims to provide a solution to Tier 2.

1.2.2 Limitations and Delimitations

A limitation of the project is that it has not covered all parts of a transition to automated testing. Tier 3 is described in another Master's Thesis report which covers implementing test scripts and the test executing tool to be run on the test objects. Tier 1 is developed by Ericsson employees, which means that the design and implementation of the final database for storing test results is not part of the thesis work and neither is the implementation of the web based system for test result management.

The project description mentioned one delimitation of the web interface to be developed by this thesis project. It does not have to support web browsers other than Mozilla Firefox 3.0 and Microsoft Internet Explorer 7.0.

1.3 Problem Definition

To handle the responsibilities assigned to Tier 2, it needs to provide a complete work-flow starting with locating a test to execute and an asset to execute it on, ensuring said test is run on said asset and storing the result of the test where it can be studied at a later time.

The list below will summarize the functionality that the tier must implement:

- The tier must hold information on the test assets and their properties
- Users must be able to locate (search for) the assets they need to perform their test
- Users must be able to locate tests that are to be run automatically
- The tier must allow users to match the above mentioned tests and assets with the intent of scheduling and executing the tests on the assets
- The tier must pass the results from the executing tier (Tier 3) to the result tracking tier (Tier 1)

Some further requirements that were elicited in the pre-study phase (see section 3.1) are described below:

- The tier must enforce the policy that certain assets can only be used for certain types of testing[1]
- The tier must track and display the availability and correctness of both the test assets and the sub-systems in Tier 3, to allow rapid discovery of errors

While the requirements mentioned above are all important for the tier (and the whole system) to work, they are not in any way unique to this thesis. Automated testing has been discussed at length, from economic perspectives (see Ramler et al. [1]), from development perspectives (see Artho et al. [2]) and from automation perspectives (see Bouquet et al. [3]).

To allow us to hopefully contribute something new to the area, the following questions were put as the focus of the project:

Question 1: How can we allow for a flexible transition from a mostly manual testing to a mostly automated testing?

With the large number of different tests, computer models and broadband modules that are in play today, going from manual to automated testing in one single step is not a viable option. Also, the software and hardware the tests are executed on are not all currently suited for automation, but since work is underway to fix this, the system must allow for a gradual increase of the test automation.

Question 2: How can we implement and deploy a highly usable system with a minimal need for re-training and re-structuring?

While every system has its flaws, simply throwing everything away and starting from scratch can be time-consuming both for the developers and the users. The new system should carefully consider each part of the old system to find what can be reused. This ranges all the way from simply using the same terminology to actually integrating parts of the old system into the new.

[1] E.g. a computer that has once been in an environmental test (for instance subject to excessive heat) may never again be part of a conformance test (for instance making sure the radio amplitude is within bounds).

Question 3: How can we ensure that the system implemented can handle changes in the environment with minimal or no need for re-implementation?

The area of mobile broadband modules is in no way stagnant. New types of devices and new functionality are announced all the time. The system must thus be able to handle both new types of devices to be tested, an increase in the number of tests executed and new types of tests. Also, a rise in the production numbers would likely lead to an increase in the number of developers/testers, and thus an increase in the number of users of the system, so the system must be able to handle that as well.

1.4 Related Work

While neither the area of functional testing nor the area of test automation are new, there seems to be a very limited amount of information available on automating functional testing of software drivers for computer accessories. Berner et al. [4] describes their observations from implementing test automation on software projects but focuses on parts of the testing solution outside the scope of this project such as organizational structuring and strategies. Cai et al. [5] covers test automation on kernel code but uses virtualization to perform the tests rather than dedicated hardware. Edwards [6] discusses a framework for automated testing of software, but only in a linear scope, one application, one test. Elssamadisy et al. [7] covers the area of test automation very well but, as a side effect, spends very little time on the subject of test schedulers.

Most of the software testing tools used today do not work well with database applications since the state of the database changes during test execution. The problem is even harder when parallel[2] testing is to be executed, as Haftmann et al. explains in [8]. In the same article it is explained that a global scheduler can be used to coordinate the execution of tests to run. Each machine can have a local scheduler which communicates with the global scheduler and coordinates the local test run execution. This is different from the work described in this report.

There are likely companies releasing, similar computer accessories that perform automated testing, but no writing on this effort has been found.

1.5 Methodology

The methodology used to plan, design, implement and evaluate the system with a focus on the three questions asked in section 1.3 was a mix of the traditional waterfall principle and the, both praised and criticized, Scrum method. The waterfall principle, described in [9], generally divides the work into the following steps: Requirement elicitation, design, implementation, testing/verification and maintenance. The Scrum method enforces a more powerful method to structure with the goal of keeping the focus on what needs to be done, and a clear division of work during a software development project, as described in [10]. A to-do list influenced by the Scrum method was used throughout the project.

The pre-study phase contained literature studies, observations of the current system and work flow, planning the work and creating a time plan and gathering requirements. The requirements were collected through interviews and discussions with supervisors and potential users. A specification of requirements was written to present measurable goals. The system was divided into sub-systems which finished the pre-study phase.

[2] Parallel testing is when tests are executed on more than one machine and executed several test runs concurrently on each machine.

The sub-systems had individual design and implementation phases. The design phase included work with UML[3] in terms of defining user roles, writing use cases and drawing sequence diagrams, further described by Blaha et al. in [11]. In this phase, a domain model of the domain in which the system would be deployed was also created.

The implementation phase of each sub-system was followed by a testing and verification phase to ensure that they worked properly. Thereafter came an integration phase where the sub-systems were integrated together as well as integrated with surrounding systems. The project was finished by a testing and verification phase of the integrated system.

1.6 Reading Guidelines

This report consists of 11 chapters and a set of appendices. Chapter 1 gives the background to the thesis as well as the scope and the motivation to why this thesis work is done. It also describes the problem, the method used to solve it and related work done by other people. Chapter 2 contains the theory behind the areas of automated testing and software development used in this thesis work. Chapter 3 describes the pre-study-phase and the system design phase. Chapter 4, 5, 6 and 7 describes the design and implementation of the different subsystems. Chapter 8 discusses the outcome of the project. Chapter 9 concludes the entire report and mentions possible system extensions. Chapter 10 lists the references we have used in this work. Chapter 11 briefly describes the appendices available in this document.

Due to the project type and the nature of the research questions, this report has no dedicated result section. For the result of the implementation see chapters 4 through 7 and for the answers to the research questions see chapter 9.

[3] Unified Modeling Language

2 Theory

This chapter offers some of the underlying theory for technologies used and decisions made for this project. The topics covered are: Automation of testing, web application frameworks and data interchange formats.

2.1 Automation of Testing

The purpose of automating testing

The purpose of automating testing is usually to produce a comprehensive automated test system that executes tests quickly and continuously while producing useful feedback to the user who ordered the test. For instance, the feedback could be notifications to the user when a test fails and when the test execution is done. If the result is logged during the execution, these logs can be stored at a common shared file location where the users easily can access the files and analyze the result.

Benefits with automated testing

There are plenty of benefits with well-working automated testing compared to manual testing. The productivity increases and the level of test coverage become higher. A computer neither needs breaks nor complains about working 24 hours a day meaning more tests can be executed in a shorter period of time. Also, the computer is more likely to perform the same test exactly the same way when performing it a second time. Berner et al. [4] mention that organizations often expect a very short ROI[4] on their test automation investment. They also bring up two strong points for automating tests: "*Automated testing does allow for shorter release cycles*" and "*The quality and depth of test cases increase considerably, when testers are freed from boring and repetitive tasks*". By moving to automated testing the number of re-runs of the test cases can be taken to a higher level. Also, by storing the automatically generated test results in a well structured way, the possibility is given to generate useful test reports and test statistics.

The possibility to reuse test code

Using automated testing can be very useful in organizations where several products need to be tested in parallel. The fact that tests tend to be repetitive gives the opportunity to reuse code. Another opportunity is to reuse utilities and drivers. In other words, it is likely that test code used in one project can also be reused in other projects.

Still, as is the case with most concepts, automation of testing is not the universal solution for all types of testing. It requires some work to implement, it is not suited for all types of testing and the test automation implementation must be tested itself to guarantee reliable results. These issues will be described in the following paragraphs.

Evaluate testability – Automated testing is not suitable for all kind for testing

It is also important to remember that not all kinds of testing are suitable for automation. Some pieces of functionality are easier to test manually and not worth the investment of time in automated functional testing. For instance, GUI[5] testing is usually complicated to automate. The fact that a GUI usually has a large input space compared to e.g. a console application makes it very challenging to develop automated testing in this area. It can be concluded that automated testing cannot replace manual testing completely, which is also discussed by Berner et al. in [4].

[4] Return of Investment

[5] Graphical User Interface

Testability is something that is a usually forgotten non-functional requirement. In the design of the test environment three problems are rather common: The first one is manual installation and configuration procedure for the SUT[6]. The second one is lack of access to essential infrastructure like the configuration management system. The third one is a lack of observability of the text execution in certain types of test automation. These problems are mentioned by Berner et al. in [4].

Automation of testing is not a simple task

The automation of testing is usually not a simple process. Berner et al. say in [4] that a required condition to succeed with automated testing is to have an appropriate test automation strategy. They also point out that it is a good idea to start in small steps and automate simple test cases first and then expand from there. The transition to automated testing can be very time-consuming and expensive, especially in the beginning of the process. Also, it can be hard to maintain the testware[7] on the SUTs, because it has to be modified for each new release. A common mistake is that the architecture of the testware is not well documented because it is not treated as a real software project.

The test code needs to be tested

It is important not to forget that the test code used in the automated testing also needs to be tested. Berner et al. [4] points out how important it is to verify the test code by saying "*Any software utility or reusable component built for test automation should be tested at least superficially with an automated test suite. Test cases should be forced to fail at least once, in order to make sure it doesn't[sic] miss the point it was designed for.*"". If the verification phase is not performed properly, the money and effort spent on the automated testing investment might be a waste because it results in an unreliable testing tool.

Steps in automated testing

The automation of testing can be divided in several steps that will be described further in the following sections:

- Generate the test cases
- Manage the test queue
- Execute the test files
- Store the test results

2.1.1 Generate Test Cases

To be able to automate any type of testing the manual test specifications must be re-implemented as test case scripts. The test cases have to be written as specifications that can be parsed and executed by a computer program. These test scripts can be specified in XML[8] format which is recommended by Johnson et al in [12]. Each test script can represent one or more scenarios to test. The scripts can be formulated to include parameter data, functional operations and information to be able to validate the results.

The test scripts can be generated automatically if the test environment supports this. On the other hand, the testers can be taught how to generate their own test cases manually and then share the scripts with each other to extend the test script bank. If the regular testers should be able to generate test scripts the process needs to be simple even if the testers must have the competence for it. It might be necessary to "approve" all generated tests before starting to use them, to make sure they are correct.

[6] System Under Test

[7] A collective name for everything needed for automated testing i.e. test scripts, test drivers, input parameters and expected output etc

[8] eXtensible Markup Language

2.1.2 Manage the Test Queue

As soon as there are more than one test to be run a test queue is needed to make sure the tests are executed in the right order. The test queue can be as simple as a single first-in-first-out queue implementation. Since this kind of queue is inefficient when it comes to test execution on several assets in parallel, a more complex queue implementation might be needed. In cases where several tests need to be run on the same asset (not in parallel), it is useful to have one queue instance for each asset.

To achieve a more flexible automated testing, the test case prioritization technique described by Rothermel et al. in [13] can be used. The prioritization technique enforces an execution order that ensures that the tests with the highest priority, according to some criteria, are executed earlier in the test process than lower priority test cases. The number of different prioritization levels can be adjusted to fit the test environment.

The test queue needs to be managed by a scheduler to execute tests in the right order. In [4] Haftman et al. describes an approach of test execution automation where a global scheduler is used to coordinate the whole test run execution and a local scheduler is installed on each machine. The local scheduler communicates with the global scheduler and coordinates the local test run execution. A more complex test queue requires a more complex scheduler.

2.1.3 Execute the Test Files

To be able to perform automated testing a test execution tool that can interpret the test scripts is required as described by Levesque et al. [14]. The tool must be able to parse the operations in the test case into steps of program code that can be executed on a SUT. Each step must have an expected outcome to allow the testing tool to be able to decide if the test step has passed or failed. The test executor has to put the SUT and the test system into appropriate initial states to be ready to execute the test automatically. Furthermore, the test executor needs to communicate with the scheduler and the SUT.

2.1.4 Store the Test Results

When executing a test the automated test tool must be able to validate the result and decide if the test case has passed or failed. If the test case has failed, it is important that the result contains information about what kind of failure, the reason for it and information about how to re-generate the error that caused the failure.

To achieve the benefits of automated testing the results need to be stored somewhere in a well-organized structure. The results can be stored in a database, an excel spread sheet or similar. A database is be recommended by Randal Root et al. in [15]. By keeping track of the results, statistics can be calculated and trends can be detected in an early stage.

2.2 Web Application Frameworks

"A web application framework is a software framework that is designed to support the development of dynamic websites, web applications and web service. The framework aims to alleviate the overhead associated with common activities used in web development."

-Wikipedia[17]

Most projects that implement a system with some sort of web front-end will have to deal with the issue of choosing a web framework. Doing an exhaustive review of all available frameworks may become an overwhelming task. Vosloo et al. [16] lists 80 server-centric web frameworks, Wikipedia [18] lists 92. Instead of embarking on a long journey to describe all these frameworks, three well-known frameworks using different programming languages and techniques will be singled out and summarily described below.

2.2.1 CakePHP

CakePHP [19] is an MVC[9] framework written in PHP[10]. The development of the framework is backed by the Cake Software Foundation, a non-profit corporation with the goal of promoting CakePHP development. PHP, in difference to the two frameworks mentioned below, is typically not compiled, but executed by an interpreter when the page containing the PHP code is requested by a visitor. PHP code can be interleaved with HTML[11] or used in PHP-only files. CakePHP supplies an AJAX[12] library to help developers with creating interactive web pages.

2.2.2 JavaServer Faces

Developed by Sun Microsystems, JavaServer Faces (JSF) [20] is a web application framework allowing Java code to be embedded in an otherwise static context, such as HTML pages. JSF uses JavaServer Pages (JSP) as the backing technology. The Java code in the static context is compiled by a special JSP compiler into Java Servlets, which are objects that, described simply, receive a request and generate a response based on that request. Together with the static HTML, this allows for dynamic web pages to be written without generating the whole page from within a servlet. Several external AJAX components exist to enhance the JSF framework.

2.2.3 ASP.NET

ASP.NET [21] is developed by Microsoft and is similar to JavaServer Faces in that it also allows for embedding a programming language into a static context. The two programming languages supported out of the box are Visual Basic and C#. As with CakePHP, some AJAX functionality is included. This functionality can also be extended with third-party libraries, such as the AJAX Control Toolkit described below.

2.2.3.1 The ASP.NET AJAX Control Toolkit

The ASP.NET AJAX Control Toolkit is an open-source project which is built on top of the Microsoft ASP.NET AJAX[13] framework. It provides a powerful infrastructure that allows the user to write reusable, customizable and extensible ASP.NET AJAX controls[14] and extenders[15]to make classical ASP.NET web pages more attractive. The AJAX Control Toolkit also contains plenty of controls that can be used out of the box to create an interactive web experience. In total, the toolkit contains more than 30 controls that enable the user to create rich and interactive web pages. More information about AJAX Control Toolkit can be found on the web, see [22].

2.2.4 Extensions Common to All Web Application Frameworks

All the reviewed web application frameworks, and likely most of the remaining ones, support the inclusion of JavaScript into the pages. It is, for instance, a requirement for AJAX functionality. JavaScript can further be used to give the client, i.e. the browser, functionality to update the page based on user interaction without requesting an update from the server.

[9] Model-View-Controller
[10] PHP: Hypertext Preprocessor
[11] HyperText Markup Language
[12] Asynchronous JavaScript and XML
[13] Part of ASP.NET 3.5 and contains cross-platform-compatible pre-built AJAX components that can be included in ASP.NET web pages
[14] A control is an element / component that is used to create web interfaces
[15] An object you attach to an existing control to extend its functionality

2.3 Data Interchange Formats

A data interchange format is a way to encode data when it is sent from one party to another. Three formats commonly used for transferring data over a network are presented below.

2.3.1 Bencode

The Bencode data-interchange format is very light-weight, using token characters and lengths to separate message parts from each other. Unfortunately, it has no standards group behind it, so it lacks both implementation- and documentation-wise.

2.3.2 JSON

JSON[16] [23] is standardized in the IETF RFC 4627[17]. Also, it appears to have a lot of backing and there were more than 100 known implementations in over 30 programming languages at the time of this review. Amongst these implementations were several choices for both C# and Java. The verbosity of the format was very similar to that of Bencode, but the way the message is structured was more similar to that of a programming language (not surprising, considering JSON is a subset of the third edition of the ECMA-262[18] standard, i.e. JavaScript).

2.3.3 XML

XML [24] is arguably the most widely used of the three reviewed formats. It is the base of the new XHTML format, as well as several new document formats and database query languages. Most of the available programming languages have support built-in for XML serialization, among these languages are C# and Java. While the verbosity of XML can be optimized to approach that of Bencode and JSON, the default structure is more verbose than the two other formats.

[16] JavaScript Object Notation Format

[17] Internet Engineering Task Force: Request For Comments no. 4627

[18] An international standards organization for Information Communication Technology and Consumer Electronics

3 Pre-Study and System Design

This chapter describes the work and choices that were made during the pre-study and design phases of the project. The design described here concerns the whole system. Further design choices only affecting one sub-system are described in the corresponding sub-system chapter.

3.1 Requirements Elicitation

A set of non-measurable requirements were listed in the thesis description that was handed to the project members as an initial description of the project goals. These requirements were discussed with the supervisors to be clarified and re-formulated into measurable requirements as a first step in the requirement elicitation process. Two requirements that were very clear from the beginning were that the system needed to use a database for information storage and provide a web interface for user interaction.

The list of requirements was extended by conducting interviews with the supervisors and some of the potential users. The questionnaire used in, and the outcome of, the interviews are discussed in section 3.1.2.

Specification of requirements

All requirements were written in a document, called the specification of requirements, to define which tasks the system should be able to perform before it was deemed to be completed. Some modifications of the requirements were made during the project. The specification of requirements is shown in Appendix A.

Wish list

To be able to finish the Master's Thesis work in time and not lose focus, suggestions to improve the system that were not necessary for the main functionality were placed in a wish list. The wished-for functionality was to be implemented if there was any time left in the end of the project. The wish list is shown in Appendix B.

3.1.1 Observations of the Existing System

The existing system, which was basically a set of mutually agreed-on rules for how the manual testing was to be executed, used several Excel spreadsheets administrated by a couple of persons to keep track of the assets and their current configuration, status and location etc. Using spreadsheets for this way of working had several disadvantages:

- The spreadsheets did not contain information about all assets at the department
- The spreadsheets were hard to keep updated with the correct information
- The spreadsheets did not support to keep track of the history of an asset i.e. where it had been before and what test had been run on it etc.
- The spreadsheets could not be edited simultaneously by several users

A requirement of the new system is that it must use a database to store all its information. A database as central storage has many features that make disadvantages of the spreadsheets mentioned above a nonissue. For instance, it supports more built-in validation and error-checks[19] of the data before it is stored in the database and it is easier to keep track of history when using a database. Thanks to database transactions, it is possible to have several users changing the content of the database simultaneously.

[19] Examples of validation and error-checks performed are 1. Making sure an item that is referenced already exists and 2. Making sure values specified as unique really are unique and 3. Enforcing that something of the type DateTime actually contains a date+time value (or NULL)

Regarding the current method used for performing the functional testing, one of the first tasks the project members performed was a test session using the existing method (briefly described in the background in section 1.1 but reiterated here for clarity). The testing was executed manually and consisted of reading from a test specification that contain the prerequisites for the test, the actions that were to be performed during the test and the expected result of each action.

3.1.2 Questionnaire and Interviews

A questionnaire (see Appendix D) consisting of 10 questions was used for individual interviews with some of the potential users. To get as much as possible out of the interviews, the questions were carefully formulated to elicit descriptive answers. The questions covered the following areas:

- Description of the current system, what works well? What works less well?
- Thoughts and ideas about the new system to be developed? Expectations?
- How much will the new system be used?

Five employees were selected for the interviews. They had different roles in the department and would use the system in different ways to suit their tasks. A couple of them were verification engineers and one of them represented the conformance team. The mix of roles was used to catch different points of view on the system to be developed.

The questionnaire was divided into two parts. The first part dealt with questions about the inventory and booking part of the web interface while the second part dealt with the test scheduling part.

3.1.2.1 First Part of Questionnaire – Focus on Inventory and Booking

The first part of the questionnaire focused on how assets and bookings of assets should be handled. A summary of the answers from the interviews gave hints about what features to focus on. To mention some of these focuses: Make the web interface to the new system user friendly and show updated and correct information. It is also important to show useful information in the search result for assets. Overall the planned search functionality is what will probably be used the most. Other frequently used functions would be the booking functionality, the check in/out functions[20] and the schedule tests.

On the question regarding what to show when the user first uses the system, the most useful information would be personal information for the logged in user i.e. bookings and checked out assets etc. A way to get to the search functionality quickly was also requested.

3.1.2.2 Second Part of Questionnaire – Focus on Testing

The second part of the questionnaire focused on the manual work with functional testing. The main problems of the current system and work flow are that information is missing, it is very time consuming to run the test manually and there is no good way to keep track of the test result history.

The most useful functionality in the new system would probably be the possibility to schedule and run test remotely on an asset, run several tests in parallel on different assets and easy access to the test results and assets history. Ideas about making the test status and results available on a web page were suggested. Regarding the feedback from the system, one idea was that it should be possible to choose if you want to be notified by e-mail and/or SMS. The test result message could then include information about what test was executed, and on which asset, if the test has passed or failed (with the reason why), and a link to the result log.

[20] Used by the system to see which user has which assets

3.1.2.3 Summary of the Interview Answers

Overall the interviews gave a very good view of the problems in the current system, especially the fact that the system had lack of traceability regarding asset location, status and information and that it had a hard time showing what assets belong together as an asset family[21].

There were lots of great ideas about the new system. Many of the people that were interviewed pointed out how important it is to have a system that works well, is easy to access and suits the current work flow. It also became apparent that the current administrators saw a possible risk in that users might not feel required to go through the trouble of using the system but rather circumvent it by, for instance, grabbing an asset to test on and walking out without first notifying the system that they are taking it.

Based on the interview answers it seemed like the whole system would probably be used several times a day by some users and a few times a week by other users depending on their role.

3.2 User Roles and Usage Scenarios

The system supports different ways of using it to better suit the requirements for different user roles. Three types of users are defined. These are registered users, unregistered users and administrators. The unregistered users are called guests. To avoid damage to the database, a set of database accounts with more or fewer permissions were used to separate the user roles (these database accounts are described in section 4.5.2.1).

The usage scenarios that describe the normal work flow including the alternatives and the exceptions are presented as a set of use cases following the UML standard. A summary of these can be found in section 3.2.2.

3.2.1 User Roles

This section describes the three user roles defined in the system. These roles are not supposed to use the system the same way and do not have the same access rights in the web interface. The limitations of access in the web interface are described in the following sections.

3.2.1.1 Administrator

An administrator should be able to maintain the content in a database by using the administrative interface. An administrator should be able to add/edit/delete users, assets, asset properties etc. as well as check on the current status of the system.

3.2.1.2 Registered User

A registered user is someone who may need to search for information about assets and other data in the database, make/manage bookings of assets, enqueue tests on assets that are part of the test bed and check in/out assets. To be able to use the system the user needs to have a user account in the system. The user will use the web interface to interact with the system in a way that allows him/her to rapidly find information about assets, search for available assets, book assets, and enqueue tests on assets etc. A regular user does not have access to the administration interface.

3.2.1.3 Guest

People that are not registered users in the system are treated as guests. A guest has very limited access to the web interface. If a guest browses any of the web interface pages, she/he will be redirected to the guest page. This page explains that the guest needs to contact an administrator to become a registered user before she/he is allowed to use the system.

[21] An asset family is a group of assets that belongs together e.g. a laptop and a charger

3.2.2 Usage Scenarios

Based on interviews, discussions with supervisors and the developers own ideas the most common usage scenarios were formulated. The usage scenarios were presented as a set of use cases that follow the UML standard. The use cases were written to show the normal work flow together with the most common alternative and exception flows in detail. The implementation of the web interface was based on these use cases. The 15 use cases, for regular users and administrators, are briefly described below. The full descriptions can be found in Appendix C.

3.2.2.1 UC1: Logging in

The user has a valid Ericsson id (signum[22]) and is registered as a user of the system. The information about the user is stored in the database. The user logs in to his/her computer and visits the web interface. The web interface matches the domain user name of the visiting user to its own database to find the user details. The user is shown the start page.

3.2.2.2 UC2: Searching for Assets

The user is logged in and goes to the search page and enters a set of search criteria. The web interface lists all the assets matching said criteria.

3.2.2.3 UC3: Managing Search Results

The user has logged in and entered the search page where she/he has performed a search. The user manages the search result by selecting, sorting or requesting details of an asset.

3.2.2.4 UC4: Booking Assets

The user has selected one or more assets to book, either by attempting to check them out or by searching on the search page. The user enters purpose, start and end date and time for the booking. If the booking is possible and placed successfully, the system records details on the booking in the database. If not, the user is shown a message detailing what went wrong.

3.2.2.5 UC5: Managing SQL Queries

The user is logged in and browses to the SQL[23] search page. The user performs the actions: execute, save, edit and remove on SQL queries.

3.2.2.6 UC6: Looking at Asset Details and History

The user has logged in and wants detailed information on an asset. The user has been given a link to the detail page of the asset through any of the other pages i.e. start page, search page etc. The system will display information on the asset.

3.2.2.7 UC7: Checking Out Assets

The user is logged in and wants to check out an asset. The user selects an asset to check out via the start page or the check out page. The check out page shows all assets included in the booking (if there is any booking). If the asset was part of a booking, ready to check out, the booking status is set to active. If the asset is available, the user will be sent to the booking page and the asset will be checked out automatically when a booking is placed.

3.2.2.8 UC8: Checking In Assets

The user is logged in and wants to check in an asset. The user scans or enters the barcode or marking of the asset in the text field on the check in page. The system displays a page with all the required check-in-information and ends the booking of the asset by that user.

[22] A unique identifier given to all Ericsson employees

[23] Structured Query Language

3.2.2.9 UC9: Exporting Search Results to Excel

The user is logged in and has searched for some data in the database. If the search page has been used, the system generates, on request, an Excel document containing all asset and property data on the assets that are part of the search result. If the user has made the search on the SQL search page, the data can be exported to Excel as well by pressing the export button.

3.2.2.10 UC10: Updating the Status of an Asset

The user is logged in and updates the status of an asset that she/he has checked out. The updated information is stored in the database.

3.2.2.11 UC11: Extending the Length of a Booking

The user is logged in and has an asset that is currently in a booking placed by that user. The user uses the link on the start page to the booking page and the system allows the user to enter a new booking end date and stores it into the database, if no collisions with other bookings were found.

3.2.2.12 UC12: Scheduling Automated Test on Asset

The user is logged in and has searched for assets on the search page. The asset that the test will run on is part of the test bed and set to active. The user enters test information and schedules what test to be run on the selected asset. The user can select test from a pre-defined list of test or type in a path to its own test file.

3.2.2.13 UC13: Adding a New Record in the Database (Administrator only)

An administrator is logged in. She/he adds a new record of any of the types: asset, asset type, asset type group, manufacturer, service type, asset owner, user, user group or category to the database via the administration interface. The type of record to add is selected in a list containing a set of tables in the database. The information to be added is then entered into the form and submitted. In some cases several database tables will be affected when the information about a new item is stored. For instance when a new asset is added, information is stored in both the table for assets and the table for holding information about properties.

3.2.2.14 UC14: Editing/Removing a Record in the Database (Administrator only)

An administrator edits or removes a record of any of the types: asset, asset type, asset type group, manufacturer, service type, asset owner, user, user group or category in the database via the administration interface. To be able to remove a record some requirements, such as the item not being currently used by the system, have to be fulfilled. The administrator can put an asset in edit mode by using the edit link on the asset details page or in the administration interface directly.

3.2.2.15 UC15: Receiving an Email about a Long Booking (Administrator only)

An administrator receives an email about a booking that is longer than the specified maximum duration. The email contains information about the booking i.e. the purpose, duration and who is responsible for the booking. The administrator has the possibility to cancel the booking via an attached link.

3.3 Distributing the Functionality into Sub-Systems

In the problem definition (see section 1.3) some of the functionality needed in Tier 2 was presented. It was decided early on that, to increase modularity and maintainability, the functionality of Tier 2 would be distributed into several sub-systems with each sub-system being implemented in a separate application (or similar).

Work was done to categorize the functionality in a clear and logical way to ease the understanding and implementation of the system. After careful consideration, three sub-systems were proposed: TEA, TIME and TEATIMEDB (described later in this chapter). In addition to these sub-systems, a protocol, TEACUP, was designed to enable communication with Tier 3.

The division into sub-systems of Tier 1 and Tier 3 were not done as part of this project, but will be briefly mentioned here since Tier 2 interacts with the two other tiers.

Tier 3 was divided into a test execution manager, called "TEXAS Manager" and a test execution service, called "TEXAS Daemon". The manager could connect to a single instance of the service locally or remotely, using TCP sockets, to specify and control the test execution. The daemon was implemented both for Windows and Linux based operating systems. When using test coordination, TEA controls multiple instances of the TEXAS Manager, each controlling a single instance of the TEXAS Daemon. The TEXAS Manager and TEXAS Daemon use an extended version of the TEACUP protocol for communication. Relating to the theory section on the automation of testing, the combined TEXAS system is responsible for the parts of parsing the test scripts and executing the test (see section 2.1.3)

Tier 1 was divided into a web interface and a database, neither of which had a formally agreed-on name at the time of publication, but were referred to as the "Planning and result web interface" and "Planning and result database" respectively. Both the TIME and TEA subsystems of Tier 2 interact with the database.

The tier and sub-system distribution can be seen in Figure 1.

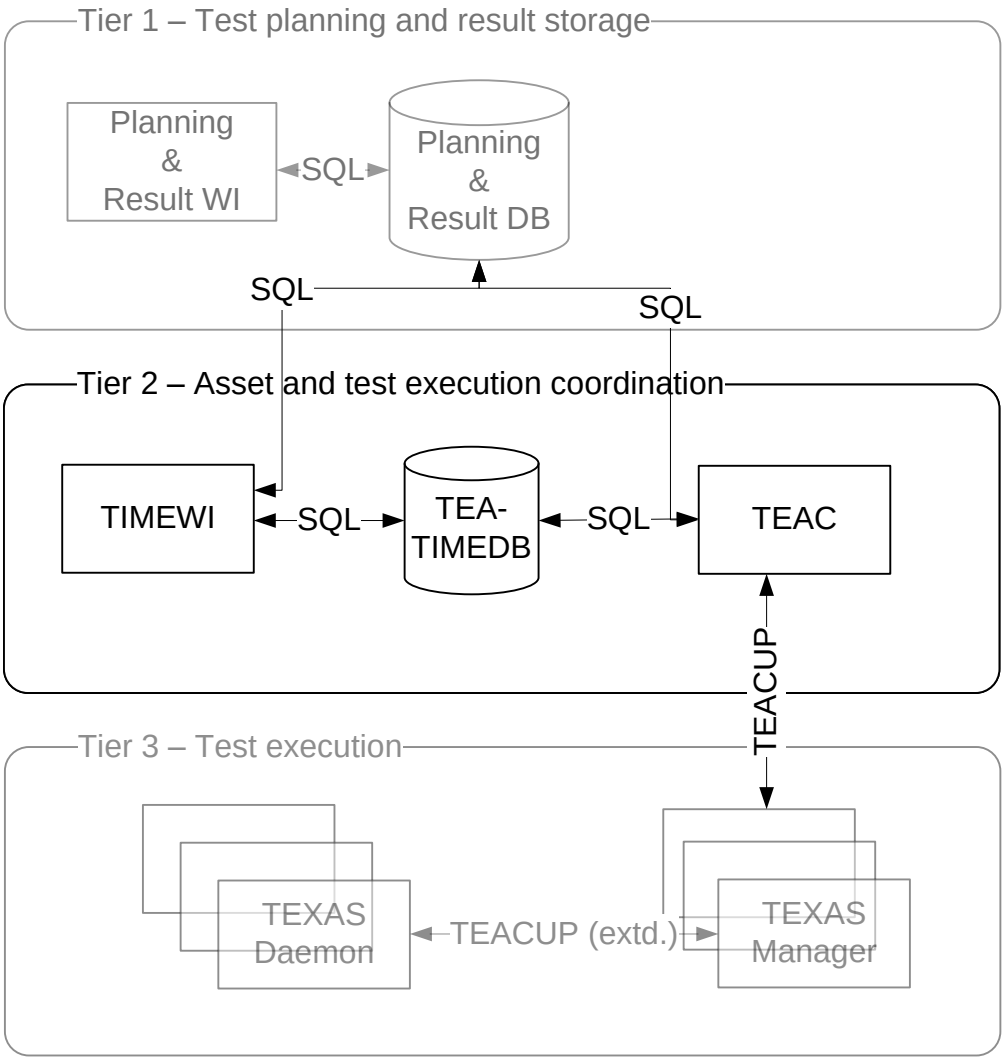


Figure 1: An overview of sub-systems in tiers 1 through 3

The remainder of this sub-section gives brief descriptions of the proposed sub-systems/protocols and the choices that were made during their design phase.

3.3.1 TEA – Test Execution Automation

TEA is implemented in the TEA-Coordinator (TEAC or "coordinator"). Its main responsibilities are listed below:

- Fetch test specifications from TEATIMEDB
- Execute the tests on assets using Tier 3
- Proxy test results from Tier 3 to Tier 1
- Inform users on test execution completion

The TEA-Coordinator keeps track of running tests and fetches information about the tests in queue, which asset to be run on, what test files to run and number of iterations etc. The results of the executed tests are stored in a separate database. Initially the results were stored into the Temporary Result database, described in section 3.3.1.1 which was later replaced by the final Planning and Result Database described in section 3.3. The TEA-Coordinator keeps track of the test queue and launches a new instance of TEXAS Manager for each test to execute. For details on the implementation of the test coordination, see section 5.2.1.

3.3.1.1 Temporary Result Database

A small, simple database was designed to store the result from the test executions. It was temporary and only to be able to test the TEA-Coordinator and the implemented result database handler. The temporary result database was then used as a base for the final Planning and Result Database in Tier 1, which was designed later on. The design of the Tier 1 planning database was not within the scope of the Master's Thesis work.

3.3.2 TIME – Tracking Inventory Manager

TIME is implemented in the TIME Web Interface (TIMEWI or "web interface") and the physical check-in/out station. Its main responsibilities are listed below:

- Searching for information stored in the TEATIMEDB database
- Showing information stored in the TEATIMEDB database
- Booking, checking in/out of assets
- Scheduling of automated tests
- Administration the data stored in the TEATIMEDB database
- Communication with TEA via the TEATIMEDB database

The TIME Web Interface works as an interface between the users and the TEATIMEDB database. For details on the implementation of the web interface, see chapter 4.

The physical check-in/out station supports the web interface by allowing quick check in and out of booked assets. It is described in section 3.3.4.

3.3.3 TEATIMEDB – Asset Tracking and Test Scheduling Database

TEATIMEDB is the designed database for asset tracking and test scheduling. Its main responsibilities are listed below:

- Storing information
    - Holding information on tests such as which test scripts they contain and what position they have in the queue, together with general test information
    - Holding information about assets and surrounding items
    - Supporting generic properties of assets
- Working as an interface between the TEA and TIME sub-systems

The TEATIMEDB database stores all necessary information for the system and also allows the other sub-systems to interact with each other. For details on the design of the database, see chapter 6.

3.3.4 The Check-In/Out Station

A physical station for check-in and out of booked assets was to be installed and configured. Its main responsibilities are listed below:

- Primary location for a label writer, which is needed to print the unique bar code labels needed to show that the asset is part of the TEATIME system
- Primary location for a bar code reader, which is needed to scan the labels[24]

The station will support simple and quick check in and out of assets. The installation and configuration of the station is described in section 4.3.11.

The choice of bar code reader and label writer are described one by one in the following sections.

3.3.4.1 Bar Code Reader

There are several bar code readers to choose from on the market. They range both in price and functionality which had to be taken into consideration to be able to find a suitable one. Some requirements were decided on for the bar code reader. These requirements are listed below:

- Minimum readable bar code height: 3 mm
- Support the standard bar code types i.e. Code39, EAN-128 etc.
- USB connection
- Reasonable price (1000 – 2000 SEK)
- The seller must have a contract with Ericsson

After finding bar code readers that fulfilled the requirements above, there were two other choices to make. The first choice was to decide if it should have a cable or be portable. The cable model has the advantages that it is cheaper and it will stay at the station. On the other hand, it has a disadvantage in that it is less flexible to use if the cable is short. The second choice was to choose between laser and CCD technology. Laser has the advantages that it can read the code further away from the label and it is faster, but it is much more expensive than a reader using CCD technology.

The final choice

The final choice of the bar code reader fulfills the listed requirements in the previous section and uses laser technology. The reader has a cable and is connected to the computer via either USB or PS2. The brand is DELTACO and the model is called DUR 801, see Figure 2.

Figure 2: Final choice of bar code reader, a DUR 801

[24] Every asset is given both a unique bar code (printed on a label) and a unique marking that can be used for identification

3.3.4.2 Label Writer

Similar to the bar code readers, there are several label writers with different properties and prices available on the market. A list of requirements was defined for the label writer to fulfill. These requirements are listed below:

- Label size requirement to fit on certain assets: 2 x 5 cm
- Support the standard bar code types i.e. Code39, EAN-128 etc.
- USB connection
- Reasonable price (1000 – 2000 SEK)
- The seller must have a contract with Ericsson

The final choice

Different label writers were evaluated and after finding a couple that fulfilled the requirements above the final choice became a DYMO Label Writer LW400, see Figure 3. This printer had a great advantage because it was delivered with the DYMO Label Software SDK[25]. The SDK included Microsoft Internet Explorer and Mozilla Firefox scripts to be able to print labels using the high-level COM[26] and XPI[27] interfaces.

Figure 3: Final choice of label writer, a DYMO LabelWriter 400

3.3.5 TEACUP – Test Execution Automation, Common User Protocol

This section describes the development and design of the TEACUP protocol.

3.3.5.1 Leveraging Existing Technology

To make the implementation of the protocol simpler, a decision was made to use a pre-existing data-interchange format. After a brief study, the formats Bencode, JSON and XML (see section 2.3) were brought up for review. The review focused on the aspects of verbosity (how much data needs to be sent to convey a specific message) and ease of implementation.

The decision was made to go with JSON due to the good balance between verbosity and ease of implementation.

[25] Software Development Kit
[26] Component Object Model
[27] Cross-Platform Install

3.3.5.2 Designing the Protocol

To elicit the messages that would be needed in the protocol, a set of scenarios were modeled as UML sequence diagrams (see Appendix F) ranging from the simple error-free case to more uncommon cases such as resuming a test after a software crash or a network outage. The elicitation process resulted in a basic set of 30 messages that are common for both the coordinator-to-manager communication and the manager-to-daemon communication. The message set was also extended to support some unique manager-to-daemon functionality.

4 TIMEWI - The Web Interface

This chapter covers the design, implementation and analysis of the TIME web interface, TIMEWI.

4.1 Introduction

The web interface was developed as an interface to the database. This made it possible for users and administrators to store data into the database as well as access the data. The main features of the web interface are:

- Searching for assets or surrounding information
- Scheduling automated tests
- Booking assets
- Checking in/out assets
- Administrating the content in the database

4.2 Application Design

This section covers the design phase for the web interface, ranging from choosing a web application framework to the design of the user interface.

4.2.1 Choosing a Web Application Framework

Three well-known web frameworks were chosen for review; CakePHP (see section 2.2.1), JavaServer Faces (see section 2.2.2) and ASP.NET (see section 2.2.3). Collectively, the project members had previous experience with each of the frameworks, so the need for an exhaustive review was deemed unnecessary. A brief study of the available choices led to ASP.NET coming out on top. A summary of the deciding factors is given below:

- The frameworks for review were found to be functionally equivalent for the purpose of the project
- Both the systems in Tier 1 and Tier 3 would be implemented at least in part in C#. One of the programming languages supported natively by ASP.NET is C#
- One of the project members had positive experiences from working with ASP.NET in an earlier project

From the list above can be concluded that using ASP.NET with C# would not only fit well into the rest of the system, but would also enable the project members to draw on previous experience to minimize the need of spending valuable time on learning a new framework.

4.2.2 Application Architecture

The web interface was intended to be used as a replacement of a current practice, and it was likely that it would be used for quite a long time which put special requirements on the implementation. It had to be easy to change, both by developers familiar with the system as well as newcomers, to support new ways of interaction and usage. Because of this, a decision was taken early in the process to create and maintain a logical application architecture. In short, the architecture can be described as "*Model-View-Controller with localized ASP.NET code-behind and a data access layer*". The remainder of this sub-section will describe the architecture in more detail.

4.2.2.1 Model-View-Controller

The model-view-controller pattern is a software design pattern that proposes a clear separation of responsibilities for presenting and working with data. The pattern suggests dividing the code into three parts:

- The model – A representation of the data that is being worked with
- The view – The user interface that shows the data
- The controller – The functions that manages the data

Model

In the case of the web interface, the model part is a collection of C# classes mostly implemented as a 1-to-1 representation of a table or set of tables in the database. Every column in the database is represented by a property (field) in the class. These models were used whenever some manipulation needed to be done to the data or when some extra layer of functionality was needed before the view could present the data. When the view only needed to show a direct representation of the data from the database, the step of loading the data into the model was often skipped for performance reasons.

View

The view part consists of ASP.NET pages. The pages are typically not related to any database table, rather they are related to a functionality, such as booking or searching.

Controller

A set of classes containing functions to manage the model and provide useful functionality for the view made up the controller part.

4.2.2.2 Localized

Every line of text should be localized[28], which also meant it would not be hard-coded into the view. The actual strings would be stored in a set of resource files with one file for each language localization.

4.2.2.3 ASP.NET Code-Behind

Using code-behind means separating each page in the view into two sub-parts, a static page and a set of functions to fill that page dynamically. The code implementing these functions is stored in a code-behind file, hence the name. The static page is an ASP page where all controls have unique ids. These ids can be referred to by the code-behind functions when populating the page with data.

4.2.2.4 Data Access Layer

As stated above, the controller is responsible for managing the model, meaning both fetching data into the model and facilitating manipulation of that data. In the web interface, the work of fetching data was a data access layer. This layer was implemented in a set of classes containing methods to select a subset of the total data, based on specified criteria, plus methods to load the selected data into the models. Having a data access layer means the rest of the controller does not need to know or care about where the data comes from, meaning it would be easier to switch the data storage facility, for instance from an SQL database to an XML file, should that be necessary.

[28] Language localization, enabling the same content to be represented in several languages

4.2.3 The User Interface

This section covers the guidelines for implementation and the layout for the web interface.

4.2.3.1 Guidelines for Implementation

Sommerville (see [25]) presents a set of principles for user interface design. A summary of these designs are presented below:

A. User familiarity – Use terms and concepts that are known by the users
B. Consistency – Similar operations should behave the same way
C. Minimal surprise – Users should not be surprised by system behavior
D. Recoverability – Users should be assisted in recovering from errors
E. User guidance – Provide meaningful feedback and context-sensitive help
F. User diversity – Give different types of users different tailored ways of interaction

Based on the previously mentioned principles, a set of guidelines for the user interface implementation were set up. Some of these guidelines, together with which of principle(s) above they implement, are listed below:

1. Status messages should always be shown after an operation, regardless of whether that operation was successful or not. Messages indicating an error should always be printed in red. Messages indicating a successful action should always be printed in green. (B,E)
2. Whenever there is a risk of users thinking that performing actions in a certain order would not lead to a conflict, even though it actually would, the interface should either hide or disable the elements that would allow the conflicting action, or display a message detailing why it would conflict. (C)
3. The terminology used for objects in the interface should follow the terminology that was elicited in the interview phase (see 3.1.2). When new terms must be introduced, they should have a question mark beside them which can be hovered to display a description. (A,E)
4. The user and administration parts should use the same layout, but be clearly separated. Each part should be modified to suit the intended way of interaction. (A,F)
5. Whenever there is an error relating to user input in a form, the field that caused the error should be named in the error message. All data that was entered before the error occurred should be restored to the form. (D,E)

Examples on how these guidelines were used in the implementation can be seen in the following sections, as well as in the user guide (see Appendix G).

4.2.3.2 The Layout

The layout of the web interface is based on the one that is used as a general template for all Ericsson Internal pages, specifically the one used on the Ericsson Internal portal page. The heading of the page contains Ericsson logo, the system name, and the name and signum of the user currently logged in. The menu, used for navigation, is placed to the left on the page. The middle part displays the content of the page currently visited. A master page that all the other pages inherit was created to make the web interface more consistent. An example of the layout can be seen in Figure 4.

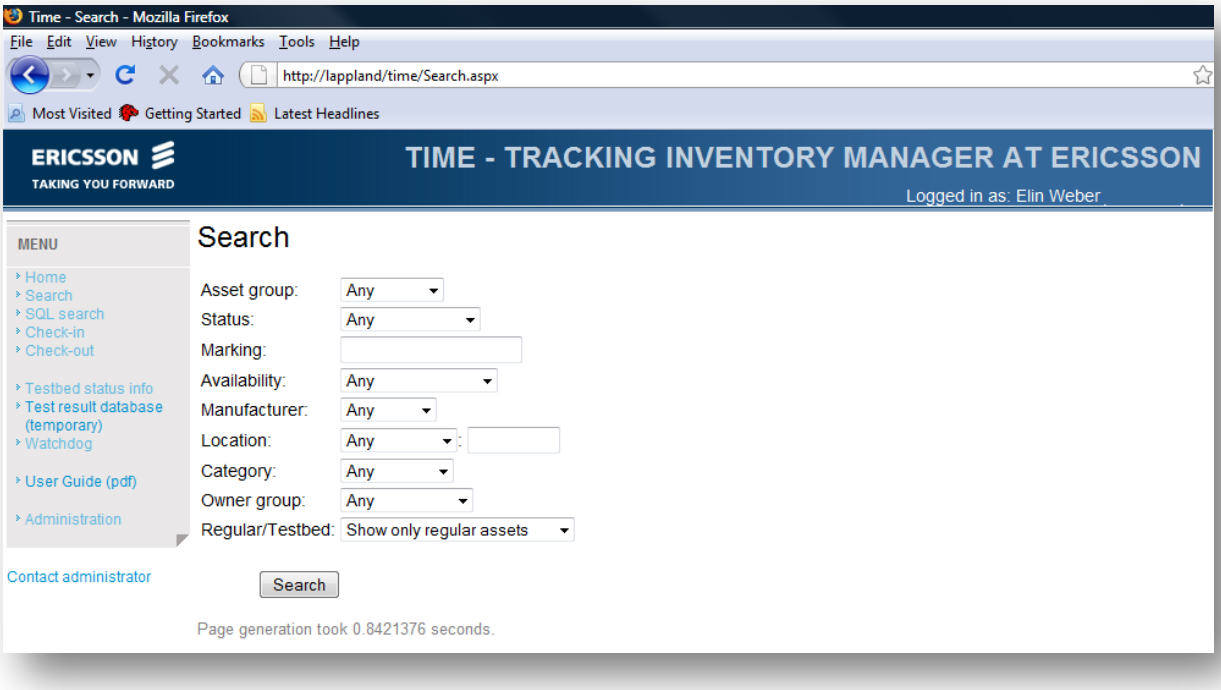


Figure 4: The layout of the web interface

4.3 The Implementation, Part One: User Functionality

Relating to question 2 in the problem definition (see section 1.3), one of the system requirements was to make the system easy to use and very user friendly. Several functions were implemented to achieve this requirement and make the introduction of the new system as painless as possible.

4.3.1 Consistent Layout and Direct Feedback

A clear and consistent layout, described in the design section (see section 4.2.3.2) is used with the intention of providing a familiar feel to the web interface. This is supported by guidelines 1 and 5 in section 4.2.3.1.

Another important feature to make the user comfortable is the visible feedback and quick response from the system. Most of the actions in the web interface give the user feedback about the outcome as soon as it is triggered. Status messages are usually shown on each page after an action i.e. book assets, check in or out assets and store new information in the database, has been triggered. By using such messages the user feels that the system is active and responds to the requests, see section 4.2.3.1, guideline no. 1.

4.3.2 Inline Help and User Guide

As mentioned before there is a user guide (see Appendix G) that describes how to use the system in details. There is a link to this guide in the menu on the web pages. Much effort is put on making the pages such informative that the user does not need to read the guide to find out how to use the system. Also, there is in-line help implemented on the pages. The in-line help is presented as question mark which can be hovered to display a description visible on the pages to make the user more comfortable of how to use the system without having the guide. This is supported by guideline 3 in section 4.2.3.1. One example of in-line help is shown in Figure 5, where the term *Priority* is clarified.

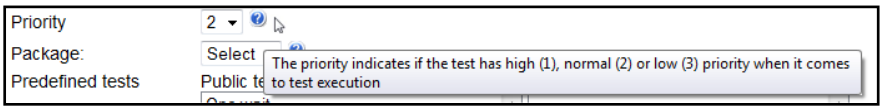


Figure 5: An example of in-line help

4.3.3 Dynamic Content Generation

According to question 3 in the problem definition (see section 1.3) the system to be implemented should be flexible and able to handle changes in the environment with minimal or no need for re-implementation. As described in section 6.3.1 the database supports adding and editing of asset properties of asset type groups during runtime, which also needs to be supported by the web interface, together with generation of dynamic content. Therefore, on the search page, the properties related to the selected asset group are created dynamically to match the data stored in the database. If a new property is added to any of the asset types the system will then add it to the search criteria on the search page. The input field for a multi-valued property is rendered as a drop-down list and for a single-valued property is rendered as a text box.

Another example of dynamic content generation is on the check-out page where all assets part of a booking are listed in a table when a user attempts to check-out one asset part of that booking. Since different bookings include a different total number of assets and asset families, the table needs to be dynamically generated.

The rest of the system supports changes of assets in terms of composition of asset families and whether an asset is part of the test bed or not, which thus also needs to be supported by the web interface. Thus, the search page, test bed info page etc. are affected by these changes and adjusted to the asset modifications.

4.3.4 Other Techniques and Components Used to Improve User Interaction

The well-known components from JavaScript, ASP.NET and the AJAX Control Toolkit, all briefly described in the theory section 2.2.3.1, were used in the web interface to make it more interactive.

Several JavaScript elements were used in the implementation of the web interface. To mention some of them, see the list below:

- The function *setFocus* was used to set the pointer in the text field on the check-in and check-out pages to shorten the process.
- The function *charCounter* was used on text fields for comments and description to keep track of remaining characters. An example of the charCounter function in use on the check-in page is shown in Figure 6.

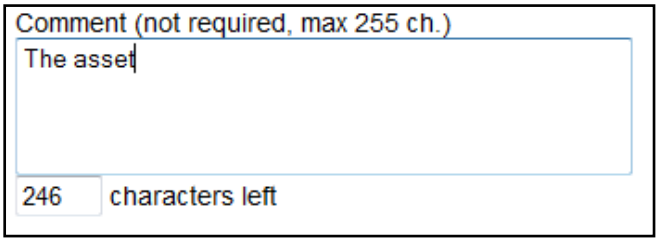


Figure 6: A charCounter for a text field

Many ASP .NET controls were used i.e. *Calendar* on the booking page, *GridViews* to show data on the search pages and in the administrative interface etc. Another frequently used ASP .NET control is the *UpdatePanel* that has been used on almost all pages to avoid page-reloading-related interruptions. Using update panels instead of traditional ASP panels is faster because the AJAX approach minimizes traffic to the server by sending and requesting the minimum amount of data needed, says Garret [26].

A couple of controls from the AJAX Control Toolkit were also used to make the web interface more interactive towards the user. For instance, the *AutoComplete* extender was used on a few pages to help the user to fill in asset markings in text fields faster. An example is shown in Figure 7. The *AutoComplete* extender can be attached to a TextBox control. It will associate that control with a popup panel to display a dropdown list with words that begin with the prefix typed into the textbox.

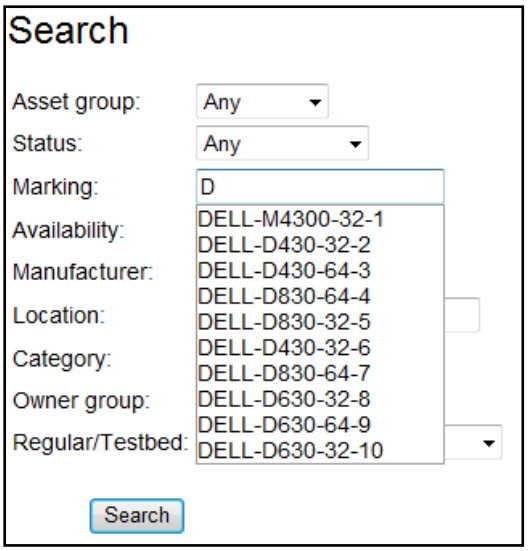


Figure 7: AutoComplete used in a text field for marking

Another AJAX Control Toolkit extension is the *MaskedEditExtender* which is used on the booking page where it assists the user with writing the start and end time in a valid format. An example is shown in Figure 8.

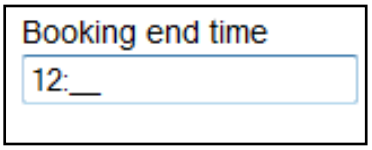


Figure 8: A text field with the MaskedEdit extender

4.3.5 Presenting Context-related Information in a Clear Manner

The terminology brought up in the interviews (see section 3.1.2) was used in the web interface to get the user feeling more familiar to the new system, see guideline 3 in section 4.2.3.1.

To show relevant information where it is needed is important to make the interface useful for the users. The interviews, mentioned before, and discussions with the supervisors was useful when deciding what information to show. The information shown on the start page is specific to the logged in user. The page contains information about the nearest upcoming bookings, most recent check-ins, active bookings and upcoming scheduled automated tests to give the user a quick update of what is going on.

When the user enters the booking page to book one or more assets, their markings and asset groups will be listed at the top of the page, see Figure 9. The assets that are part of the booking are typically of different groups and showing this information gives a simple overview of what the user is really booking.

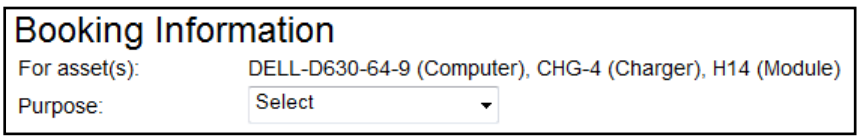


Figure 9: Assets listed with marking and asset group

4.3.6 Allowing Users to Take Short-cuts

On several pages there are links to other pages that the user might be interested in. A good example is the start page which contains personal information regarding bookings and tests, if there is any for the logged in user. On the start page, the assets listed in the tables contains links to the asset details page, the booking page, the check out page, the check in page as well as the possibility to extend active bookings or cancel upcoming bookings. All these links contain the id or marking of the asset, which the receiving page can use to easily look up the asset in question. An example of what the start page might look like for a very active user is shown in Figure 10.

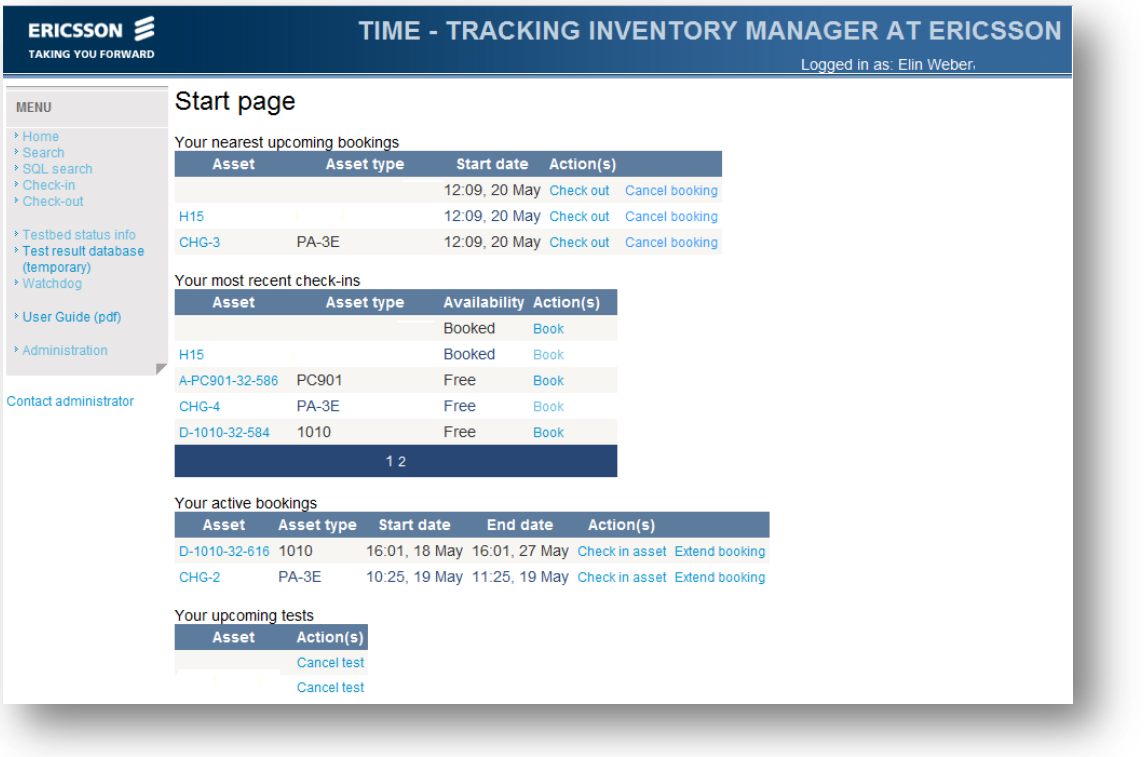


Figure 10: Screenshot of the user start page with links to common actions

Another example is the search page, where all assets in the search result are linked to their asset details page. If the asset is part of an asset family, there will be links to these assets on the details page, making it easy to find the properties for a whole asset family. Also on this page, there are visible links to book the asset and to schedule automated tests on the asset.

On the booking page, when a booking has been placed successfully and the start date is within a configurable amount of hours, there will be a visible link to either the check out page or the auto test page depending on whether the booked asset is part of the test bed or not. All these links mentioned above are there to provide an ease of use for the web interface.

4.3.7 Searching for/Locating Assets

The system provides two ways of searching the data. The search page is meant to be used for assisted searching of assets while the SQL search page is used for more complex searches, or searches of information not directly related to assets. See the user guide in Appendix G for more information about how to use the search pages.

The user can easily find available assets of a certain type by using the search criteria to filter the results on the search page. Figure 11 shows general search criteria for assets to the left and a subset of the search criteria for the asset group "computers" to the right. The easiest way to display information for an asset is to use the links in the search result to navigate to the asset details page. This page will show all the available information about the selected asset, which makes it easy to find the previous and current location of an asset.

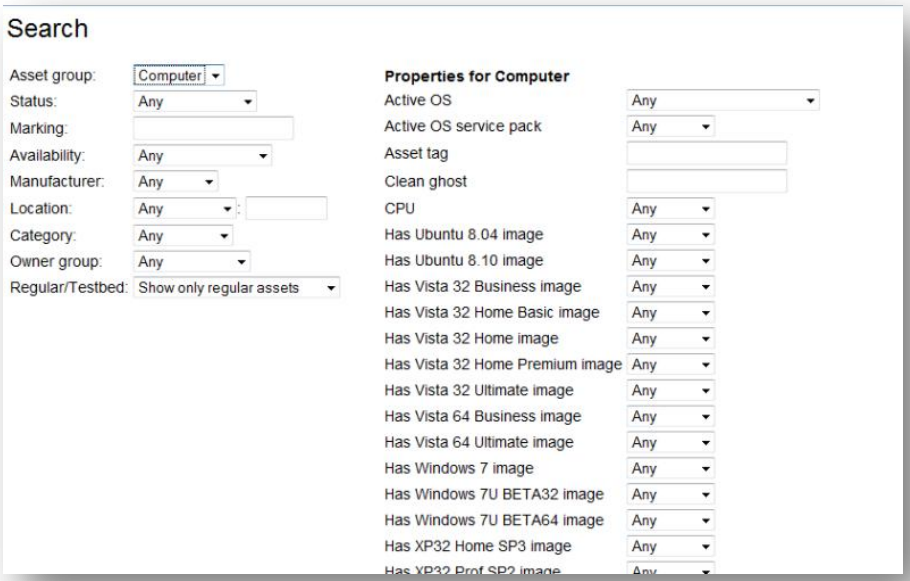


Figure 11: Search criteria for assets

The search results on both the search page and SQL search page support the possibility to sort the data in the result table. The sorting direction will alternate between descending and ascending for every click on the header. See the user guide in Appendix G for more information about how to sort the search results.

4.3.8 Scheduling Tests and Accessing Test Results

The manual functional testing will be replaced in steps by automated testing. Therefore, the users need a simple way to schedule automated tests on assets in the test bed. The web interface has a page that helps the user to schedule an automated test on a selected asset. When a testable asset has been located (easily be done by using the search criteria "currently available" and "part of test bed"), the search results will contain links to the page where the user can enqueue tests on test bed assets. See the user guide in Appendix G for more information.

To support the introduction of more types of automated testing there are several ways of using the test scheduling page. The user can choose to select predefined test suites stored in two different databases and she/he can choose to upload her/his own test file. Once the user has selected a database to pick tests from, the available tests will be listed. This makes it possible for the user to specify a test suite by adding one or many test cases in a very flexible way. The web interface also gives the user the possibility to select a package[29], a priority level and a number of iterations for the test suite to execute etc., see Figure 12. All this built-in flexibility is implemented to make the change to a more automated testing as easy as possible. Thus, the automated testing can be introduced stepwise.

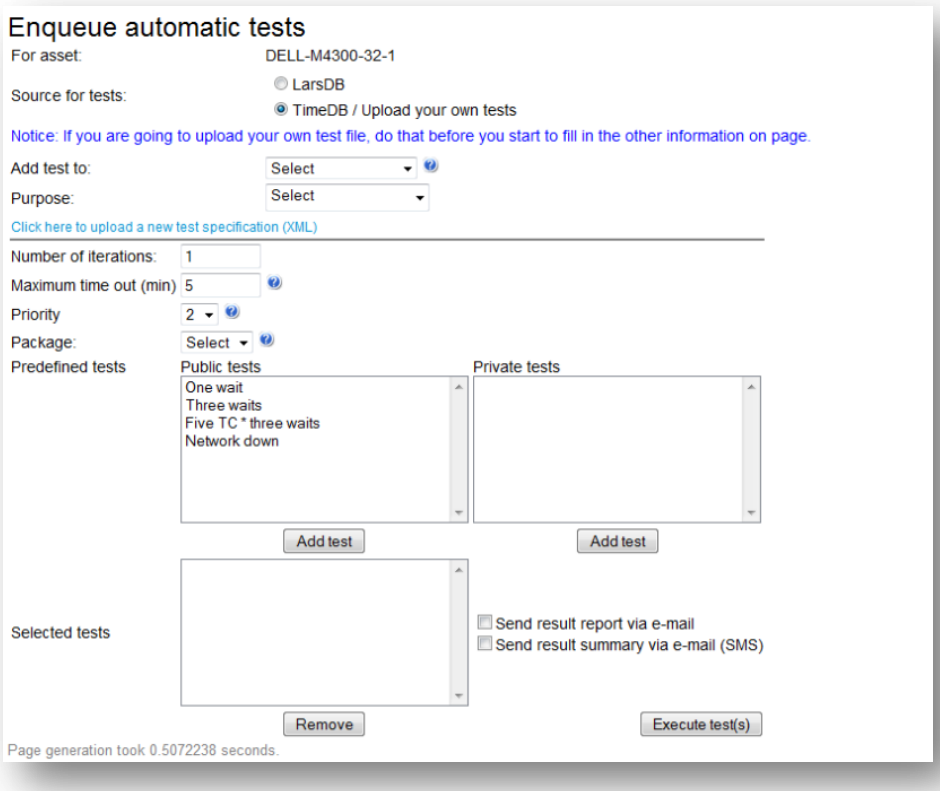


Figure 12: Screenshot of the scheduling of automated tests page

When the user scheduling a test places it in the test queue she/he also selects what type of feedback she/he wants. The user can choose between the following types of feedback: No feedback, email and/or SMS. There are also other ways of reading the result: The user can visit the Planning and Result Web Interface of Tier 1 (see section 3.3) and there is also a page in TIMEWI that reads from the Tier 1 database and shows real-time results for a specified test suite.

4.3.9 Booking Assets

The booking procedure is designed to fit into the normal work flow as much as possible. The user has two possibilities, either she/he searches for a suitable asset to book and places the booking before the

[29] The package contains all files that are needed for the test, such as drivers.

check out or, the other way around, finds an asset and tries to check it out and if the action succeeds a booking is placed. See the user guide in Appendix G for more information about how to use the booking and check-in and out pages.

The booking step is implemented in a clear and simple way. The user needs one or more assets and a purpose for the booking. Both calendars and text fields are then used for start and stop date and time for the booking. The booking page, where three assets are to be booked, is shown in Figure 13.

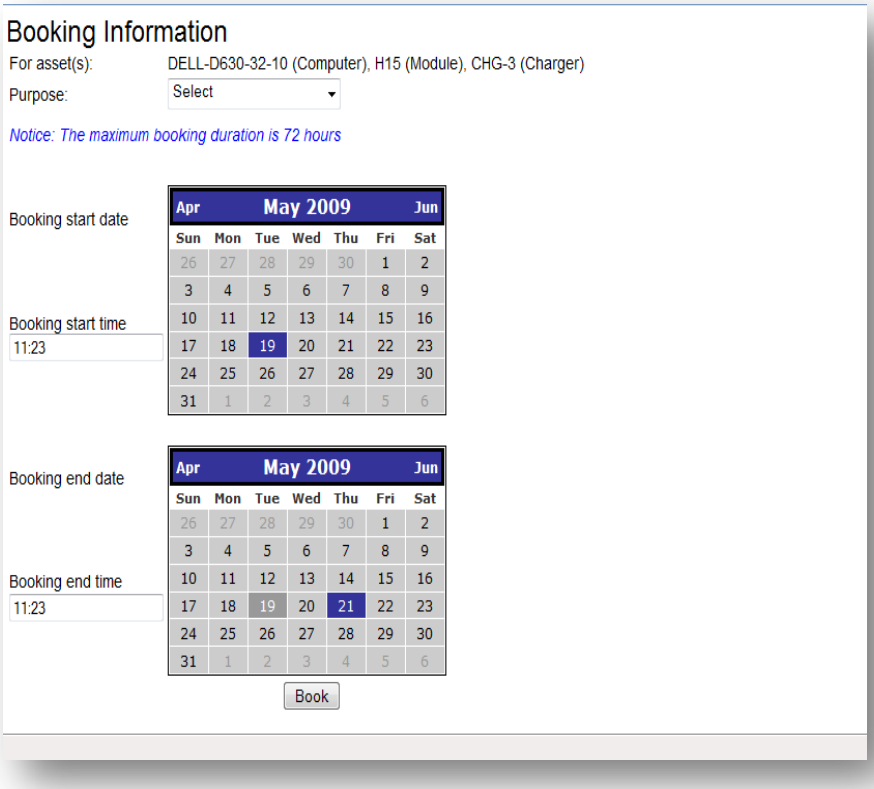


Figure 13: Screenshot of the booking page

If the user books an asset that is part of the test bed, it has to be the only asset in the booking. The reason behind this is that there is a difference between booking a regular asset (to physically take it and run tests on it) and booking a test bed asset (to get exclusive use of it). Making a booking of a test bed asset ensures that the user can add tests to the booking before it has started, but also when the booking is ongoing, and no other tests will interrupt the test execution.

4.3.10 Checking Out / Checking In Assets

The web interface contains separate pages for check-in and check-out. The assets can be checked in or out by using either the keyboard or the barcode reader to type the marking or scan the barcode. There is a physical check-in/out station with a bar code reader. The station is described in section 3.3.4.

All members of an asset family are supposed to be checked in or out at the same time, as an attempt to keep the family together all the time. To help the user to find out what other family members an asset family has, all the members will be listed in a table as shown in Figure 14. The check-out action will only succeed if all assets of a family is checked out (or marked as missing) at the same time.

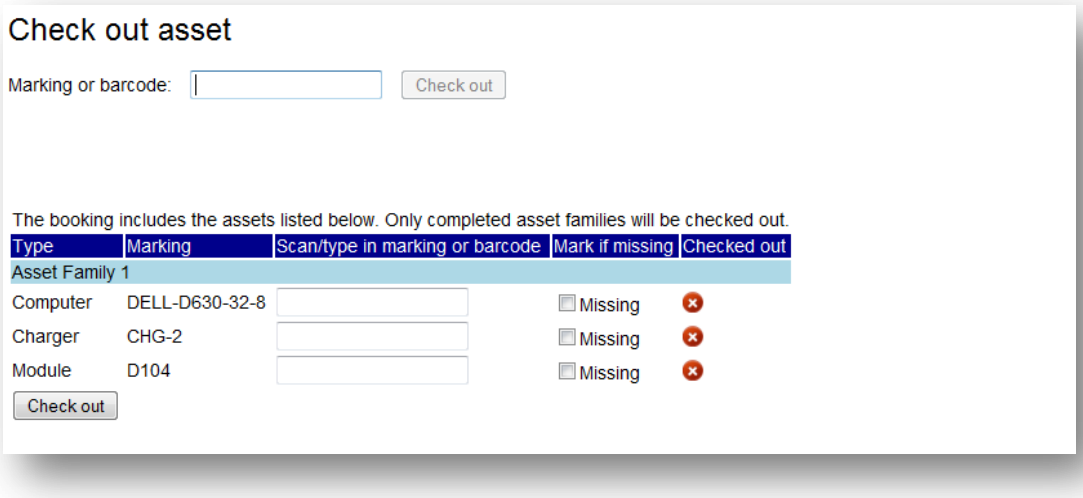


Figure 14: Checking out an asset family with three members

The table and status symbols are used for bookings too. As soon as there is more than one asset to check out all the assets of the booking will be listed in a table to speed up the check-out process, see Figure 15. The user can then choose which assets she/he wants to check-out now and which to leave for later. Again, only complete asset families will be checked out. When the check out action is triggered, the outcome is shown with status symbols, as can be seen in Figure 16. The example shows that family two and three have been checked out successfully (where the charger of family two has been reported as missing) but not family one, which can be checked out later.

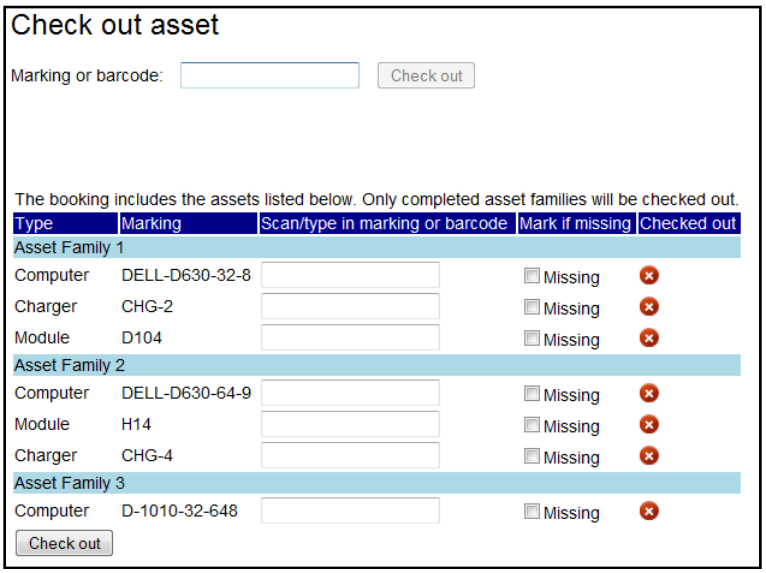


Figure 15: Three asset families to check out

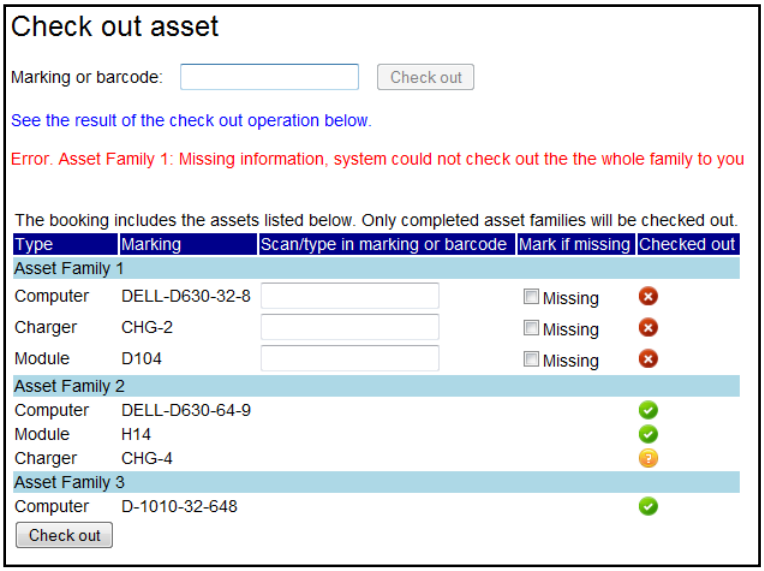


Figure 16: Family two and three has been checked out

The check-in page only supports check-ins of one asset at the time to handle the display of properties that must be updated on check in. A set of status images are shown on the page, detailing which assets in the family are checked in and which are not. An example of this, where only the module has been checked in, is shown in Figure 17.

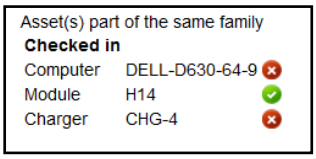


Figure 17: The check-in status of the current asset family

4.3.11 Check-in/out Station

There is a physical station, called the check-in/out station that supports the check-in and check-out procedures. It is located close to the storage area for most of the assets registered in the system. The station is configured on a stationary PC and uses two peripheral devices, a bar code reader and a label writer. Both the label writer and the bar code reader were configured to print and scan bar codes of type EAN-128[30]. The background to the final choices of the label writer and the bar code reader are discussed in the design phase, see section 3.3.4. The web browser used on the station was configured to use the TIME web interface as the start page.

[30] European Article Numbering using Code 128 barcode specification

4.4 The Implementation, Part Two: Management Functionality

This section covers the functionality in the web interface that assists in administrating, managing and monitoring the system.

4.4.1 Administration of Assets and Related Data

There are some users in the system that have administrator privileges. According to guideline 4 in section 4.2.3.1 the administration interface has been separated from the user interface as it is only used by a small group of users. These users have access to the administration interface, where an administrator can add/edit/delete records in the database. These actions are described briefly in this section, see the user guide in Appendix G for more detailed descriptions.

When adding, the administrator first selects what database table to edit. The page then shows editable fields for adding a new record of the selected kind. When the administrator has entered the information in the fields and pressed the submit button, the data is validated and added to the database.

Some types of data need special support functionality to leave the database in a working state. For instance, if an administrator adds a property to the asset type group computers, all the existing computers will have this property type from now on. For multi-valued properties, the default data value for an asset will be "Unknown" and for single-valued properties the value field for the asset will be left empty in the database.

Also, when the administrator has selected which table to edit, the existing database records of the selected table will be displayed in the table on the page. The table content is sorted by the unique record id in descending order by default to show the most recently added record first. The first column, called the action column, from the left in the list on the page contains links for editing and deleting records.

To avoid accidentally remove of records, all the *Delete* links, will show a confirmation pop-up and prompt the user to answer the question: "*Are you sure you want to delete the record?*" before any action is taken to delete information in the database.

If the *Edit* link is pressed, the information stored about the record will be displayed in the editable fields at the top of the page. The *Save* button will be replaced by two other buttons; *Update* and *Cancel*. In edit mode the action column in the table will be hidden to guide the user in the right direction of work. This is supported of the guideline 2 in section 4.2.3.1. If the user presses the Update button, the system will verify the information and try to update the data in the database. The system will notify the user about the outcome of the requested operation. If the user presses the Cancel button, the edit action will be canceled and the page will be reset.

The administrative interface is implemented to follow all the guidelines listed in section 4.2.3.1 to be as user friendly as possible. As it is likely that, when adding an asset of a certain type, the administrator may want to enter a similar asset into the database (for instance if she/he is adding two assets of the same type) a function that copies the information stored about an existing asset was implemented. This function simplifies and saves time for the user to e.g. add twenty new modules to the system by using this function and only change the unique information i.e. marking and barcode for each new asset.

4.4.2 Maintaining the System

For a system with plenty of users and assets and ongoing actions a maintaining system is needed. A watchdog has been implemented in the web interface. Its task is to keep track of what is going on in the system and take required actions.

4.4.2.1 The Watchdog

The watchdog is responsible for maintaining the system, reducing the need for having someone who manually monitors the system to, for instance, find users who are not returning their booked assets on time. It has four main responsibilities:

- Remind users that has late bookings by sending emails
- Notify owners to the late assets by emails
- Find bookings that has started but not been checked out and cancel such bookings if the start time is overdue by 24 hours
- Find assets that needs service and inform the asset owner

The watchdog shows its activities on a certain page in the web interface. Each responsibility has its own section of performed tasks, where every row describes an activity, the result and an action. The symbol to the left shows the status of the activity. A wheel means ongoing, a green means ok/pass, a red means error/fail and a yellow question mark means unknown. A screenshot of the watchdog in action is shown in Figure 18.

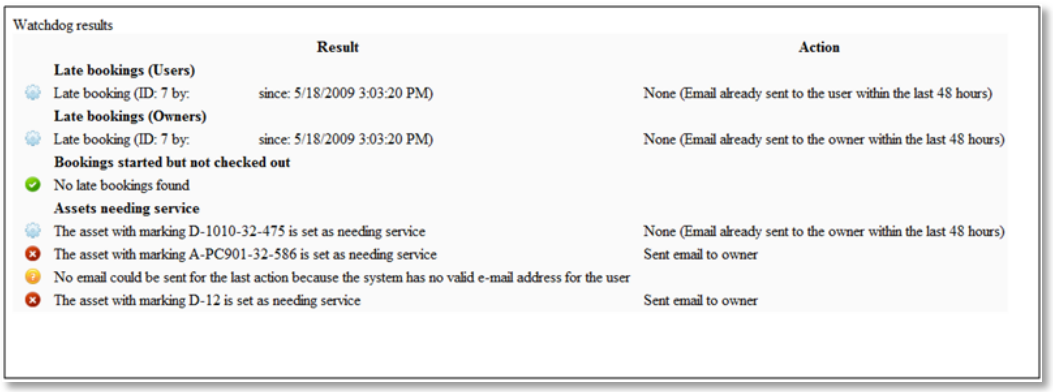


Figure 18: The result of a watchdog run

The watchdog keeps track of the actions it has taken to avoid annoying people by sending multiple emails at each update. Before each action it plans to take, the watchdog makes sure the action has not already been reported by the watchdog within the last 48 hours. If not, it is allowed to send or re-send emails if the problem still exists. Since the watchdog is primarily meant to be run by a scheduled script, it does not have the same layout and features as the rest of the web interface.

4.4.2.2 Test Bed Status Monitor

The web interface has a page for monitoring the status of the assets in the test bed. Each asset in the test bed (called an SUT) is represented by a table that shows information about the current status and the time of the latest update as well as the IP address. There is also information showing if the asset is active or not. An active asset has a visible link for scheduling tests. A bookable asset has a visible link to the booking page. A screenshot of the test bed information page is shown in Figure 19.

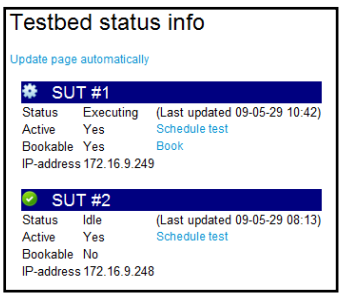


Figure 19: Status page for assets in test bed

4.4.2.3 The Configuration File

There is a configuration file that contains values that may need to be changed during runtime of the system. The configuration can only be changed by someone with remote access to the web server. The file contains configuration data i.e. how long to wait before the system shall send an email to the user that has checked out an asset but not returned it when the booking expires and the default booking duration etc. are example of values that are defined in the configuration file. The content of the configuration file is described in details in the user guide found in Appendix G.

4.5 System Quality Attributes

This section analyzes the implementation by using some known system quality attributes.

4.5.1 Performance

The number of users of the system for the system is expected to be significantly smaller than 100. This combined with the fact that very few resource-intensive operations are done in the web interface means performance is not considered a very important part of the web interface. Still, some work has been done to lower the amount of resources needed to run the system. This section will mention part of the work done divided into two categories: Data caching and other manual optimizations.

4.5.1.1 Data Caching

Data in the database that is not likely to change often is cached in memory, using the ASP.NET caching mechanism [27]. Saving a copy of this data comes with a certain risk of the data being out of date. The system implements three measures to counter this risk. First, all cached data is set to expire after two hours, regardless of use. Having this rule appears to be a good compromise between decreasing the database access and keeping the data reasonably up to date. The second measure is to ensure that the cache-refresh routing is called as soon as any administrative task changing said data is changed. The administration part of the web interface is responsible for calling said routine after it has saved data to the database. The third measure is to provide a link to a web page whose purpose is to call the cache-refresh routine. When an administrator changes the data using a connection to the database that is not controlled by the administration part of the web interface, the administrator can visit the cache-update page after committing the change to the database, thus ensuring that the cached data is up to date.

The web interface also uses data from a configuration file, described in section 4.4.2.3 . To minimize the number of I/O operations needed to access this data, it is cached using the Configuration Data Caching pattern described by Welicki [28].

4.5.1.2 Other Manual Optimizations

A list of further suggestions for manual performance optimizations is provided by Howard [29]. Two directly applicable suggestions were put as guidelines for implementation, namely "Return multiple result-sets**"** and "Use the view state". The effects of the guidelines on the implementation are discussed below.

Return multiple result-sets

If more than one database query was needed to extract the data needed for an operation, effort was made to send all of the queries as a batch statement, resulting in a data set containing the results of the queries separately. Even some cases where the second query depended on the first could be optimized to be sent as a batch, using conditional statements[31]. The benefit of this optimization is summarized by Howard as "*By returning multiple resultsets[sic] in a single database request, you can cut the total time spent communicating with the database*".

[31] For instance the IF..ELSE and IF EXISTS Transact SQL statements

Use the view state

The view state allows for data to be persisted to the client as part of the POST[32] data that is sent by the client when requesting an update of the page. Each piece of data that was generated during page-generation was evaluated based on the following three questions: "Will this data be needed again?", "How expensive was it to generate this data?" and "How much space will it take to serialize the data?". A piece of data that will be needed again and is expensive to generate but small in size is a perfect candidate for being stored in the view state. The next request to the page can simply check for pre-existing values of the piece of data in the view state to see if a re-execution is needed.

4.5.2 Security

Noureddine et al. [30] offers a set of "industry best practices" regarding security. The system implementations of two of these are described below.

4.5.2.1 Restricted Database Access

To prevent users from intentionally or accidentally executing mal-formed queries to the database, three database accounts were set up. For the SQL query search, a database account with select-only permission is used. For other types of regular use, an account with insert-select-update permissions on certain tables is used, and for administration, the queries are executed with an account with insert-select-update-delete permissions on all tables.

4.5.2.2 User Access Rights

Since the system will only be used in an internal domain-based network, the system uses domain-user information to give users access to the web interface. A database table is used to match the user name to a set of permissions. A user for which the user name is not found in the table is given guest access rights, meaning only searching for data is allowed, and only using the select-only database account.

[32] The browser sends POST data when making a request to a web server. It is part of the HTTP standard.

5 TEA-Coordinator – Test Execution Coordination

This chapter describes the design and implementation of the TEA-Coordinator, which provides the functionality of the TEA sub-system of Tier 2. Section 5.1 describes the design and architecture, section 5.2 details the implementation of the test automation, and section 5.3 analyzes the implementation of the application.

5.1 Application Architecture

While there appear to be abundant information on how to design a good user interface (see section 4.2.3) and which architectural patterns to use when designing a web application (see section 4.2.2), there seems to be less specific information for test coordinators (perhaps not surprisingly). Hence, a set of common practices, such as separation of concerns and other guidelines, described in Appendix E, were used as guidelines for the design.

Following the practice of separation of concerns, the application was divided into a set of modules. A list of these modules, with a brief explanation, can be found below:

- Core – The core module handles start-up and shut-down of the application and provides a set of functionality used by other modules, such as logging and state tracking
- TestController – The test controller module provides functionality to schedule and execute tests, described in detail in section5.2.1
- TEATIMEDBHandler – The TEATIMEDB handler module handles requests for data from the TEATIMEDB made by other modules
- ResultDBHandler – The result database handler module is responsible for passing the results of tests to the planning and result database in Tier 1
- CommunicationHandler – The communication handler module implements the TEACUP protocol and a set of functions to enable the TEA-Coordinator to interact with the TEXAS Manager in Tier 3

5.2 Efficient Test Automation

The guidelines for successful automation of testing, mentioned in the theory section 2.1, were followed during the implementation of the TEA-Coordinator. The TEA-Coordinator was implemented to manage the test queue and execute automated tests in an efficient way by using priority and separated test queues for each asset to be able to run tests in parallel. Figure 20 shows an overview of how the modules involved in the test scheduling and execution interact with each other. The work flow starts witch fetching tests from the TEATIMEDB. The tests added in the queues of each asset. Each asset has a set of three queues, each with a specified priority. The priority levels are 1 (highest), 2 and 3 (lowest). The priority level for each test is stored as an attribute in the TEATIMEDB. By having set of priority queues for each asset it is possible to perform efficient testing.

The test controller that manages the queue starts new threads running test executors for each test item to be run. The test executor then controls the execution run on the SUT by communicating with the test controller and the TEXAS Manager. The result of each subtest, test case and test suite[33] is stored in the ResultDB via the ResultDBHandler. The test controller thread, started by the thread that runs the GUI for the coordinator, will run in an infinite loop until the coordinator is told to stop. The work flow is described in details in the following sections.

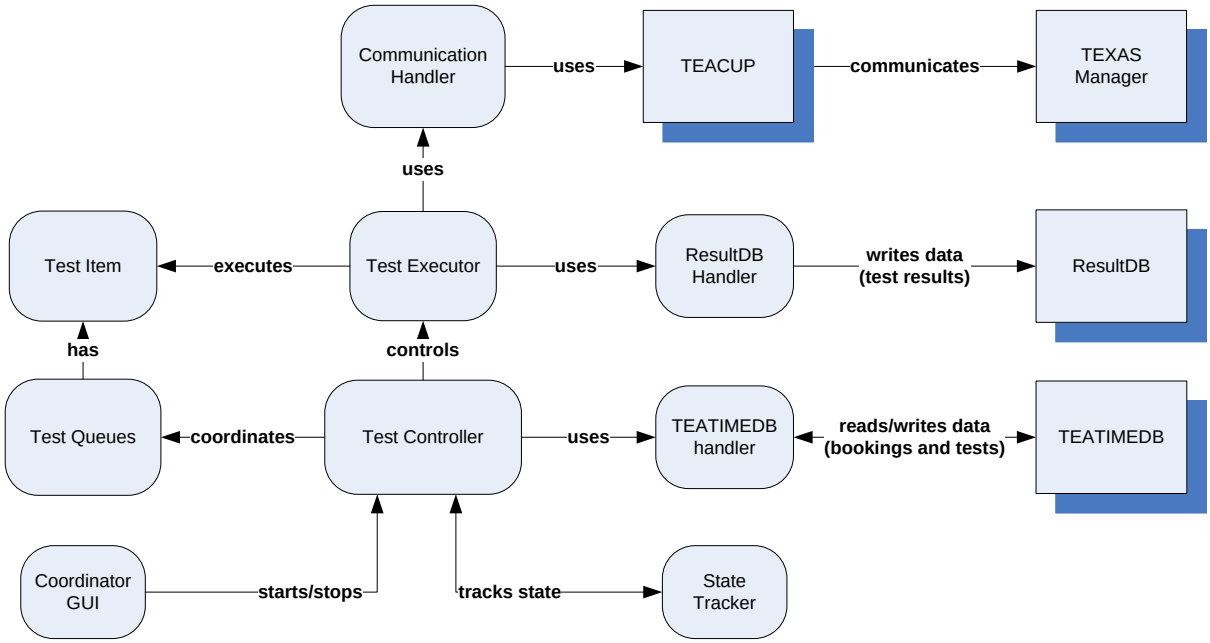


Figure 20: The interaction diagram for the test automation

[33] A test suite contains one or more test cases which in turn contains one or more sub-tests

5.2.1 The Test Controller

This section describes the test controller work flow which is visualized in Figure 21.

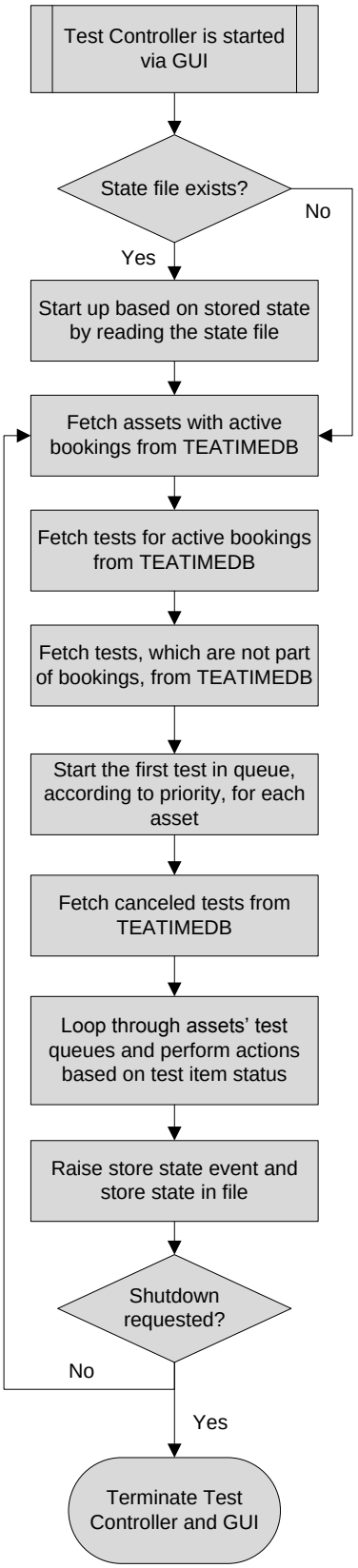


Figure 21: Test controller work flow

5.2.1.1 Initialization

The test controller is the master module of the coordinator. Initially the test controller creates an instance of the test queue and an instance of the result database handler as well as the state tracker. It also create necessary variables i.e. lists and dictionaries for later use. A list of TCP[34] ports is initiated to indicate what ports to use during the test execution for communication with TEXAS Manager. A main timer is created and set to execute a function when the specified time has elapsed. If a state file exists, the test controller starts up based on this state, see section 5.2.1.5. Otherwise it starts up from scratch.

5.2.1.2 Test Controller Main Loop

When the main timer elapses, the controller thread calls a function that takes care of polling the TEATIMEDB database for information.

Fetch assets with active bookings

The first task for the controller is to fetch the assets with active bookings. If there is an ongoing booking, the current test queues of that asset will be cleared and filled with the tests that belong to the active booking.

Fetch tests for active bookings

The second task is to ensure that, for each booked asset, all tests part of the active booking for that asset are fetched. If any new tests are found, the tests are placed in the priority queues (see section 5.2.2) of the asset and their status is set to *Pending*. As soon as a test item is changed the timestamp of the last contact is stored to keep track of which test items are alive. The controller remembers the highest test id number of the fetch tests to avoid fetching tests it has already put in the queue.

Fetches regular tests

The third task for the controller is to get newly added test not tied to any booking from the database. If there are any tests, they will be placed in the priority queues using the same method as for the tests part of a booking. If the asset is free, and there is a test in queue for that asset, the first test in the queue with highest priority, which is not canceled, is moved from the queue, to a list holding the currently running tests. The test is then started by setting up a new test executor (see 5.2.1.3).

Fetch canceled tests

The controller's fourth task is to ask the database for canceled tests. If a user has canceled a test, the controller will notify the test executor running that test by changing the status of the test item in queue to *Cancel*.

[34] Transmission Control Protocol

Perform actions based on test item status

Depending on the previous status of the test item a switch-case statement is used to select a suitable action.

- *SUT Busy*: If the SUT was busy last time the test on the asset was going to be executed, the controller checks if enough time has passed to try to connect to the asset again.
- *SUT unreachable*: If the SUT was unreachable on the last attempt, the controller checks if it is time to try to connect the SUT again. If that is the case, the controller starts up a new test executor to start executing the test. The number representing the amount of attempts made is increased, and if the max limit is reached, the controller sets the asset's status to inactive in the TEATIMEDB database and send emails to the people responsible for the test bed, informing them about the issue.
- *Done*: If the status is *Done*, the controller updates the test status in database and removes the test item from queue and updates the status in the database, setting it to *Complete*. It then releases the asset from the list of currently busy assets and also releases the TCP-ports used for the finished test item.
- *Pending, Starting Up, Cancel* or *Running:* Nothing happens for the test items whose status is in the range *Pending, Starting Up, Cancel or Running.*

Next step: Check if all queues for the asset are empty, if that is the case, check if the asset belongs to a current booking, if that is also true, check if the coordinator is allowed to add more tests (means not stopped by user) and if the booking has expired. If all the steps above are true, set the status of the booking to *Completed* and remove the asset from the list of ongoing bookings. Reset the asset to normal automated test execution mode and fetch the tests from the database. An important reset is done here by resetting the stored highest fetched test id and use id equal to 0 to fetch all the tests that are neither set to *Done* nor *Canceled* for each asset. If there are any tests in the database, add them to the queues by using the method described in section 4.3.1.4. Otherwise, remove the asset from the main queue. Do the same if there is no ongoing booking for the asset. End this step by flushing the buffer of the result database handler, to ensure that anyone looking at real-time results does not need to wait an overly long time to see the results of tests that are currently buffered.

Raise the store-state event

The test controller finally fetches the information necessary to store a state. The information stored in the internal state is the converted test queue object and the state of the current result database handler. The state is serialized by the state tracker and written it to an XML file.

Check if a shutdown of the TEA-Coordinator was requested

When the loop is done the timer is reset and the controller thread waits for the timer to elapse. If the coordinator is stopped, another function will be called which stops the test controller main timer, after making sure that no new tests are added to the queue and that the ongoing test executors will be finished their tasks before the coordinator are shut down.

5.2.1.3 Setting Up a Test Executor

When setting up a test executor, the controller first adds the asset to the list of busy assets. Then the controller searches for the two first available TCP ports in the list of allowed ports defined by the configuration file. Even if a port is set to be free in the port array, the controller has to make sure it is not used by the system. If the ports are free, they are set to be used from now on.
The controller then sets the test status to *Starting up* and stores the timestamp for test started in the test item object after which it starts a new thread and instantiates a test executor.

5.2.1.4 Add Tests to Queue

When a list of test items is to be added to the queue the test controller first checks if there are any ongoing bookings. If test will be run on asset that is not part of an ongoing booking, the test is enqueued for the specified asset. This is done by checking if the asset has a set of priority queues, if that is the case place test in right queue depending on priority level. Otherwise create new set of priority queues and add the test in the right queue. The test added to a queue gets its status set to *Pending*. The process is completed by storing the highest test id of the test items placed in any of the queues.

5.2.1.5 Start from Stored State

If a state file exists, the test controller will run a function to validate and restore information and try to start up based on this state. The state file stores all assets that have test items in their queues, the result database handler and the current running tests. If the state file is missing, the controller starts up from scratch.

The validation is done in three steps. First, both the test database handler and the result database handler are given the last state that was stored for them in the state file. Second, each test in the test queue is matched with the two database handlers. If the state of the test does not match the state in either of the databases, for instance the test might be set as complete in the test database but as running in the state, it is discarded and a warning is written to the log. Third, for each test that has been validated against the database and has a status that indicates it is still running, an attempt is made to contact the TEXAS Manager that was responsible for executing this test. If the TEXAS Manager is not found, the test is discarded and a warning is written to the log. If the test passes this step, validation is complete and the test executor is given the test and set to start in resume mode (the protocol implementation for resuming a test can be seen in Appendix F).

5.2.2 Test Queue and Test Item

The automated test execution is supposed to be as efficient as possible. This means that the coordinator must be able to schedule and execute tests on several assets in parallel. The first implementation of the main queue was pretty simple and did not fulfill all the requirements i.e. give bookings higher priority than the regular automated test queue and different priority levels on tests in the queue. The second version of the main queue implementation became a matrix of queues with three queues ordered by priority for each asset. The controller sets the asset as free when all its test queues are empty.

A test that is placed in a test queue is of the data type called *TestItem*. A test item holds lots of information about the test to be executed. For instance, the information is about what asset to run on, what test files to use, number of iterations, priority etc.

5.2.3 Test Executor

A test executor thread is started for each test. The test executor gets a reference to the test item to execute. A test state is created and an instance of the logger as well as the settings and the result database handler. A passive communication handler is initiated for the communication with the coordinator and the TEXAS system. The test executor work flow is shown in Figure 22.

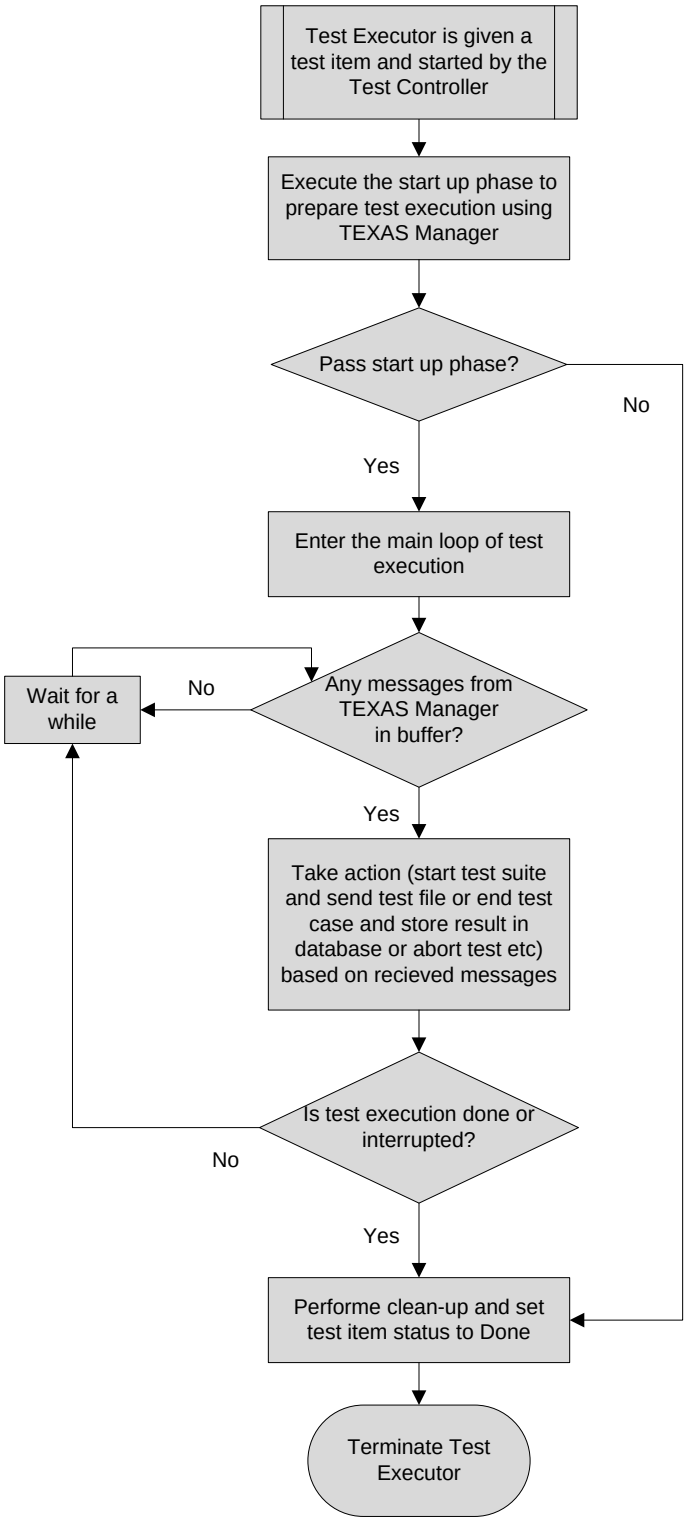


Figure 22: Test executor work flow

5.2.3.1 The Start-up Phase

There is a method that is used for running the test executor in normal mode. The method catches errors caused by the database, network and result database state. It checks if the test is to be started up from the beginning (which includes an interrupted start up phase) or from a stored state, described in section 5.2.1.5. A method is then called to try to connect to the communication handler. If the test to be executed is new, a start up phase is started. This phase consists of several preparation steps listed below.

- Step 1: Send status query message to TEXAS Manager
- Step 2: Wait for TEXAS Manager to be ready
- Step 3: Send message to connect to TEXAS Manager
- Step 4: Establish daemon communication (expect messages saying either connected ok or daemon busy)
    - If daemon is busy, stop test executor by running the special clean-up method
- Step 5: Send path to package of drivers and branding sheet etc
- Step 6: Start session for the result database handler
- Step 7: Save all paths to test files to be send in a list, send the first test file path to TEXAS Manager and remove it from the list
- Step 8: Wait for confirmation that the test suite has been set up
- Step 9: Send start test execution message to TEXAS Manager
- Step 10: Update status for test to *Running*, means it has passed the starting up phase

If a step fails the test executor thread exits by running the clean-up method.

5.2.3.2 The Main Flow Loop

If the start up phase is passed, the run-main-flow method is called. The method contains the main flow, which loops until the test execution is done or aborted etc. (includes clean up). The main flow loop first checks if the test is canceled by the user. It then checks if the SUT is currently in a planned outage and, if so, if the specified max time has passed. If that is the case, the clean-up method will be called and the user will be notified about the failure.

The next step is the check if there are any messages in the buffer of the communication handler. If this is the case, first message is fetched and read. If it is one of the expected messages, take care of it. When there are no more messages in the buffer, the test state is updated in the test item object after which the thread sleeps for a while before it starts from the beginning again.

5.2.3.3 The Start and Finish of Test Suite, Test Case and Subtest

As soon as a message is received that a new test suite is started, the test executor increases the iteration number and starts a session for the result database handler for the test suite. When a test case is started, a session for the result database handler for test case is started. There is no session for the sub test level.

When a test case is completed, the result is reported to the database. In the case where a sub test is completed, the result is reported to the result database, and if the SUT had planned outage, the planned outage flag is now set to false.

When a test suite is completed the result is reported to the result database and the result log sent by the SUT is moved to a network share. The test executor then waits for the TEXAS Manager to either say it is ready or continue by starting the next test suite. If the manager is ready, the test executor checks if there are more test files in current iteration to be run, if that is the case, the file path to the next test file is sent. If there are no more test files, check if there is any more iterations to run. If that is the case, decrease number of iterations left and refill list of path to test files to be sent and re-send the first file path followed by a new start message to the TEXAS Manager. Otherwise, the test is done and a clean-up is performed.

5.2.3.4 Other Messages Received

A message is received from the TEXAS Manager if the SUT has planned outage, which means that it will be unreached for a while. To keep track of when the SUT has been disconnected too long, a flag is set and the time for max planned outage is recorded in the test item object.

If a test is aborted by the user who is responsible for the test, the manager sends an acknowledge message and a note about the abortion is reported to the database followed by the clean-up function.

A message is received from the manager if the test execution has failed. This can happen if the test execution was aborted because the SUT has crashed or similar. A note about the failure is taken to the result database and the test executor calls the clean-up method.

Almost all actions mentioned above are logged. Wherever something important is done that needs to be reflected in the state, the state of the test controller is forced to be updated, this is described in section 5.3.1.2.

5.2.3.5 The Clean-up Methods

There is a clean-up method that cleans up when a test is finished somehow i.e. when it is done or when it fails unexpectedly during execution. The clean-up method sends a stop signal to the TEXAS Manager, to show that the communication is over and that no more messages will be sent. If there is any ongoing result database handler session it will be ended. The clean-up function also sends email and/or SMS to the user that ordered the test as a result of what happened. Finally, it changes the status for test item to *Done*, which means that it will be removed from test queue by the test controller.

There is a separated clean-up method that is called when a test is postponed. This special clean-up does the same as the normal clean-up except of that it resets the process id for the TEXAS Manager and it stores the current test state to the test item instead of setting the status to *Done*. By doing this, a new test executor thread as well as a new manager is started when the controller tries to start the test again.

5.2.3.6 The Pass-step Method

There is a method that checks if the test executor is allowed to pass a certain step or not. A list of status messages is sent as input parameter and the method returns a message. The method uses a while-loop with a thread sleep call to wait for a number of messages to be received from the communication handler or if no messages are received, to increase a counter until a certain number is reached. The method parses incoming messages to see if any of them matches any of the expected messages. The loop breaks if it is an expected message, and returns it. Otherwise it returns null which typically leads to that the clean-up method is called and the test executor thread exits.

5.3 System Quality Attributes

This section will analyze the implementation using a few system quality attributes.

5.3.1 Availability and Reliability

As the coordinator interacts with several other systems out of its control, it must also be able to handle errors in these systems.

5.3.1.1 Keeping Errors from Propagating

The coordinator has internal safeguards to keep errors in one of the external systems from crashing the whole application. These safeguards are mostly implemented using a series of try-catch blocks, ranging from handling "common" scenarios[35] to handling unknown errors. When an external system performs an invalid action, an exception will be thrown either by the .NET framework or by one of the internal validation methods. This exception will then be handled by one of the safeguards. The test execution sub-system implements *clean-up* functionality 5.2.3.5 with the goal of making sure that the coordinator is left in a valid state if a test needs to be aborted due to an error. This clean-up also attempts to inform the administrators of the system about the cause of the error (via e-mail and database logs)

5.3.1.2 Keeping a Persistent State

To be able to recover from a hardware or software crash, the coordinator keeps a checkpoint copy[36] of its state to disk. The module responsible for storing the state to disk is run in a separate thread, not to interfere with (nor be interfered by) the rest of the software. The persistent copy of the state is updated by two separate mechanisms. First, a timer forces a state update after a configurable period of time (pessimistic state flush). Second, any thread that changes its internal state in a way that must be persisted for recovery to work[37] raises an event that forces a state update (optimistic log flush). This way of storing a state to disc is covered by Wang et al.[31].

When recovering from a crash, the coordinator begins by checking the disk for a stored state. If one is found, it validates the content against the environment, checking for open result sessions, test queue state and running processes. Based on the environment it changes parameters on the test to be executed before handing them over to the test controller.

5.3.2 Correctness

The coordinator has not been formally verified, but three methods have been used to promote correctness: Using recommended practices, code review and unit testing.

5.3.2.1 Using Recommended Practices

A lot of effort was put into following existing recommended practices for design and implementation, which promotes correctness. One collection of guidelines is published by Microsoft as part of its MSDN[38] library [32]. A similar collection is published by dotnetspider [33]. As the number of recommended practices in these collections exceed 100, it is not practical to list them all in this report. A small subset of the practices used and examples on how they were used can be found in A Subset of the Coding Practices Used, with Examples.

[35] E.g. a remote test assets fails during a test

[36] A checkpoint copy is a copy made at a well defined state

[37] E.g. a test has passed the set-up phase and entered the running phase

[38] Microsoft Developer Network

5.3.2.2 Code Review

The code produced during the implementation was continuously reviewed by the developers according to the proverb "*Given enough eyeballs, all bugs are shallow*", sometimes referred to as *Linus' Law*. Any error or question that came up as a result of the review was noted in the Scrum document, mentioned in section 1.5, to ensure it would be taken care of.

5.3.2.3 Unit Testing

Unit testing was used heavily in the initial development phase to ensure that the basic functionality was correct. The unit testing framework was also used to run a set of tests, which can be seen as a middle-ground between unit tests and functional tests, with the use of some dummy implementations of other systems the coordinator would interact with, such as the TEXAS Manager and an SMTP[39] server. The unit test framework was set up so that it would first start the dummy implementation and then run a series of tests on the code of the coordinator that interacted with the dummy. The final code coverage reached with the unit tests was approximately 30%.

5.3.3 Performance

Due to lack of hardware access, it was not possible to verify the upper number of simultaneous tests the coordinator could execute. Still, profiling was used and some heavy-load test cases were designed and executed to get a feel of how the coordinator behaved under pressure.

5.3.3.1 Profiling

To find areas in the code for improvement, the Visual Studio Profiler was used. The profiler keeps track of how much time is spent inside the functions in the code and generates a report of this information from which information can be extracted on which functions the total execution spends the most time in.

5.3.3.2 Heavy-Load Test Cases

Because of the previously mentioned hardware access limit, the coordinator has not been tested with tests being executed simultaneously on more than two SUTs. In an attempt to make the best of the situation, two test suites were set up to show that the coordinator could handle certain loads. These test cases and the result of their execution are described below:

Test suite 1: Many sub-tests

A test suite containing 10 sub tests was created. This suite was then set to repeat 15 times in Tier 3, leading to a total of 150 sub-tests being executed. This suite was then further repeated 10.000 times in Tier 2 meaning that as soon as the test suite was reported as done in Tier 3, the same suite was started again, using the same SUT and the same TEXAS Manager. This test suite was then set up to run on both the connected SUTs, meaning that a total of 300.000 sub tests were to be executed. The test execution took 61 hours, and the result showed that while one of the SUTs had stopped the test execution after ca 50.000 sub-tests due to an error in Tier 3, the other SUT finished the whole test suite. Analyzing the result, the following three conclusions were drawn: First, the coordinator can handle the execution of test suites with 150.000 sub-tests in the same session. Second, the memory imprint of the coordinator was the same both before and after the test, meaning that it does not "leak" memory and it does not accumulate objects in memory over time. Third, using logging for session of this magnitude with the log level set to DEBUG (the logging class is described in A Subset of the Coding Practices Used, with Examples) resulted in a log file with a size of over 250 megabytes. A file of that size is impossible to open in many text editors. A decision was made to limit the log file size to 10 megabytes before starting a new log file.

[39] Simple Mail Transfer Protocol

Test suite 2: Buffering test results

The test suite created for the first heavy-load test was modified to repeat itself in Tier 3 100 times, meaning that 1.000 subtest would be executed. It was also set to make the SUT disable the network connection to the coordinator while the test was executing. When the test executor was finished with the test set-up and told the SUT to start the test, the SUT responded with a message saying that the network was to be taken down, which puts the executor in a waiting mode. After a few minutes, the 1.000 sub tests had been executed and the SUT enabled the network again. As soon as the network was up, it sent the 1.000 sub-test results back, in a very rapid succession, to the coordinator via the TEXAS Manager. The conclusion that was drawn from this test was that all the parties involved in the test could handle sending and receiving results in a very high frequency.

6 TEATIMEDB – Asset Tracking and Test Scheduling Database

The data needed for inventory and booking management, such as information on asset configuration and history, for the system is stored in one source in a relational database. There is also information about an asset's location, owner, services, bookings, test history as well as user history. The database is used by the TIMEWI and the TEA-Coordinator. The database is created in MS SQL Server 2008.

6.1 Design of the Relational Database

As mentioned in the design section 3.3.3, a relational database was designed in four steps. First an E/R diagram[40] with entities, relations and attributes was drawn. The second step was to verify that the E/R diagram correctly models the domain and its constraints. The third step was to define primary keys and also underline them in the diagram. Finally, the last step was to translate the E/R diagram into a set of relations with keys and references.

A SQL script of CREATE statements for all tables was written to be able to create the database easily by running the file. A set of database views[41] was created to allow for less complex queries to support the execution of the search function in the web interface.

6.2 Implementation of the Database

The database was created in Microsoft SQL Server 2008 by running the SQL script mentioned earlier. The database is called TEATIMEDB and it is deployed on a database server at MBM, Ericsson.

6.3 Content in the Database

The exact design and the content of the database are confidential. Thus, only an overview of the database design is given in this section.

The database contains 25 tables. There are a set of main tables that holds information about assets, asset type groups, users, bookings and the queue for automated test execution.

All tables, except for the connection tables[42], have attributes to store information about which user made the last change, and when, for each database record. This makes it possible to keep track of edits and the person responsible for the data in the database.

6.3.1 Information about Assets

In the database there is a set of tables that holds information on assets. There is a three-level hierarchy used for identifying the asset. Topmost is the asset group which divides the asset based on the general group they belong, examples could be "computers" and "cables". Below the group is the type, which generally represents the model of the asset, examples here could be "Eee PC 701" or "2m CAT5e network cable". The type is further tied to a manufacturer. The final level in the hierarchy is the asset itself.

A different set of tables holds information about asset properties. Examples of properties could be "Operating systems" for computers or "Shielded" for cables. When a new asset is entered into the system, a set of property types tied to the asset group of the asset is used as a template for the properties that are generated for the asset. Each property has a data type and a value, the value being either a text field or a reference to a set of predefined data values tied to the data type. The multiple-value data types can be thought of as drop-down lists.

[40] Entity-relationship diagram

[41] A view gives access to a virtual table representation of the result of a query involving one or more tables

[42] A connection table connects two tables by supporting a many-to-many realationship

As there is no fixed set of properties for assets, new properties can be added and old ones changed or removed as necessary. If, for instance, sometime in the future computers start to become waterproof, a property with the data type name “Is waterproof” and the values “Unknown”, “Yes” and “No” can easily be added to the database and tied to the “computers” asset group.

Information about the users in the system is stored in a table. The users are divided into groups that are related to the different asset categories mentioned above. The web interface supports SQL query searches by users. These queries can be named and stored in the database. A certain table holds information about the saved queries.

6.3.2 Information Stored for Automated Test Execution

The web interface allows users to schedule automated tests to be run on assets. Information about the order of test execution, what test to be run on each asset and information about what test files to run are stored in a set of related tables in the database. The TEA-Coordinator reads and writes data from these tables when it manages the automated test execution.

7 Support Sub-Systems

A number of supporting applications were developed during the implementation. While they are not part of the original scope of the project, they provide some features that may be interesting for future development.

7.1 Package Monitor

The package monitor was originally developed as a proof-of-concept, showing that it would be possible for an external application to specify and schedule tests entirely without human interaction. The package monitor polls a file system directory for changes. When a new folder appears, it enters the folder and waits for a specific file saying that the upload of the folder is completed[43]. When it finds said file, it reads the package specification (a human-readable XML file) and schedules one or more tests based on the content of the contents of the specification. Any application capable of creating XML files and transferring data to a network share or FTP server could thus schedule tests.

7.2 Spreadsheet Data Import

Before the deployment of the system, two spreadsheets were used to hold test asset information (mentioned in section 3.1.1). The spreadsheets were constructed to be easy to read and use, but were unfortunately not well suited for direct data import. An application was developed that parsed the content of said spreadsheets into the database. The application does some validation before storing the data in the database and uses a database transaction to be able to roll back the input on error.

[43] This is done to avert the risk of scheduling a test without having the complete specification uploaded

8 Discussion

In this chapter we, the project members and report authors, will evaluate our work and describe some delimitations and problems that were encountered.

8.1 Research Questions

First, the three research questions that were stated in section 1.3 are all of the type of question that can only be given a final answer after the system has been in use for a while. At the publication of this report, the system has only been in active use for a little over a week. Initial reports from the users have been very positive but we guess this might at least in part because the users are just glad the old way of doing things has been discarded.

Most of the comments, also the most positive ones, have been given from the people tasked with administrating the functional testing and keeping track of the assets used for this purpose. We feel this indicates that the regular user does not notice that much of a difference, while the system helps the administrators a great deal, something we feel is very positive.

8.2 Problems Encountered

The factor that has been the greatest hurdle of this project is time. Both in the sense that we may have set to big a scope to begin with, but also that external resources promised were removed in favor of upholding a reasonable schedule for the regular functional testing, which is fully understandable but nonetheless regrettable for our project. The project also became a victim to a syndrome called “feature creep” (meaning that, despite our best efforts to prevent it, some requirements were added late in the project), something that, while unfortunate, is not uncommon for system design and implementation projects of this type.

One of the work packages that were to be put on external resources was a two week formal verification of the system. As the verification was not performed, we can only use our own limited best-effort verification to comment on how well the system fulfills the requirements. As stated in section 2.1, lack of testing of the test system itself might lead to unreliable test results.

8.2.1 Trouble Keeping the Time Plan

We were able to hold the time plan that we set up in the beginning of the project fairly well. The pre-study and design phase went according to plan, and the first half of the implementation did as well.
In the latter half of the implementation, when we focused on making the functionality implemented in the first half really user friendly, we ran into a few unexpected and time-consuming problems. Chief of these was that using *UpdatePanels* (see section 4.3.4) together with dynamically generated controls (see section 4.3.3) such as drop-down lists and textboxes was not natively supported by the ASP.NET framework which forced us to keep track of all generated controls in catalogues and to regenerate these controls in certain situations. Compared to the simple use of fixed controls this generated quite a few hours of work.

8.2.2 Delayed Hardware Delivery

A further mistake we made was to wait until three months before the project due date to order the server that the system was to be deployed on. Between discussions of the hardware requirements, confusing ordering methods and uncertainty on who was ultimately responsible for making sure everything went according to plan, the final delivery of the server was a mere week before the system was set to be deployed. Luckily, only a few errors that were unique to the deployment environment showed up.

8.2.3 Requirements that were not Fulfilled

One item in the requirements that we were not able to complete was a unit testing coverage of more than 90%. When we realized that time was running out we decided to focus on implementing and correcting the other features in the requirements and see if there was any time left for writing unit tests after that. The low coverage we achieved is extra unfortunate considering the planned formal verification did not take place.

9 Conclusion

This section will give and discuss answers to the research questions and discuss some possible extensions to the system.

9.1 Answers to the Questions Posed in the Problem Definition

In this section we will show how the research and work done answers the research questions stated for the project.

9.1.1 Question 1 – Transition to Mostly Automated testing

> Question 1 asked: "How can we allow for a flexible transition from a mostly manual testing to a mostly automated testing?"

9.1.1.1 Storing Automated Test Results Together with Manual Test Results

The TEA-Coordinator uses the Planning and Result Database of Tier 1 to store the results of automated tests (see section 5.2.3.3). The same database, even the same tables, are used by the web interface in Tier 1 that allows people performing manual tests to enter the results while they are carrying out the test. Ultimately, this means that the person looking at the results in Tier 1 does not have to make any difference between automated and manual test results which allows for an almost seamless transition from manual to automated testing in this respect.

9.1.1.2 Using the Same System for Booking and Tracking Computers Used for Manual Tests as the Computers Used to Execute Automated Tests

In order to have a central point for the automated testing as well as the manual testing work, the same system is used for both booking and tracking computers used for manual tests (see section 4.3.9) as well as booking and tracking the computers used to execute automated tests (see section 4.3.8). By letting users interact with Tier 2 system via the TIME web interface the traditional manual testing could continue as before while the automated testing could be introduced successively by starting small and grow as the number of SUTs in the test bed and the number of test scripts increased.

9.1.2 Question 2 – Minimal Need for Re-training

Question 2 asked: "How can we implement and deploy a highly usable system with a minimal need for re-training and re-structuring?"

9.1.2.1 Using a Consistent and Intuitive Layout Based on an Existing Well-known System

Both the Tier 2 and Tier 1 web interfaces derive their layouts from the Ericsson Internal portal.
Having different layouts would force the user to familiarize him-/herself with two different systems, which is an unnecessary requirement and delays adoption of the new system.

9.1.2.2 Writing In-line Help and Printing Meaningful Status Messages in a Consistent Location

Based on guidelines number 1 and 2 as presented in section 4.2.3.1, the web interface provides in-line help and status messages whenever an action is taken. Any item of in-line help is marked by a question-mark that will show a descriptive text when the user holds the mouse pointer over it. The web interface utilizes this method whenever a previously unused term is introduced or when a deceivingly short term is used for something more complex. Regarding the status messages, the result of any action is shown, be it positive or negative. The status message contains as much information as possible without risking exposure of sensitive information. The messages are color-coded and the location of this message is always at the top of the page below the title.

9.1.2.3 Using a Familiar Terminology

As described in the design section of the web interface (see 4.2.3.1), one of the guidelines for the implementation was to use terms that were already in use at the location of deployment. Using known terms increases the level of familiarity, resulting in an easier adoption of the system.

9.1.2.4 Using a Familiar Work-flow

Early in the pre-study phase (see section 3.1.1) the current way of working in the department was elicited. One of the interview questions inquired about what was wrong with the current way of working. Based on the information elicited from the answers to that question a new work-flow was designed that took the good parts of the old way of working and replaced the malfunctioning parts with new ones. Maintaining the general work-flow also results in an easier adoption of the system.

9.1.3 Question 3 – Handling Changes to the Environment

Question 3 asked: "How can we ensure that the system implemented can handle changes in the environment with minimal or no need for re-implementation?"

9.1.3.1 Allowing Asset Properties to Be Added, Changed, Updated and Removed

As described both in sections 4.4.1 and 6.3.1, the system allows for the properties of assets to be changed in whichever way is necessary. Regarding computers, new operating systems are released all the time, and the user may want to know whether a certain computer supports a certain operating system.

9.1.3.2 Avoiding Hard-coded Values

Most of the parameters that control how the different sub-systems behave have been extracted into configuration files (see section 4.4.2.3). These files contain parameters ranging from "What is the 'from' e-mail address of automated e-mails sent" to "How many milliseconds should the coordinator wait before forcing a state update". Some sub-systems re-read the settings file when it updates, some of them read them on startup only. Regardless, none of them require re-implementation to act on the updated parameter values.

9.1.3.3 Coordinating Other Test Executors

The coordinator does not care which process is on the other end of the socket it uses for communication, as long as it uses and conforms to the TEACUP protocol. Both the protocol and the underlying data-interchange format were deliberately chosen to be easy to implement. Thus, both existing and new software/systems can be modified to be coordinated by the TEATIME system (see section 3.3.5.2). Part of the answer to the original question would be "*By making sure existing and new systems can interact with the implemented system*".

9.2 Future Work

This section will describe some ideas for future extensions to the system. For further ideas, see the wish list in Appendix B.

9.2.1 Have the System Analyze the Configuration of the SUT

There is no guarantee that a test execution leaves the SUT in the state it was before the test was started. The Tier 3 TEXAS Daemon even supports replacing the current operating system with a different one stored in a drive image. There is an obvious risk for complications here, for instance one might expect to have a scheduled test executed on a computer running the Y operating system, but it turns out it is currently running the X operating system, which likely means the results of the test are useless. A feature that has been discussed but not implemented, due to the perceived complexity, is to enable Tier 2 and Tier 3 to exchange information about the current configuration of the SUT. This information could be used both as a regular status update and to be stored along with a test result in Tier 1, showing exactly which configuration the test was executed on.

9.2.2 Fully Automated Execution of Tests Set as Automatic

The current implementation of how Tier 2 fetches tests from Tier 1 requires a user to match the necessary configuration for a test with an asset in the test bed. If the configuration analysis mentioned above is implemented, Tier 2 could automatically match the required configuration to an available one and execute the test without human interaction. It could further be extended to use any free time to re-execute previously executed tests with the hope of finding errors that may not show up in one execution.

10 Bibliography

[1] Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost
Rudolf Ramler and Klaus Wolfmaier, Software Competence Center Hagenberg GmbH. AST'06, May 23, 2006

[2] Advanced Unit Testing
Cyrille Artho (Nat. Inst. of Informatics, Japan) and Armin Biere (Johannes Kepler Univ.), AST'06, May 23 2006

[3] A Test Generation Solution to Automate Software Testing
F. Bouquet (University of Besancon), C. Grandpierre, B. Legeard, F. Pereux (LIERIOS), AST'08, May 11, 2008

[4] Observations and Lessons Learned from Automated Testing
Stefan Berner, Roland Weber, and Rudolf K. Keller (Zühlke Engineering AB and University of Montreal), ICSE'05, May 15-21, 2005

[5] Test Automation for Kernel Code and Disk Arrays with Virtual Devices
Lin-Zan Cai, Rong-Shiung Wu, Wen-Ting Huang, and Farn Wang (National Taiwan University), ASE'07, 2007

[6] A Framework for Practical, Automated Black-Box Testing of Component-Based Software
Stephen H. Edwards (Virginia Tech), Software Testing, Verification and Reliability, Vol. 11, 2001

[7] Functional Testing: A pattern to Follow and the Smells to avoid
Amr Elssamadisy (Gemba Systems) and Jean Whitmore, PLoP'06, 2006

[8] A Framework for Efficient Regression Tests on Database Applications
Florian Haftmann (i-TV-T AG), Donald Kossmann (ETH Zurich) and Eric Lo (ETH Zurich), The VLDB Journal 2007, September 13, 2006

[9] Waterfall model – Wikipedia, the free encyclopedia
URL: http://en.wikipedia.org/wiki/Waterfall_model
Read: 15 January 2009

[10] Scrum (development) – Wikipedia, the free encyclopedia
URL: http://en.wikipedia.org/wiki/Scrum_(development)
Read: 15 January 2009

[11] *Object-Oriented Modeling and Design with UML, Second edition*
Michael Blaha and James Rumbaugh, Pearson Prentice Hall, 2005

[12] Using XML as a flexible, portable test script language
Johnson, D.J. and Roselli, P. AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference, September 22-25. 2003

[13] Test Suite Composition and Cost-Effective Regression Testing
Gregg Rothermel, Sebastian Elbaum, Praveen Kallakuri (Univeristy of Nebraska - Lincoln), Alexey G. Malishevsky, Xuemei Qiu (Oregon State University), ACM Transactions on Software Engineering and methodology, Vol 13, No 3, July 2005

[14] Test Execution Control Tool: Automating Testing in Spacecraft Integration and Test Environments
Michael Levesque, John Louie, Ana Maria Guerrero (Jet Propulsion Laboratory), Aerospace Conference Proceedings, 2000

[15] A Tester's Guide to .NET programming
Randal Root, Mary Romero Sweeney, APress, 2006

[16] Server-Centric Web Frameworks: An Overview
Iwan Wosloo (Reahl Software Services) and Derrick G. Kourie (University of Pretoria), ACM Computing Surveys, Vol. 40, No. 2, 2008

[17] Web application framework – Wikipedia, the free encyclopedia
URL: http://en.wikipedia.org/wiki/Web_application_framework
Read: 18 January 2009

[18] Comparison of web application frameworks – Wikipedia, the free encyclopedia
URL: http://en.wikipedia.org/wiki/List_of_Web_application_framework
Read: 18 January 2009

[19] All Things CakePHP
URL: http://book.cakephp.org/
Read: 18 January 2009

[20] JavaServer Faces Technology
URL: http://java.sun.com/javaee/javaserverfaces/
Read: 18 January 2009

[21] ASP.NET MVC Tutorials
URL: http://www.asp.net/learn/mvc/
Read: 18 January 2009

[22] Ajax Control Toolkit Samples
URL: http://www.asp.net/ajax/ajaxcontroltoolkit/samples/
Read: 15 March 2009

[23] JSON
URL: http://json.org
Read: 14 February 2009

[24] Extensible Markup Language (XML) 1.0 (Fifth Edition)
URL: http://www.w3.org/TR/REC-xml/
Read: 11 May 2009

[25] Software Engineering, Eighth Edition
Ian Sommerville, Pearson Education, 2007

[26] J. Garrett. Ajax: A new approach to web applications
URL: http://www.adaptivepath.com/ideas/essays/archives/000385.php
Read: 12 March 2009

[27] ASP.NET Caching
URL: http://msdn.microsoft.com/en-us/library/ms972379.aspx
Read: 8 March 2009

[28] The Configuration Data Caching Pattern
Leon Welicki, ONO (Cableuropa S.A.U.), PLoP '06, October 21–23, 2006

[29] ASP.NET: 10 Tips for Writing High-Performance Web Applications
Rob Howard, MSDN Magazine, January 2005

[30] Security in Web 2.0 Application Development
Adam A. Noureddine (Microlead Business Solutions) and Meledath Damodaran (University of Houston – Victoria), iiWAS2008, November 24–26, 2008

[31] Log-Based Recovery for Middleware Servers
Wang, Rui, Betty Salzberg, and David Lomet, *SIGMOD'07*, 2007

[32] Design Guidelines for Developing Class Libraries
URL: http://msdn.microsoft.com/en-us/library/ms229042.aspx
Read: 23 February 2009

[33] C# Coding standards
URL: http://www.dotnetspider.com/tutorials/CodingStandards.doc
Read: 23 February 2009

11 Appendices

The following appendices are available:

- Appendix A contains the specification of requirements elicited in the pre-study phase, and subsequently used during the design and implementation phase
- Appendix B contains a list of wishes that were not deemed critical enough to be requirements, but "would still be nice to have"
- Appendix C contains the use cases that were constructed in the beginning of the design phase
- Appendix D lists the questions that were asked during the interview part of the pre-study phase
- Appendix E discusses some programming/coding guidelines that were used during the implementation
- Appendix F contains UML sequence diagrams for the TEACUP protocol
- Appendix G contains the TEATIME User Guide, abbreviated and modified to be suitable for public release

Appendix A. Specification of Requirements

Requirements for the test coordinator

- The system shall have high reliability (availability), 99+%.
- The system shall have a simple platform independent protocol towards the Test Managers.
- The system shall keep track of available DUTs and their status.
- A user shall be able to specify if the user wants an e-mail if the DUT has been unresponsive for the specified time.
- The system shall respond to requests from web interface with the goal of enqueueing tests on DUTs.
- The system shall log the progress and the status of a test while it is executing. The progress shall be displayed on a web page.
- It shall be possible to do backups of the system.
- It shall be possible to run the coordinator on a server separate from that of the web interface.
- The system shall save simplified test results in a database and the full test results as files on the server. The entry in the database shall have a reference to the full test results.
- The system shall be able to continue with minimal data loss if it needs to be restarted.
- The code of the system shall have high (90%+) unit test coverage.

Requirements for the web interface

- Current Ericsson ids shall be possible to use to give users the correct user rights and access. "User" and "Administrator" shall be possible to set as an attribute in a database.
- The system shall initially be able to handle a minimum of 100 users and 10 administrators but the system shall be scalable to larger user bases as well.
- An asset shall be possible to check in or out from a storage location with the use of a bar code and a bar code reader.
- It shall be possible to book assets using a web interface. If the asset in questions is already checked out the user shall be put in a queue and notified by e-mail when the asset is available.
- It shall be possible to give assets generic properties using a web interface. If new types of properties need to be added it shall be possible to do so via the web interface.
- A history of which users an asset has had, and vice versa, shall be possible to view on a web page.
- A history of reported and fixed asset issues shall be possible to view on a web page.
- The system shall be able to generate new bar codes for assets that are added to the system.

- Tasks that can only be performed by an administrator shall only be accessible by an administrator, via a link from the pages relating to said task. The link shall only be visible to administrators.

- Users shall be able to read and update the status of an asset. Updates can only be done when the asset is checked out by the user and when checking in assets.

- If an asset requires service, an e-mail is automatically sent to the appropriate person.

- It shall be possible to force a user to enter certain asset attributes when checking in an asset. (Specifically entering firmware when checking in a broadband module)

- The system shall be able to import and export asset data from Excel files, directly or via CSV-files.

- A user shall, via the system, be able to book and automatically execute tests using the currently developed test automation tool. The test results shall be possible to send as a long summary via e-mail and a short summary (max 160 characters) via SMS.

- Data from a previously existing Excel document shall be entered into the database.

- There shall exist a set of relevant pre-configured searches. Free text search shall also be possible.

- Raw SQL search shall be possible using select statements. A user shall be able to save the SQL query with a descriptive name which makes it available to all users.

- When booking an asset, the user shall be forced to enter a booking duration. If the duration exceeds an existing maximal duration, a notice shall be sent to an administrator. The notice shall include a link to a web page with information on the booking, with buttons to accept or deny the loan.

- An e-mail shall be sent to the booked user when the loan duration of an asset has expired. An e-mail shall also be sent to the asset owner if the asset has not been checked in 48 hours after expiration.

- A user shall be able to extend the loan duration of an asset if that asset is not already booked by another user.

- The web interface shall be functional in Microsoft Internet Explorer 7.0 and Mozilla Firefox 3.0.

- The source code of the system shall follow existing Ericsson programming guidelines and be well documented.

- A technical description and in-line help on the web pages shall be written for the system.

- A how-to shall be written and put on EriColl.

- Text strings in the web interface shall be "resources", not hard-coded.

- If a booked asset is deemed unusable, the users who are on queue for that asset shall be notified via e-mail. The e-mail shall contain a link to the search page that shows working assets of the same AssetTypeGroup.

- A bar code reader and a bar code printer shall be pre-evaluated and purchased.
- The system shall respond to search requests within 30 seconds.
- The system shall respond to other requests within 3 seconds.

Appendix B. System Functionality Wish-list

In this appendix you will find a list of functionality that has been asked for but not put in the requirements, together with a motivation and an indication showing whether or not they have been implemented. The wishes are grouped by what sub-system they belong to.

TEA-Coordinator

1) It should be possible to book an asset that is part of the test-bed. Having exclusive access to a computer for running automated tests enables a user to run a test, check the results and decide what test to run next, knowing that the system has not been changed since the last test. *Implemented.*

2) If a test causes the system to fail, the person who booked the test should receive an e-mail. There is a good chance the person booking the test is "listening" and can handle the situation. *Not implemented.*

TIMEWI

1) The start page should show upcoming tests scheduled by the user. Having them easily accessible from the start page allows for quick cancelling of unwanted tests. *Implemented.*

2) Show an animated symbol when searching. Showing an animated symbol indicates to the user that the system has received the request and is working on it. *Implemented.*

3) It should be possible to copy a previous asset when creating a new one. It is likely that several assets of the same type are added simultaneously, making copies would save time. *Implemented.*

4) When attempting to book an asset that is currently occupied, the system should show free time slots instead of busy time slots. *Not implemented.*

5) The system should be switched from using standard SQL statements to using parameterized queries. Parameterized queries help in countering SQL injection attacks and increases performance somewhat. *Not implemented.*

TIMEDB

1) Instead of having a comment field, like the current system, the database should have a comment history. Allowing a comment history gives a user easy access to a list showing everything that has been reported for this asset, information that can be used to decide whether the asset is suitable. *Implemented.*

2) The view used in the database should be indexed. Indexing views costs a bit more on updating but is cheaper when selecting. *Not implemented.*

3) Some information in the database should be accessible to administrators only. While the data currently in the system is the same as in the public lists, there may be sensitive data later on. *Not implemented.*

Appendix C. System Use Cases

UC1: Logging in	
Authors: Elin Weber, Erik Sternerson *Event:* User logs in *Preconditions:* User has a valid Ericsson id (signum) and is entered into the database *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user logs in to his/her computer and visits the web interface. The web interface uses the domain credentials of the visiting user to get the signum and then looks it up in its own database to find the user details. The user gets to the start page. *Related Use Cases:* -	
Typical Process Description	
User Actions	**Web Interface Actions**
1. User logs in to computer	
2. User browses to web interface	
	3. Web interface receives page request
	4. Web interface extracts domain credentials from page request
	5. Web interface fetches user details for signum
	6. Web interface displays start page for user
Exception #1: The user is not in the database	
	[5-6] The web interface displays a message to the user saying that the user needs to be registered by an administrator.

UC2: Searching for assets	
Authors: Elin Weber, Erik Sternerson *Event:* User searches for an asset *Preconditions:* User is logged in *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user goes to the search page and enters a set of search criteria. The web interface lists all the assets matching said criteria. *Related Use Cases:* UC1	
Typical Process Description	
User Actions	**Web Interface Response**
1. User browses to search page	
	2. Web interface displays search page
3. User enters information on the predefined criteria	
4. User clicks search	
	5. The web interface queries the database for the assets matching the criteria
	6. The web interface updates the search page with the search results, leaving the search criteria boxes filled in
Alterative #1: Searching for assets using dynamic properties	
3. The user needs to specify search criteria other than those that are predefined	
3.1 The user selects an asset group or asset type from a drop down list	
	3.2 The web interface reads the asset type group or asset type and queries the database or cache for the property data types relating to said asset type group or asset type
	3.3 The web interface displays a list of the properties defined for the selected asset type group
3.4 The user enters any property criteria to be used in the search	
Alternative #2: The user searches by an SQL query	
See UC5	

UC3: Managing the search results	
Authors: Elin Weber, Erik Sternerson *Event:* User manages the results of a search *Preconditions:* User has performed a search *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user manages the search result by selecting, sorting or requesting details. (The last action is not supported by the SQL search). *Related Use Cases:* UC2	
Typical Process Description	
User Actions	**Web Interface Actions**
1. The user presses the heading of a column	
	2. The web interface updates the page with the result set ordered by the heading that was pressed
Alternative #1: The list was already sorted by the heading that was pressed	
	2. The web interface updates the page with the result set ordered in reverse by the heading that was pressed
Alternative #2: User requests details for an asset	
1. The user clicks on the value in the ”Marking” column	
	2. The web interface forwards the user to the Asset Details page (see UC6)
Alternative #3: User books one or more assets	
1. The user selects one or more assets by clicking the checkboxes tied to the assets.	
2. The user presses the ”Book assets” button	
	3. The web interface forwards the user to the booking page with the assets id values pre-filled (see UC4)
Alternative #4: The users books one asset	
1. The user clicks the ”Book” link next to the asset	
	2. The web interface forwards the user to the booking page with the asset id value pre-filled (see UC4)

UC4: Booking assets	
Authors: Elin Weber, Erik Sternerson *Event:* The user books an asset *Preconditions:* The user has selected one or more assets, either by attempting to check them out or by searching. The asset to be checked out is NOT part of the automated test bed, unless it is specified to be bookable. *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user enters purpose, start and end date and time for the booking. If the booking is possible, the system records details on the booking in the database. If not, the user is shown a message detailing what went wrong. *Related Use Cases:* UC3	
Typical Process Description	
User Actions	**Web Interface Actions**
	1. The web interface constructs a calendar for the booking start date and a text field for the time. The pre-selected date and time is set to now
	2. The web interface fetches the default booking duration and default purpose from the settings-file or cache
	3. The web interface constructs a calendar for the booking end date and a text field for the time which is pre-selected to the booking start date + the default booking duration
	4. The web interface displays a booking page with a drop down for the reason for the booking, with the default purpose pre-set, and the calendars
5. The user presses the "Book" button	
	6. The system records in the database the details of the booking
	7. The system displays a message saying that the asset(s) were booked
Alternative #1: The user wants to change the start date	
5.1 The user changes the start date to suit his/her need	
	5.2 The system reads the value from the start date calendar and start time text field.
	5.3 The system updates the value of the end date calendar and text field to the new start date and time value + default booking duration
Alternative #1.1: The user wants to change the end date	
5.4 The user changes the end date and time to suit his/her need	
Alternative #1.2: The user wants to change the purpose	
5.5 The user selects a purpose from the drop-down list	
Alternative #1.2.1: The user wants to set a free-text purpose	
5.5.1 The user selects "Other" from the drop-down list	
	5.5.2 The web interface updates the page with a text box allowing free-text purpose entry
5.5.3 The user enters the purpose of the booking	
Exception #1: The asset is already booked	
	1.1 The web interface displays a message saying that the asset is already booked
	1.2 The web interface displays a list of future bookings for the asset.

Exception #2: The booking time is longer than the maximum allowed time	
	7. The system displays a warning message saying that the booking is longer than the maximum allowed length, and that an administrator can cancel the booking at any time.

UC5: Managing SQL queries	
Authors: Elin Weber, Erik Sternerson *Event:* User executes, saves, edits and removes SQL queries. *Preconditions:* User is logged in *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user manages a SQL query by either execute or save it. In the case when the user is the creator of the SQL query to manage s/he can also edit or remove it. *Related Use Cases:* UC1	
Typical Process Description	
User Actions	**Web Interface Actions**
1. The user browses to the SQL search page	
	2 The web interface displays the SQL search page with a drop down list of previously saved queries and a text filed for query input
3. The user enters a new SQL query using a SELECT statement	
4. The user enters a time limit for the search execution in a text box	
5. The user presses the ”Execute query” button	
	6. The web interface receives the query
	7. The web interface checks the query for unsafe SQL statements (only SELECT statements allowed).
	8. The web interface sends the query to the database
	9. The web interface receives the result from the database
	10. The web interface updates the SQL search page with the search results within the time limit
Alternative #1: User saves an SQL query	
5. User presses the ”Save query” button	
	6. The web interface receives the query
	7. The web interface checks the query for unsafe SQL statements
	8. The web interface displays a text box with the caption ”Query title/description” and a ”Save” button
9. The user enters a title/description for the query	
10. The user presses the ”Save” button	
	11. The web interface saves the query in the database with the given title/description
	12. The web interface updates the page and removes the text box and the ”Save” button. The saved query is shown in the query input window. The title/description is shown above this window.

Alternative #1.1: User saves the query as ”Private”	
10.1 The user checks the ”Private query (only visible to you)” checkbox	
10.2 The user presses the ”Save” button	
Alternative #2: User loads a saved query	
3. User selects a query from the drop-down list	
	4.1. The web interface reloads the page with the selected query in the SQL window
4.2. The user presses the ”Execute query” button	
Alternative #2.1: User saves a loaded query under a new name	
4.2.1 The user presses the ”Save query” button	
	4.2.2 Return to Alt1 step 5
Alternative #2.2: A user updates a loaded query	
4.2.1. The user presses the ”Update query” button	
	4.2.2 Do Alt1 steps [5-9]
	10. The web interface updates the query in the database
Alternative #2.3: A user removes a loaded query	
4.2.1. The user presses the ”Remove query” button	
	4.2.2 The web interface removes the query from the database
Exception #3.1: The SQL query contains unsafe SQL statements	
	[8-10] The web interface displays a warning to the user saying that the SQL query contains unsafe SQL statements.
Exception #2.2.1: The user is not the creator of the query	
	[4.2.1 – 4.2.2] The web interface does not show the ”Update query” button or the “Remove query” button..
Exception #2.3: The user is the creator of the query	
	4.1 The web interface reloads the page with the selected query in the SQL window and shows the ”Update query” button and the “Remove query” button.

UC6: Looking at asset details and history	
Authors: Elin Weber, Erik Sternerson *Event:* User wants detailed information on an asset *Preconditions:* The user has been given a link to the detail page of the asset *System:* Web Interface *Actors:* User *Overview/Postcondition:* The system will display information of the asset *Related Use Cases:* UC3	
Typical Process Description	
User Actions	**Web Interface Actions**
1. The user clicks the link to the asset detail page	
	2. The web interface fetches all the properties and their values for the asset from the database
	3. The web interface fetches the last five service entries for the asset from the database
	4. The web interface fetches the last five test suites and booking entries for the asset from the database.
	5. The web interface displays the information above on the Asset Details page

UC7: Checking out assets	
Authors: Elin Weber, Erik Sternerson *Event:* A user checks out an asset *Preconditions:* The user is logged in. *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user selects an asset to check out. *Related Use Cases:* UC1	
Typical Process Description	
User Actions	**Web Interface Actions**
1. The user browses to the check-out page	
	2. The web interface displays the check-out page with the marker in the ”Marking or bar code”-field
3. The user reads the bar code of the asset with a bar code reader	
4. The user presses the ”Check out” button	
	5. The system receives the marking or bar code.
	6. The system fetches the asset data and the bookings for the asset from the database
	7. The system displays the booking page
Alternative #1: User has already booked the asset	
	7. The system displays a message saying ”The system has successfully checked out the asset to you”
Alternative #2: The user uses the keyboard to enter the asset marking	
3. The user enters the marking of the asset using the keyboard	
4. The user presses the ”Check out” button	
	5. The system receives the marking
Exception #1: The asset is in a current booking, or has a booking in the next hour, by a user other than the one who is checking out the asset.	
	7.1. The system displays a warning message saying ”The asset you are attempting to check out is booked by [First name, Last name] [at #time# (xx minutes from now)]”
	7.2 The system displays two links, ”Go to the search page”, ”Book asset anyway”

UC8: Checking in assets	
Authors: Elin Weber, Erik Sternerson *Event:* The user checks in an asset. *Preconditions:* The user is logged in and has checked out the asset(s) to be checked in. *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user enters the bar code number or marking of the asset. The system displays a page with all the required check-in-information and ends the booking of the asset by that user. *Related Use Cases:* UC1	
Typical Process Description	
User Actions	**Web Interface Actions**
1. User browses to the check-in page	
	2. The web interface displays the check-in page with the marker in the ”Marking or bar code” field
3. The user reads the bar code with a bar code reader	
4. The user presses the ”Continue” button	
	5. The system receives the marking or bar code
	6. The system fetches the information on the asset from the database
	7. The system displays a page with a drop down list of available statuses.
8. The user selects a status of the asset	
9. The user presses ”Check-in”	
	10. The system ends the booking by the user of the asset.
Alternative #1: The user enters the marking via keyboard	
3. The user enters the marking	
4. The user presses the ”Continue” button	
	5. The system receives the marking
Alternative #2: The asset has properties that require update on check-in	
	7. The system displays a page with a status drop down list and for every required property it displays the property name, a text box with the previous value and a check box with the label ”use previous value”
8. The user enters values for all the required properties and a new status	
Alternative #3: The user has performed service on the asset (The asset must be in a state that requires service)	
	7. The system displays a page with a status drop down list and a drop down and text field with service type and service comment.
8. The user enters the status, service type and service comments for the asset	

Alternative #4: The user want to use another status than those who are pre-defined	
8. The user selects “Other” from the status drop-down list	
	8.1 The web interface updates the page with a text box allowing free-text status entry
8.2 The user enters the status of the asset	
Exception #1: The asset is checked out to someone else	
	10. The system displays a warning message saying ”The asset you are trying to check in is checked out by [First name, Last name]. Make sure you have the approval of that user before you check it in.” and “Continue” and “Cancel” buttons.
Exception #2.1: The user missed one of the required properties	
	10.1. The system displays a message saying ”The asset you are trying to check in requires that you update the [property name] property. You left the old value without checking the ”Use previous value” checkbox. Either write a new value or check the box, then try again.”
	10.2 Return to Alt 2 step 7

UC9: Exporting search result to Excel	
Authors: Elin Weber, Erik Sternerson *Event:* The user exports the result list of a search to an Excel document. *Preconditions:* The user is logged in and has searched for, and found, some assets. *System:* Web Interface *Actors:* User *Overview/Postcondition:* The system generates an Excel document containing all the asset and property data on the assets that are part of the search result. *Related Use Cases:* UC2	
Typical Process Description	
User Actions	**Web Interface Actions**
1. The user clicks the ”Export to Excel” link	
	2. The web interface receives the request
	3. The web interface fetches the search result
	4. The web interface generates an Excel document containing said result.
	5. The web interface saves the Excel document to a temporary folder
	6. The web interface displays a page with a link to the Excel document and a note that the document will be available [pre-set time] ahead and then be deleted
7. The user clicks on the link to download the document	

UC10: Updating status of an asset	
Authors: Elin Weber, Erik Sternerson *Event:* The user updates the status of an asset *Preconditions:* The user is logged in. The asset is checked out to the user logged in. *System:* Web Interface *Actors:* User *Overview/Postcondition:* A user updates the status of an asset that s/he has checked out *Related Use Cases:* UC7, UC8	
Typical Process Description	
User Actions	**Web Interface Actions**
1. The user browses to the update status page for a checked-out asset via a link on the start page	
	2. The web interface forwards the user to the second part of the check-in page, but the ”Check-in” button is replaced by an ”Update” button, and the system does not end any bookings.
	3. Go to UC8 step 6

UC11: Extending the length of a booking	
Authors: Elin Weber, Erik Sternerson *Event:* The user extends the length of a booking *Preconditions:* The user is logged in and has an asset that is currently in a booking created by that user. *System:* Web Interface *Actors:* User *Overview/Postcondition:* The system allows the user to enter a new booking end date and records it to the database *Related Use Cases:* UC1, UC4	
Typical Process Description	
User Actions	**Web Interface Actions**
1. The user clicks the ”Extend booking” link for the asset on the start page	
	2. The web interface displays the booking page with the start calendar read-only (”grayed out”) and the start time field disabled.
	3. Go to UC4 Alt 1.1

UC12: Scheduling automated tests on assets		
Authors: Elin Weber, Erik Sternerson *Event:* The user schedules an automated test to be run on an asset. *Preconditions:* The user has searched for assets in the search page. The asset that the test will run on IS part of the automated test bed and NOT specified to be bookable. *System:* Web Interface *Actors:* User *Overview/Postcondition:* The user enters test information and schedules what test to be run on what assets. The user can select test from a pre-defined list of test or type in a path to its own test file. *Related Use Cases:* UC2		
Typical Process Description		
User Actions	**Web Interface Actions**	**Coordinator Actions**
1. The user clicks the ”Schedule test” link on the search page		
	2. The web interface fetches a list of pre-defined tests	
	3. The web interface fetches a list of tests in queue for the asset	
	4. The web interface displays the pre-defined tests in a multiple-select box and the tests in queue as text on a scheduling automated test page.	
5. The user types in the purpose of the test scheduling		
6. The user selects one or more tests		
7. The user types in number of iterations		
8. The user chooses what type of notification of test result that s/he wants by checking the check boxes “Notification by email” and “Notification by SMS”		
9. The user presses the ”Execute tests” button		
	10. The web interface reads the list of pre-defined tests.	
	11. The web interface stores the list of tests to be run in the database	
		12. The coordinator fetches the test data from the database and executes the test

Alternative #1: The user wants to upload a self-specified test		
6. The user presses the ”Browse...” button.		
7. The user selects a test specification file		
8. The user presses the ”Add test” button		
	9. The web interface receives the test specification file	
	10. The web interface validates the test specification file with an XML schema	
	11. The web interface sends the test file to the coordinator	
		12. The coordinator receives the test file
	13. The web interface displays a message saying ”Test added”	
	14. The web interface adds the test to the list of tests	
15. The use presses the ”Execute test” button		
	16. Return to step 10 in main flow	
Alternative #1.1: The user wants to upload more than one test		
[6 - 8]. The user repeats the upload procedure as many times as s/he wants to upload more than one test		
Alternative #1.2: The uploaded test is marked as private		
15.1. The user checks the box “The uploaded test is private”		
15.2. The use presses the ”Execute test” button		
Exception #1.1: The test specification is invalid		
	[11-14] The web interface displays a message saying ”The specified test file is invalid!”	
	14. Return to step 4 in main flow	

UC13: An administrator adds a new record in the database	
Authors: Elin Weber, Erik Sternerson *Event:* An administrator is logged in. S/he adds a new record of any of the types: asset, asset type, asset type group, manufacturer, service type, asset owner, user, user group or category to the database via the administration interface. The type of record to add is selected in a list containing a set of tables in the database. In some cases several database tables will be affected when the information about a new item is stored. (For instance when a new asset is added, information is stored in both the table for assets and the table for holding information about properties.) *Preconditions:* Administrator is logged in *System:* Web Interface *Actors:* Administrator *Overview/Postcondition:* An administrator adds a new record in the database on the admin page. The type of record is selected by a drop down list of all tables in the database. If an asset is to be added, the barcode can be generated by clicking on a button on the page. In some cases several tables will be affected by the adding action. *Related Use Cases:* UC1	
Typical Process Description	
Administrator Actions	**Web Interface Actions**
1. Administrator browses to the admin page	
	2. Web interface shows the admin page
3. Administrator uses the drop down list for all tables in the database and selects the one where s/he wants to add a new record	
	4. Web interface displays editable fields for the information required (columns in database table) for the selected table and shows all existing records in a list view on the same page
4. Administrator fills in all required information in editable fields (text boxes and drop-down lists) based on the table selection	
5. Administrator clicks “Save” to store the new record	
	6. Web interface connects to and store information as a new record in the database and updates the attributes ChangedBy with the signum of the administrator and LastChanged with a time stamp for the table.
	7. Web interface notifies administrator about the positive outcome of the action, does a post back of the page, shows the new record in the list box and empties the text fields

Alternative #1: The new record is an asset	
4. Administrator selects “Assets” from the drop down list of database tables	
	5. Web interface shows a part of the page with a drop down list with existing asset type groups and a division with editable text fields for asset information (i.e. marking) and drop-down lists with available data for the rest of the information (i.e. status and category) .
6. Administrator selects an asset type group	
	7. Web interface shows a drop down list with existing asset types for the selected asset type group
8. Administrator selects an asset type	
9. Administrator fills in all information by using editable text fields and drop down lists	
10. Administrator presses the button “Generate properties”	
	11. Web interface shows the name of the available properties and an editable text field for each property value for the selected asset type group
12. Administrator types in the property values	
13. Administrator presses the button “Generate bar code”	
	14. Web interface shows a pop-up showing the bar code and the two buttons “Print” and “Cancel”
15. Administrator presses the “Print” button	
	16. Web interface shows the print dialog window
17. Administrator selects the label printer and presses the “Print” button in the print dialog window	
18. Return to step 5	
Alternative #2.1: Asset type group is missing	
6. Administrator can’t find the asset type group and goes back to step 3 in the main flow to first add an asset type group	
Alternative #2.2: Asset type is missing	
8. Administrator can’t find the asset type and adds the asset group as described in Alt 3	
Alternative #2.3: Administrator cancels print action of bar code	
15. Administrator presses the “Cancel” button	
16. Return to step 18	

Alternative #3: Administrator adds a new asset type group	
4. Administrator selects “Asset Type Groups” from the drop down list of tables	
	5. Web interface shows “Name of Asset type group” and an editable text field
	6. Web interface shows a division with editable text for information required to add a new property type (attributes DefaultValue and isRequiredInfo)
	7. Web interface shows a drop down list of all existing property data types and at the top of the list there is an alternative “Add new…”
8. Administrator selects “Add new...” in the drop down list	
	9. Web interface shows two editable text field and the labels “Property data type name:” and “Property data type value:” and a check box saying “This property data type has a range of values”
10. Administrator enters the name of the asset type group	
11. Administrator types in the required information for a new property type	
12. Administrator enters a property data type name	
13. Administrator enters a property data type value	
14. Return to step 5	
Alternative #3.1: Property data type value is a range of values	
11. Administrator checks the check box “This property data type has a range of values”	
	12. Web interface changes the text box with label “Property data type value:” to non-editable and shows two other editable text fields with the labels “Value1” and “Value2” and an “Add value” button
13. Administrator fills in the two text boxes for “Value1” and “Value2”	
14. Return to step 5	
Alternative #3.1.1: Property data type value is a range of more than 2 values	
13. Administrator presses the ”Add value” button	
	14. Web interface shows a third editable text field with the label “ValueX” (where X is the number of previous value field incremented by 1)
15. Administrator fills in the new text box for “ValueX”	
16. Return to step 5	
Alternative #4: Negative outcome of save action	
	7. Web interface notifies administrator about the negative outcome of the action
8. Administrator corrects the errors described in the message	
9. Return to step 5	

UC14: Editing/removing a record in the database (Administrator)	
Authors: Elin Weber, Erik Sternerson *Event:* An administrator edits or removes a record of any of the types: asset, asset type, asset type group, manufacturer, service type, asset owner, user, user group or category in the database via the administration interface. To be able to remove a record some requirements, such as the item not being currently used by the system, have to be fulfilled. The administrator can put an asset in edit mode by using the edit link on the asset details page or in the administration interface directly. *Preconditions:* Administrator is logged in *System:* Web Interface *Actors:* Administrator *Overview/Postcondition:* An administrator edits or removes a record in the database by using the admin page. To be able to remove a record some requirements have to be fulfilled. The administrator can also edit an asset via the asset details page. *Related Use Cases:* UC1	
Typical Process Description	
Administrator Actions	**Web Interface Actions**
1. Administrator browses the admin page	
	2. Web interface displays the admin page
3. Administrator uses the drop down list for all database tables to select the one where s/he wants to edit or delete a record	
	4. Web interface displays editable fields for the information required (columns in database table) for the selected table and a data view that shows the information of all the existing records in the selected table
5. Administrator selects an existing record in the data view	
6. Administrator presses the "Edit" button next to an existing record	
	7. Web interface queries the database for the information about the selected record
	8. Web interface receives information and displays it in the editable text boxes, drop down lists etc. on the page
	9. Web interface displays a "Cancel" button and an "Update" button
10. Administrator changes the information in the fields	
11. Administrator presses the "Update" button	
	12. Web interface connects to and store changes in the database and updates the attributes ChangedBy with the signum of the administrator and LastChanged with a time stamp
	13. Web interface notifies administrator about the positive outcome of the action, shows the changes (if it is in any of the visible columns) in the list view on the page

Alternative #1: Administrator wants to remove a record	
6. Administrator presses the "Delete" button next to an existing record	
	7. Web interface shows a pop-up message saying "Do you really want to delete selected record?"
8. Administrator presses "Yes"	
	9. Web interface queries database to delete the selected record
	[10-13]. Web interface shows a message that the action was successful
Alternative #1.1: Administrator cancels remove action	
8. Administrator presses "No"	
	9. Web interface hides the pop-up and goes back to normal
Alternative #1.2: Administrator can't remove a record	
	10. Web interface shows a message that the action was not successful and shows a message with an explanation to the error (eg. "Asset type can't be deleted because it is in use")
Alternative #2: Administrator wants to cancel the edit action	
[10-11] Administrator presses the "Cancel" button	
	12. Web interface empties the editable fields and replaces the "Cancel" button and the "Update" button with a "Save" button.
Alternative #3: Negative outcome of save action	
	9. Web Interface notifies administrator about the negative outcome of the action
10. Administrator corrects the errors described in the message	
11. Return to step 7	
Alternative #4: Administrator edits a record via the asset detailed information page	
1. Administrator browses to the detailed information page for an asset	
	2. Web interface shows the detailed information page for an asset
3. Administrator presses the "Edit" button	
	4. Return to step 7 in main flow
Exception #1: No records in the selected database table	
[5-13] Administrator does not have anything to edit (or delete) and chooses a different action	

UC15: Receiving an email about a longer booking (Administrator only)	
Authors: Elin Weber, Erik Sternerson *Event:* An administrator receives an email about a longer booking and cancels the booking via attached link. *Preconditions:* *System:* Web Interface *Actors:* Administrator *Overview/Postcondition:* An administrator receives an email about a longer booking and has the possibilities to see the booking information and cancel the particular booking via a link in the email. *Related Use Cases:* -	
Typical Process Description	
Administrator Actions	**Web Interface Actions**
1. Administrator receives an email saying that a longer booking has been placed. The email contains information about the booking and a link to that will cancel the booking.	
2. Administrator clicks on the link that cancels the booking	
	3. Web interface cancels the particular booking and sends an email to the responsible user.

Appendix D. Questionnaire Used for Requirements Elicitation

Questions about management and booking of assets:

1. Describe the problems of the current system.
2. What functionality of the proposed system are you going to use the most?
   In other words, what functionality must be most easily accessible?
3. What kind of information would you like to see on the web interface start page?
4. How often do you think you will use the system?
5. What information is important to show in the general search result directly, rather than going to a separated detail page for an asset?
6. Do you have any thoughts or opinions about the presentation? (Demonstration of prototype 1 of the web interface)

Questions about test execution

1. Describe the problems, concerning testing, of the current system.
2. What functionality of the proposed system are you going to use the most?
   In other words, what functionality must be most easily accessible?
3. How often do you think you will use the system with the purpose of executing tests?
4. How would you like to get the results of the tests?
    a. Status on a web page?
    b. Information in emails, if yes: what information?
    c. Information in SMS, if yes: what information?

Appendix E. A Subset of the Coding Practices Used, with Examples

Do not hardcode magic values. Use constants instead.

ASP.NET provides several different global or semi-global hash-tables that use strings for keys. We use a static class containing constant strings as keys for these tables. A short example follows
In Constants.cs:

```
public const string SESSION_CURRENT_USER = "CurrentUser";
```

In some other class

```
User currentUser = (User) Session[Constants.SESSION_CURRENT_USER];
```

Looking up hash-tables this way helps in at least two ways. First, if the key is used in more than one place (likely), there is less risk of errors due to misspelling the key. The compiler would catch a misspelled class member, so the only way would be to select the wrong member, a risk that can be minimized by giving the constant keys descriptive names. Secondly, code-completion can be used to fill in the value of the key. Instead of typing "CurrentUser" (with quotations), the key can be fetch with a flow similar to "Type 'Co', press ctrl+space, type 'S', press down+down+enter".

Do not hardcode strings. Use resource files.

We have used both resource and settings files for our sub-systems. The settings files in particular give a good increase in maintainability, since there is no need for a recompilation of the application for some changes, just a restart.

Make sure you have a good logging class which can be configured to log errors, warning or traces.

The coordinator contains a logger class that provides two public methods:

```
public void Log(LOG_LEVEL level, string message);
public void Debug(string message); //Shorthand for Log(LOG_LEVEL.DEBUG,
msg);
```

If the specified level is lower than the level specified in the settings, the message is discarded. If not, reflection is used to get the name of the calling class, which is printed together with the message to a log file.

Do minimal work in the constructor. The cost of any other processing should be delayed until required.

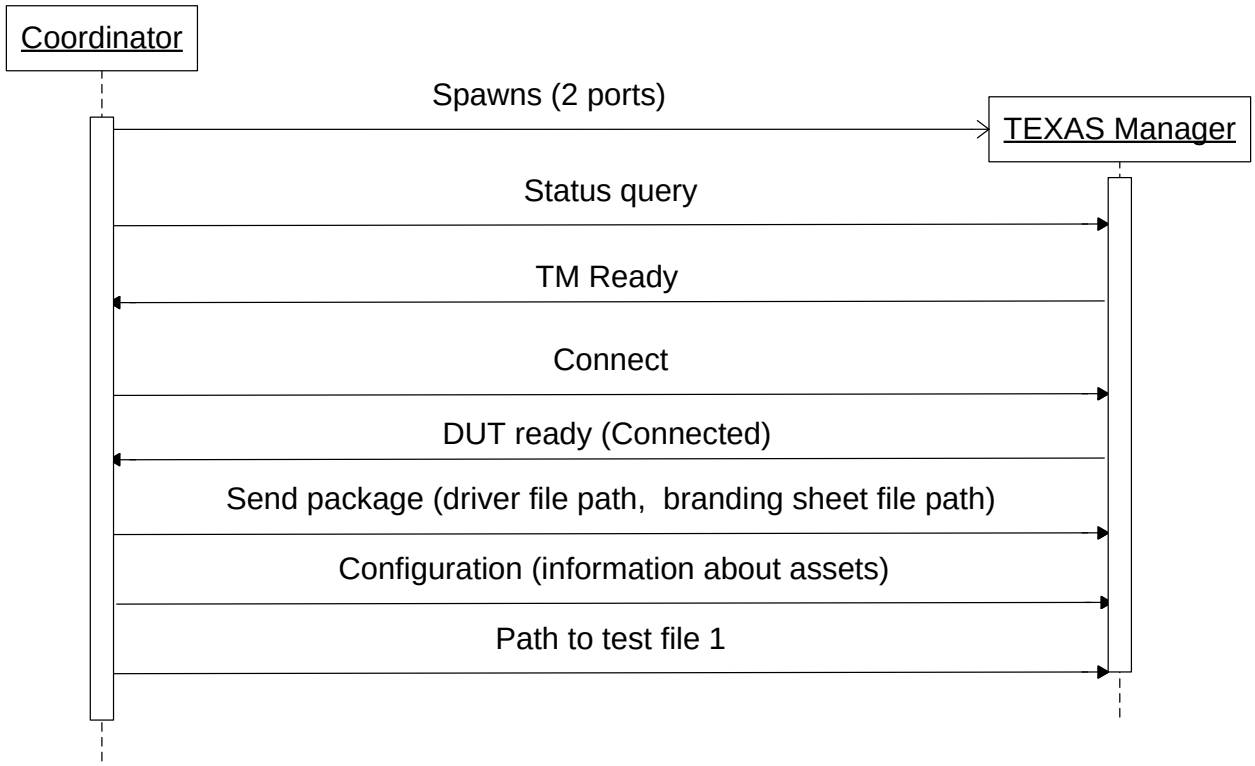
One of the most resource intensive modules in the system is the test execution module. The constructor of that class is only responsible for putting the class in a state ready for execution, using a minimal amount of work. To start the test execution, a separate method is used, aptly named runExecutor. Since each test execution is run in its own thread, but is set up by the controller thread, having the work divided this way helps ensuring that the controller thread is not overloaded.

Other practices used.

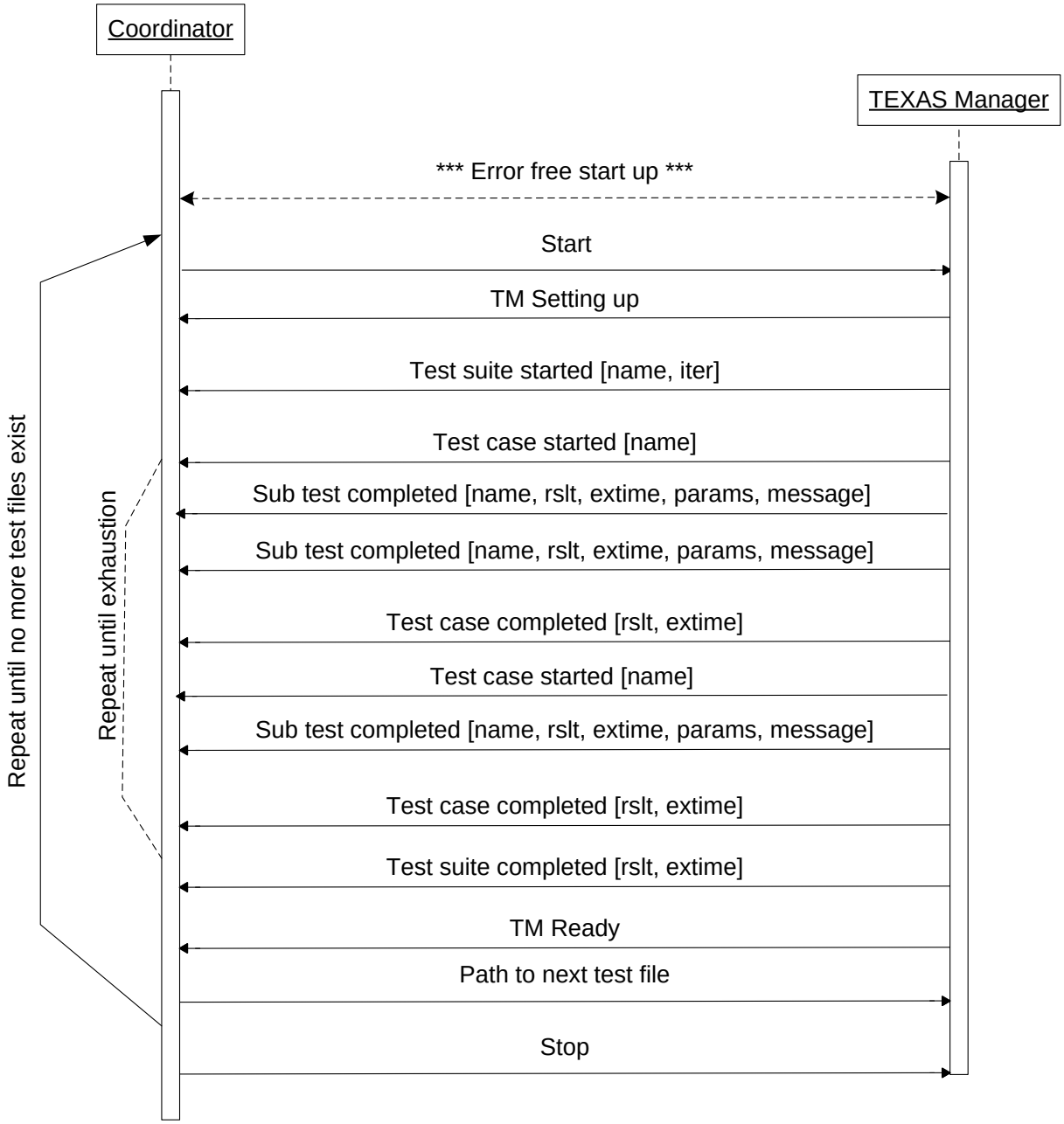
There is of course a lot more practices that we used besides the ones mentioned above. Some are not mentioned because they are not very interesting (naming conventions, comment conventions, etc.) and others are not mentioned because it's hard to give a good explanation on how we use them (interface design conventions, etc.)

Appendix F. TEACUP Sequence Diagrams

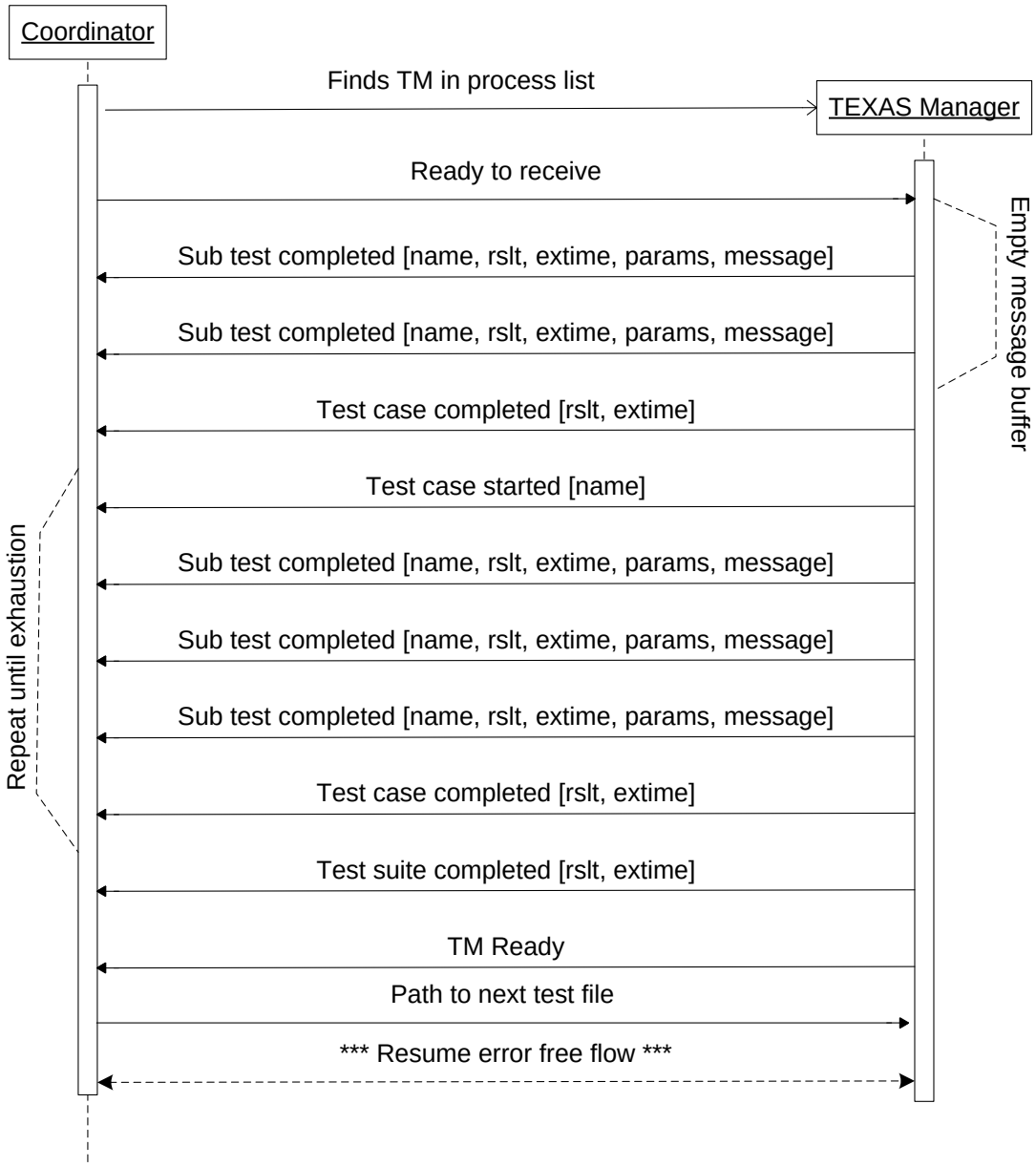
Error-free startup



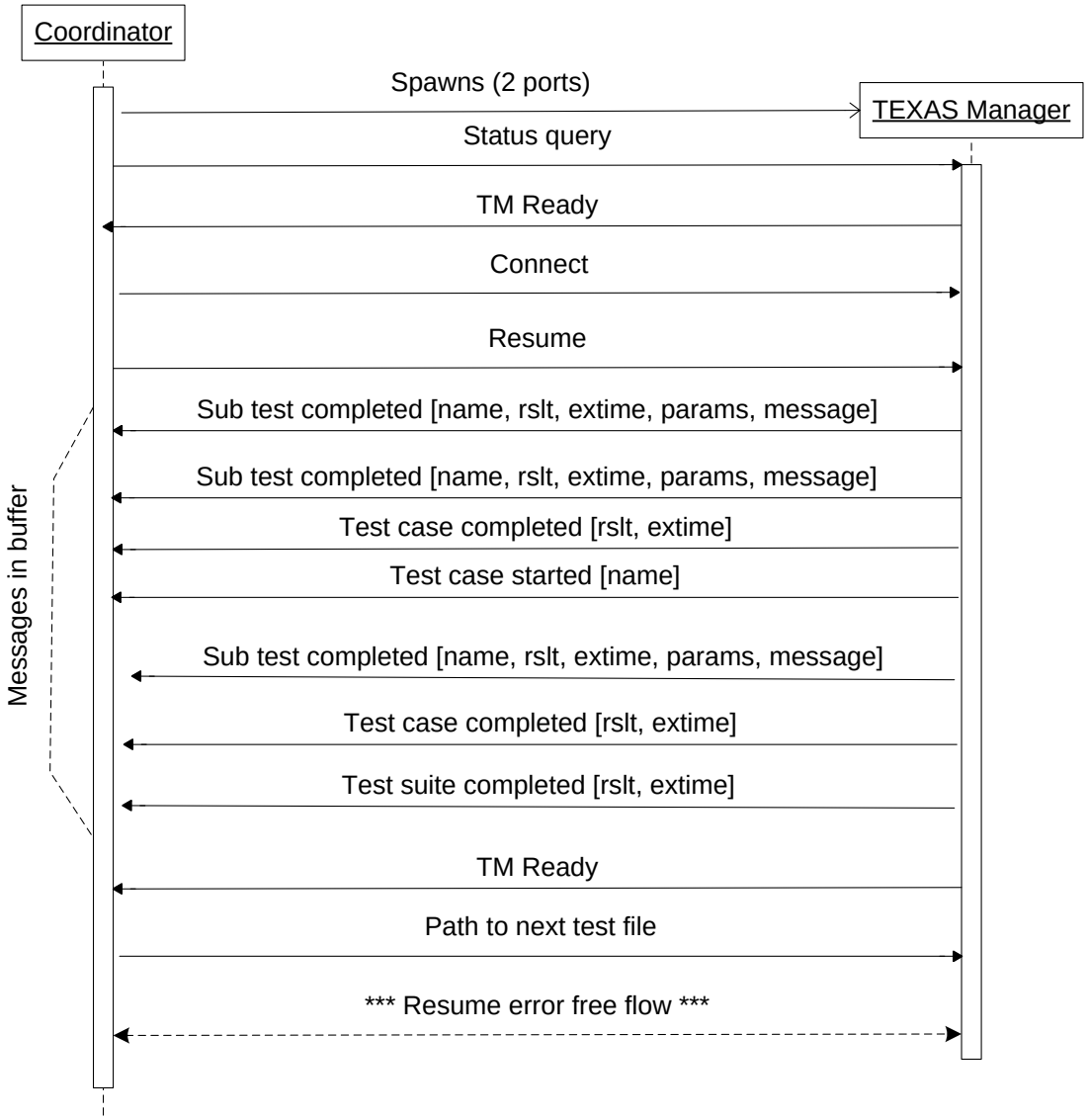
Error-free flow



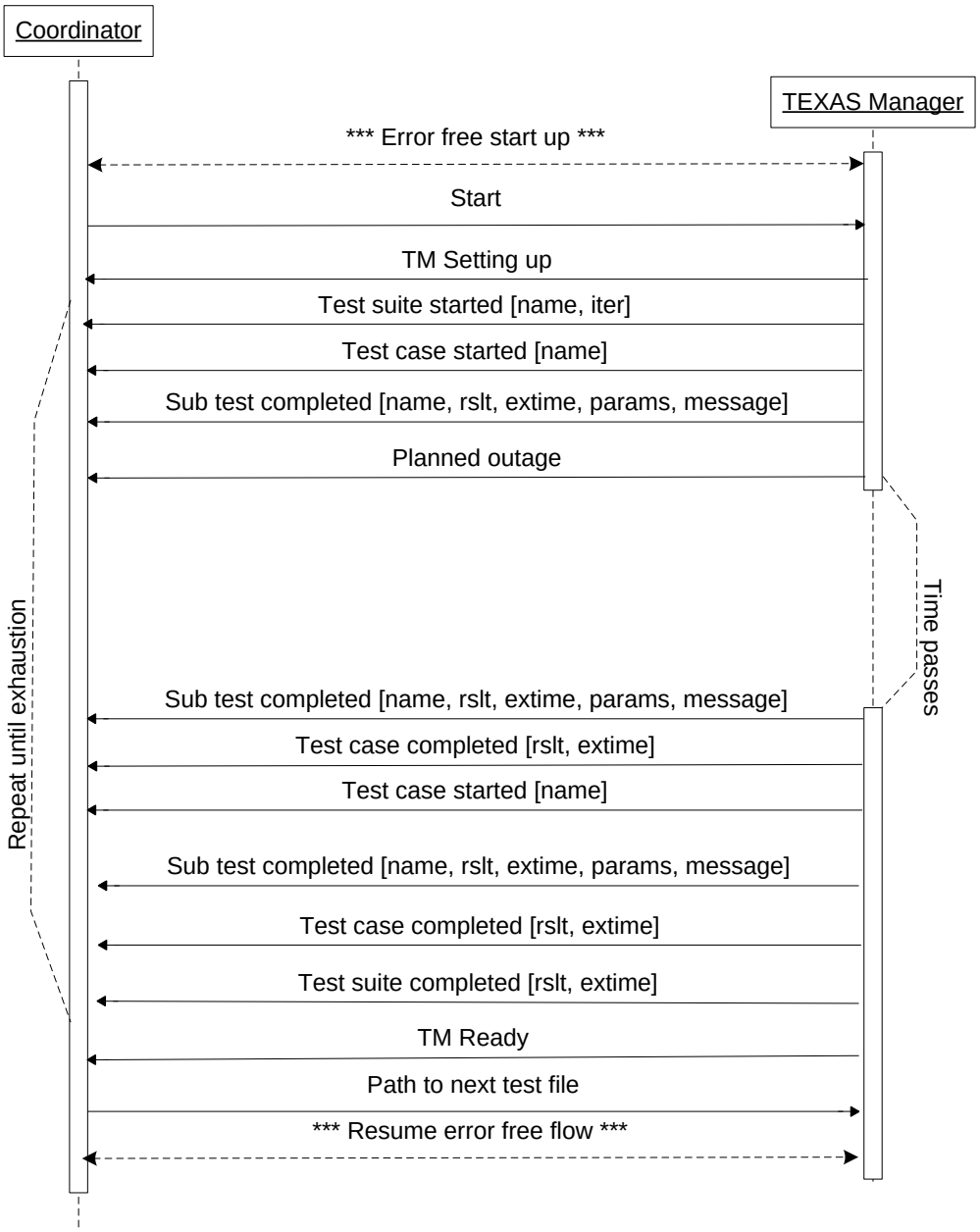
Startup from coordinator crash



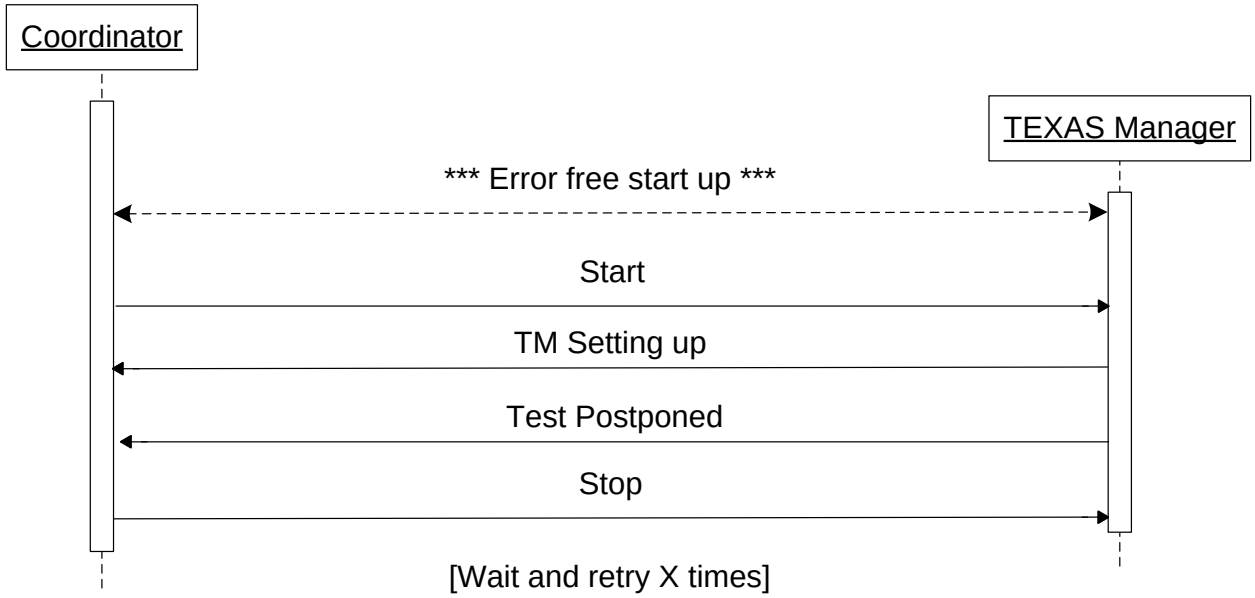
Startup from coordinator + manager crash (or computer restart)



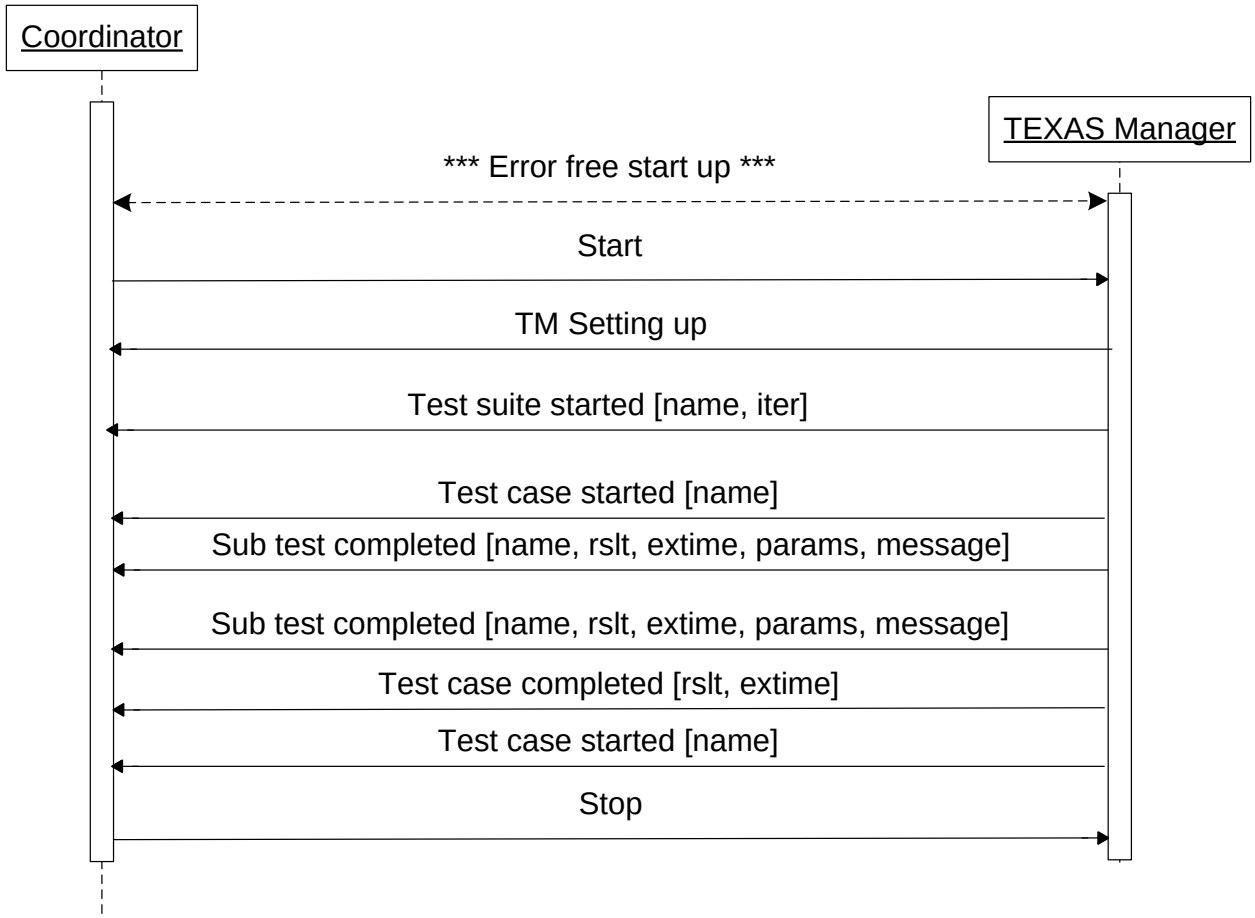
A node has a planned outage



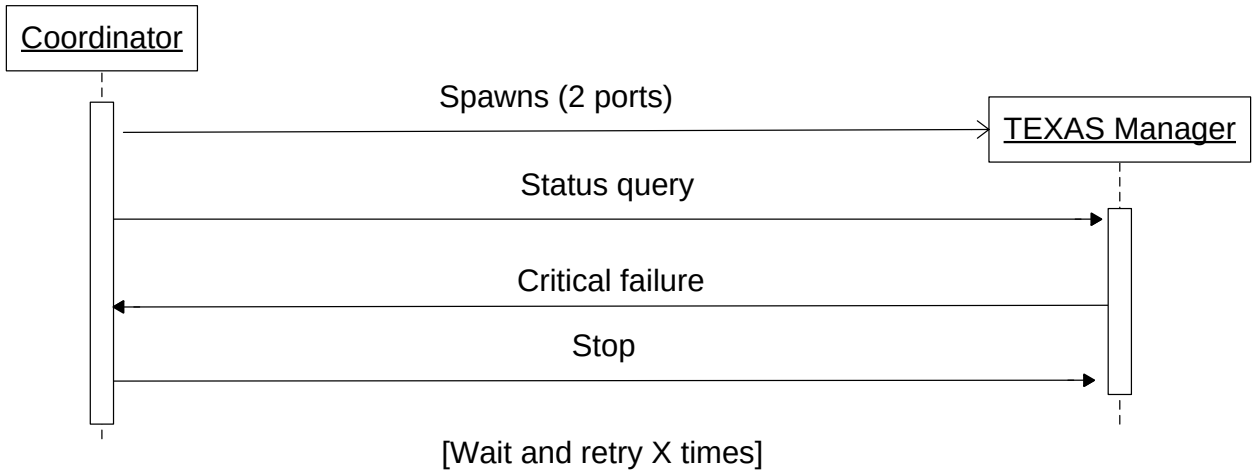
A node is busy



A user cancels a test



A node is unreachable



Appendix G. TEATIME User Guide (Abbreviated)

The following pages contain a draft of the user guide written for the TIME web interface.

TIME USER GUIDE

TABLE OF CONTENTS

1 Introduction

1.1 The TIME system

1.1.1 What is the TIME system?

The TIME system enables organizations to manage a set of assets, allowing users to access information on, book, add comments to, and request service for assets.

TIME also contains an administrative part that allows administrator to define asset groups (i.e. “computer”), asset types (i.e. a specific computer model) and assets (a physical computer) as well as other related information.

1.1.2 When was the system developed?

The TEATIME system, and thus the TIME part, was originally developed during the first half of 2009, as part of a Master’s Thesis project.

1.1.3 Why “TIME”?

While it would be fun to be able to say “Because you will save TIME when you use it” or something similar, the truth is closer to “It sounded good”.

2 About the guide

2.1 Disposition of the guide

Chapters 3-6 concerns typical usage of the system:
Chapter 3 describes the basics of the system
Chapter 4 discusses searching for and displaying information about assets
Chapter 5 discusses booking, checking out and checking in assets
Chapter 6 discusses scheduling tests to be executed automatically

Chapter 7 describes the possibilities to monitor the system

Chapter 8 describes the database design

Chapter 9 concerns the administration of the system.

Chapter 10 describes the configuration file and how to edit the values in it.

Chapter 11 deals with troubleshooting.

2.2 Thesaurus

Name	Meaning
Asset	A physical item, such as a computer or a cable
Asset type	The model of a physical item, such as a specific laptop model.
Asset group (asset type group)	The group to which an asset type belongs, such as “computers” or “cables”
User	Someone who is registered in the system
Guest	Someone who is NOT registered in the system
User group	A group to which several users belong, such as “Support” or “Development”
Category	A group to which assets belong, which ensures only user groups with access to said category can access the assets of said category.
Marking	A short identifier string which is unique for every asset.
SUT	Short form of system under test, which is another name of an asset in the testbed

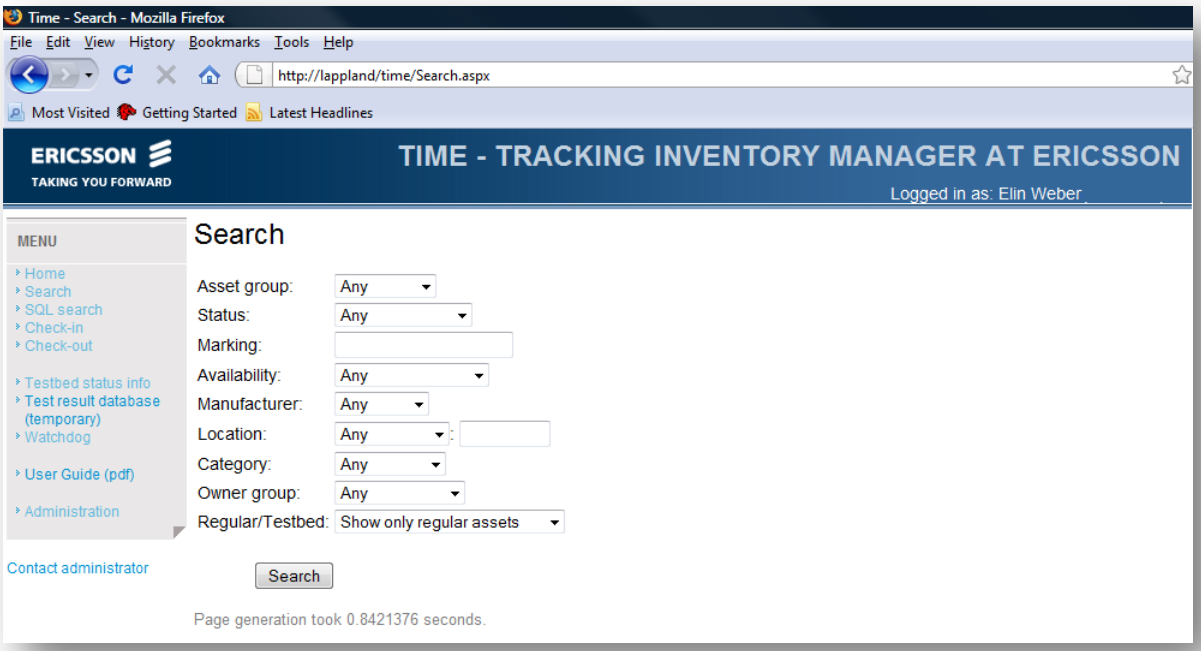
3 General information

3.1 The basics

To use the system you need to be a registered user, something that can only be done by an administrator. When the administrator registers you s/he will also assign you to a user group, which in turn gives you access to a set of assets.

3.2 Site layout

Most of the pages in the system uses the same “master” page, meaning that they have the same layout. An example of this layout can be seen below.



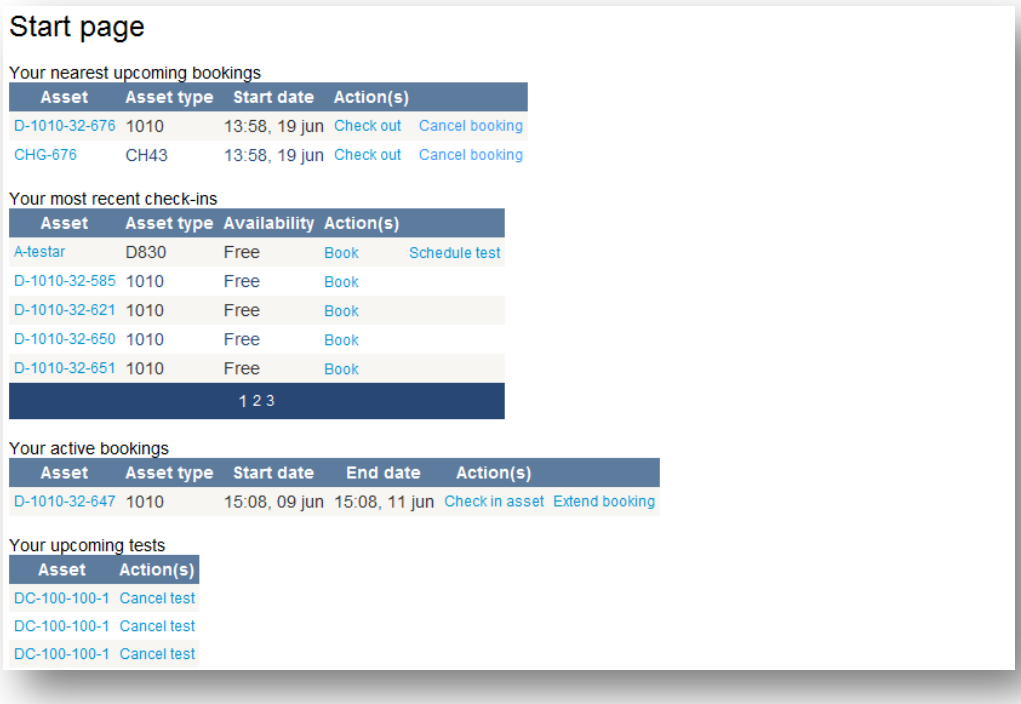
The header contains the Ericsson logo, the system name and the name and signum of the user currently logged in.

The left part of the page contains the menu which is used for navigation.

The middle/right part displays the content of the page currently visited.

3.3 Start page – User homepage

The first page a user will be presented with is the **Start page**. This page contains information on recent and upcoming activities for the user watching the page, such as upcoming and recent bookings and schedules tests. For each activity a set of related links are shown.



There are four lists available on this page, upcoming bookings, recent check-ins, active bookings and upcoming tests. All lists show asset information, such as marking and asset type.

For *upcoming bookings* the actions available are: **Check out**, which takes the user to the check-out page with the asset chosen as target, and **Cancel booking**, which, unsurprisingly, cancels the booking.

For *recent check-ins*, the actions shown are **Book asset**, which takes the user to the booking page with the asset chosen as target, and **Schedule test**, which takes the user to the test scheduling page with the asset chosen as target. The column *availability* shows whether the asset is currently free or in a booking.

For *active bookings* the available actions are **Check in,** which takes the user to the check-in page with the asset chosen as target, and **Extend booking**, which takes the user to the booking page in “extend booking” mode.

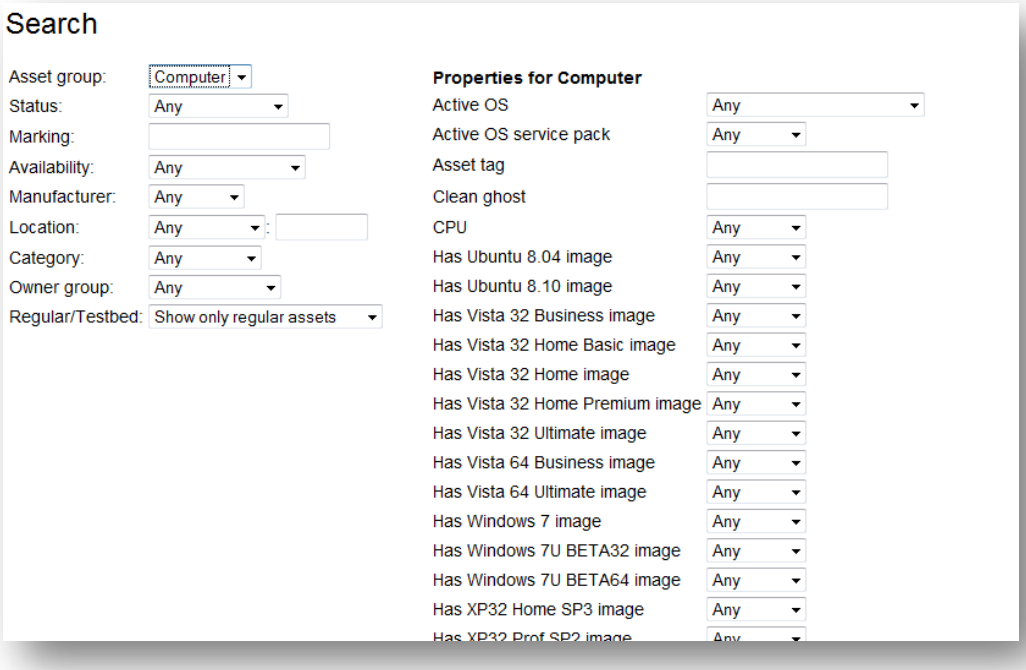
For *upcoming tests* the only available action is **Cancel test**, which, again unsurprisingly, cancels the test.

4 Searching for and displaying asset information

The system provides to ways of searching the data.
For assisted searching of assets the **Search** page is recommended. For more complex searches, or searches of other information, use the **SQL Search** page. The easiest way to display information for an asset is to use the **Asset details** page.

4.1 Search

The search page, accessible from the menu, offers several ways of searching for assets. For general searches it is possible to search by marking, manufacturer and location, amongst others. If an asset group is selected, all the properties for that asset group are shown, allowing more specific searches.



4.1.1 The basic search fields, and what they are used for

Asset group allows you to limit the search result to a specific group. It also fetches the properties for the selected group.

The **Status** list filters the result by status (i.e. “OK”, “Missing” and “Needs service”, amongst others)

Marking can be used to search for a specific asset, but also all assets with a marking starting with a certain string. The marking field is *auto-completed*, meaning that if you start writing a marking and then pause, the system will fetch ten markings that start with the string, and allow you to select one of them.

Availability allows you to limit the results to assets that are currently free or currently booked.

The **Manufacturer** list filters the result to only show assets from the selected manufacturer.

The composite **Location** field can be used to filter the results based on their location. The value selected from the list will be concatenated with the value in the text field, and any asset with a location beginning with the concatenated result is accepted.

Under **Category** the categories you have access to are show, allowing you to filter the results base on category.

The **Owner group** list allows showing only asset owned by a certain group.

The **Regular/Testbed** selection enables filtering on assets on which automatic tests can be scheduled. Available values are “Only regular”, “Only testbed” and “Both”.

4.1.2 Searching on asset properties

When an **Asset group** is selected, the system fetches all properties that are defined for the selected asset group.
If the property is of the type that only allows predefined values to be selected, a drop down list will be created, allowing you to select a value. The pre-selected choice “Any”, means that these criteria will not be part of the search filter.
If the property type allows any value to be entered, a text box will be created. If the text box is left empty, it will not be part of the search filter.

If you need to do more complex searches, such as allowing several values of one property, you need to use the **SQL Search** page.

4.1.3 The results of the search

When you press the search button, the system will fetch the assets that match the filter/criteria you have selected. These assets will be displayed in the search result box.

The result list has seven columns, “Book”, “Marking”, “Type”, “Availability”, “Status”, “Manufacturer” and “Action(s)”.

The “Book” column contains checkboxes. Checking any number of these checkboxes and then clicking the “Book asset(s)” button below the search results takes you to the booking page with the selected assets as part of the booking.

The “Marking” column contains the marking of the asset. If you click the asset you are taken to the asset detail page for that asset.

The “Type” column contains the asset type of the asset.

The “Availability” column contains information on the availability of the asset. Possible values are:

- “Free”, meaning that the asset is not in a booking and does not have any future bookings
- “Free until xx-yy”, meaning that the asset is not currently in a booking but has future bookings.
- “Booked”, meaning that the asset is currently in a booking.
- “Late since xx-yy”, meaning that someone has booked the asset but has not returned it, even though the booking has ended.

The “Status” column contains information on the status of the assets. The common value is “OK”, meaning there are no known errors for this asset. Other possible values are “Needs service”, meaning that someone has reported an error for the asset that has not yet been fixed; “Missing”, meaning the asset has been reported as missing by a user.

The “Manufacturer column contains the name of the manufacturer.

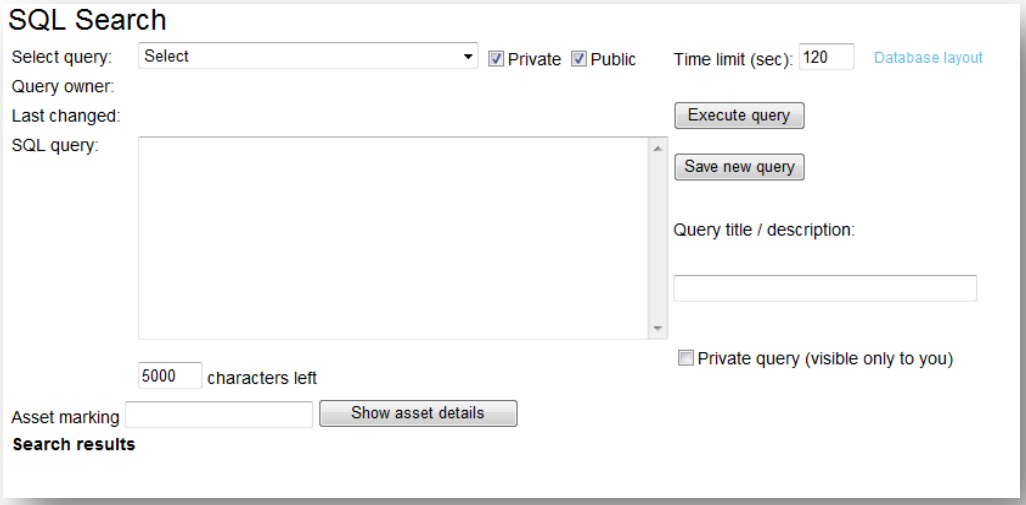
The “Action(s)”column contains a link to the booking page for assets that can be booked, and a link to the test scheduling page for assets on which tests can be scheduled.

4.1.4 Exporting the results to a file

The visible results of the search can be exported to a plain text file with semicolon field separation by clicking the **Export results to file** link at the bottom of the page.

4.2 SQL Search

The SQL Search page allows for more fine-grained searching of the data in the database. Any valid SQL select query can be executed, as long as it does not affect the data in the database in any way.



4.2.1 The basic search fields, and what they are used for

The drop-down list next to **Select query** makes it possible for you to use an existing query that is stored in the database. By selecting a query from the drop down list, the stored information (**Query owner, Last changed, SQL query**) about the query will show up on the page.

Use the text field next to **SQL query** to write your own query from scratch or base it on a selected pre-defined query.

The check-boxes **Private** and **Public** are used to filter the queries in the drop down list. The saved queries are either public or private. Public queries can be used by any user in the system. Private queries can only be used by the user that created it.

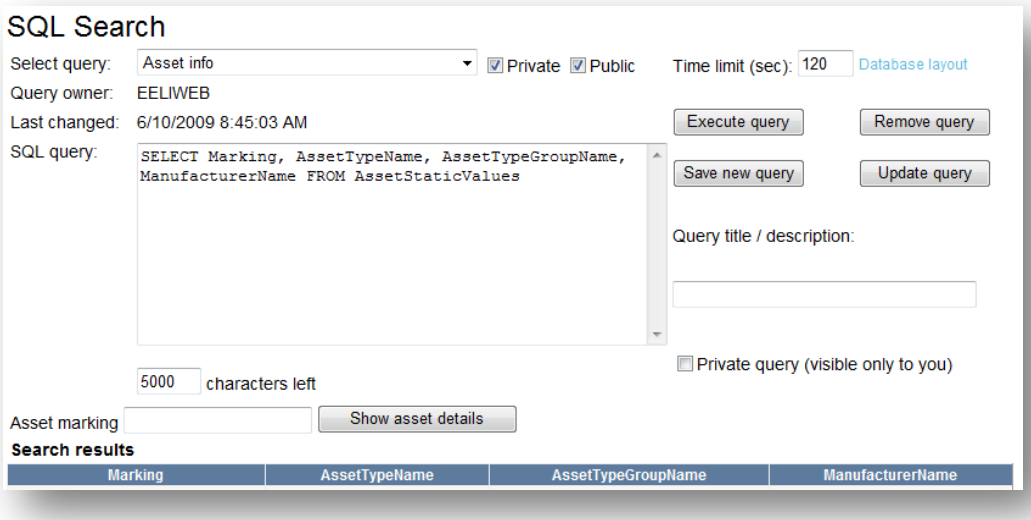
The text-box next to **Time limit (sec)** is used to define a time limit for the SQL query execution. If the time limit is set to zero seconds, the system will use the timeout set by the database. The default time limit is 120 sec.

As a help to write proper SQL queries the **Database Layout** link will open a new window, showing all the tables in the database. The table names, column names and the data type for each column will be displayed, see chapter 8.1.

Click on the **Execute query** button to execute the query. The result will be displayed in the table below **Search results** on the same page.

Before you can save a new SQL search you need to write a query in the **SQL query** text field as well as give it a name in the **Query title / description** text field. By checking the check-box **Private query (visible only to you)** you can make the query private. If you do not check the check-box the query will be public. Press the **Save new query** button to save the SQL query in the database.

If you have selected a query that you own, two other buttons will be visible on the page; **Remove query** and **Update query**. If you press the first button the query will be removed from the database. If you press the second button you will put the system in query update mode. As you click on the **Update query** button the text on the button will change to “Cancel update” and the text “Save new query” will change to “Save query”. Press the **Cancel update** if you want to cancel the update mode or press the **Save query** if you want to save the updates you have made. If the action you choose is successful the page will go back to normal.



If you want to know more information about a certain asset, you can fill in the marking in the text field above the search results and press the button **Show asset details.** You will then be sent to the **Asset details** page.

4.2.2 The results of the search

When you press the execute button, the system will fetch the result of the query. The data will be displayed in the search result box. The column names depend on the query.

The result can be sorted by clicking at the header text of each column.

4.2.3 Exporting the results to a file

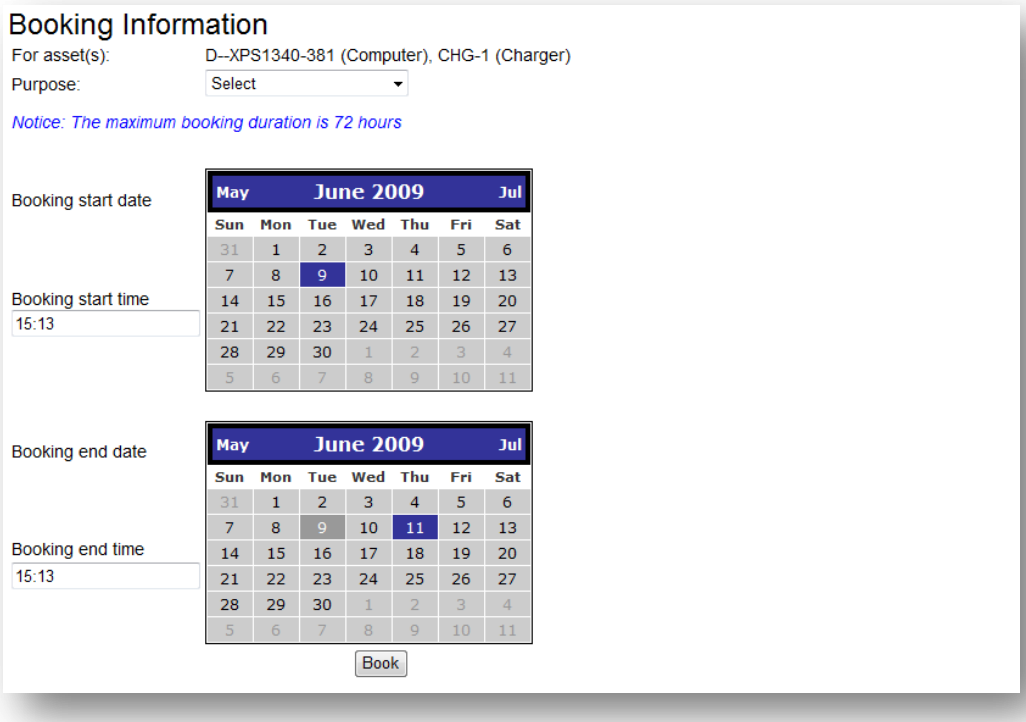
The visible results of the search can be exported to a plain text file with semicolon field separation by clicking the **Export results to file** link at the bottom of the page.

5 Booking, checking out and checking in assets

The system provides several ways to place a booking containing one or more assets by accessing the **Booking information** page. To be able to check-out and in assets in the system, the two pages; **Check out** and **Check in** are used.

5.1 Booking information page

The **Booking information** page can be accessed in many different ways. The most common way is to search for assets and use the links on the **Search** page to place a booking containing one or more assets. If you have checked in asset(s) recently, they will show up on **Start page** under **Your most recent check-ins** with a link to the booking page. There is also a possibility to book an asset via the **Asset details** page. The booking page will also be accessed if you want to extend any of your bookings (that does not contain a testbed asset), by using the link **Extend booking** in the table for active bookings on **Start page**. Finally, if you try to check out an asset that you have no booking for, the check-out page will send you to the booking page.



5.1.1 The booking information fields, and what they are used for

At the top of the page, next to the **For asset(s)** label the asset(s) that you are trying to book will be listed with their marking and information about which asset group they belong to.

Do not be surprised if there are more assets in this list than you expected to book, because if any of the assets you planned to book is part of an asset family, the whole family will be displayed at the top of the page. An example of an asset family is a computer that has a charger as part of the same package.

At the top of the page the user will be notified about the maximum booking duration. If the booking is longer than specified time, an administrator will be notified and can cancel the booking at any time. The user will get a warning about this fact if the booking is placed successfully.

A testbed asset must be booked by itself. Therefore, if any of the assets is a part of the testbed and bookable a warning will be displayed to the user to modify the booking: “Warning. You are going to book an asset that is part of the testbed”.

To be able to place a booking you need to first provide the system with some information. Use the drop-down list next to **Purposes:** to either select a pre-defined purpose or select “Other”. If you select “Other” a free text field, that has to be filled in, will show up on the right side of the drop-down list.

Use the calendar at **Booking start date** to select a start date for the booking. By default the calendar is set to today’s date. Fill in the start time in the text-box next to **Booking start time**. The start time is set to now by default. The **Booking end date** works the same way.

5.1.2 To place a booking

You place the booking by pressing the **Book** button. When the button is pressed the system will check for collisions with other bookings, stored in the database. The system will notify the user about the result. The different outcomes of the booking action are as follows:

- If the booking succeeds and the user is allowed to check out the asset(s), a message and a link button to the **Check out** page will be shown at the top of the page.

- If you have entered the page through the **Check out** page and the selected asset is part of an asset family, the booking will include all family members. If the booking is placed successfully the selected asset from the **Check out** page is already checked out, and the user will be sent to the **Check out** page automatically, to finish the checkout of the other assets in the family.

- If a testbed asset has been booked successfully a link to the **Schedule test** page will be shown.

- If a collision is found, the system will show a table holding information about the current and future bookings within the next 24 hours that includes any of the assets you are trying to book. By looking at the table you can find a free time slot. After the start and/or end time of the booking has been changed, press the **Book** button again and wait for the system response.

5.1.3 To extend a booking by changing the end date and time

If you enter the page by using the **Extend booking** link it will look a bit different. Since you are not allowed to change the start date and time of the booking, the calendar and the text field will be disabled. Also, it will not be possible to change the purpose of the booking. The calendar and text field for the end date and time will be editable.

The book button will be replaced by an **Update** button. If you press the button, the system will try to fulfill your request about extending the booking and notify you about the outcome.

5.2 Checking out assets

The **Check out** page can be accessed through a link in the menu or by using the link under **Your nearest upcoming bookings** table (if there are any bookings shown) on **Start page**.

If you enter the page by using a **Check out** on the start page, the related booking id will be part of the URL.

If you enter the page by using the link in the menu, there are several ways of checking out an asset. You can either use the bar code reader at the check in-/out station and scan the bar code label on the asset or use the links under "**Your booked assets that you can check out now**" or use the keyboard to type in the marking or the bar code in the text field **Marking or bar code:** at the top of the check out page.

As the title of the table says, it will only show the assets that can be checked out at the moment. Users are typically allowed to check out an asset some time before the booking starts, if the asset is available.

If the keyboard is used, the auto-complete function will help you to find the marking quickly.

The check out process starts as soon as you enter the page if the booking id is part of the URL or when you press the **Check out** button. Depending on current/future bookings, number of assets in a booking, the status of the asset etc, the system will notify the user about the outcome:

- If the booking ID in the URL is a number but not an id of an existing booking or a booking that has expired, the following message will be shown: ”Error. No booking was found.”

- If the single asset is available and booked by you and the booking ID is correct, it will be checked-out to you immediately. The following message will be displayed: “The asset [marking] has been checked out to you”.

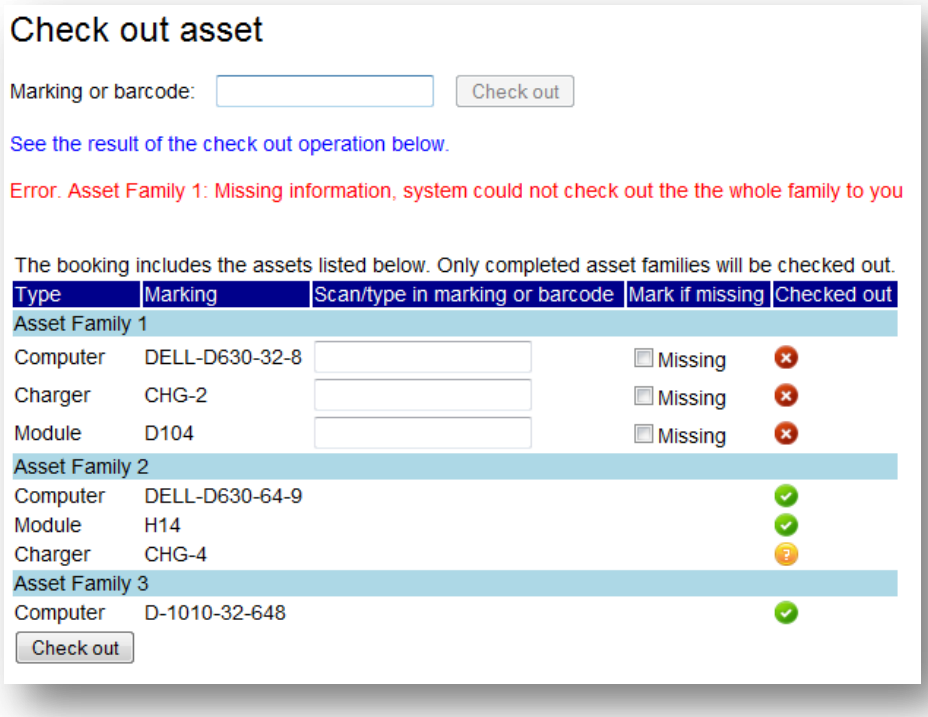
- If the single asset is booked by another user, no asset will be checked out and you will get the message: “Error. You can not check out the asset because the current booking belongs to [signum of user]”

- If there is another booking before yours, no assets will be checked out and you will get the message: “The asset(s) belong to another booking that starts before the one you're trying to check out.”

- If there is a late booking that includes any of the assets in your booking, no assets will be checked out and you will get the message: “One of the assets belongs to a late booking that belongs to [signum of user]”.

- If the booking belongs to you but it is too early to check out the asset(s) (according to the value in the settings file), no assets will be checked out and a warning message will be shown: “Warning: The booked asset(s) can not be checked out more than [check out time (min) from settings file/ 60] hours before your booking starts”.

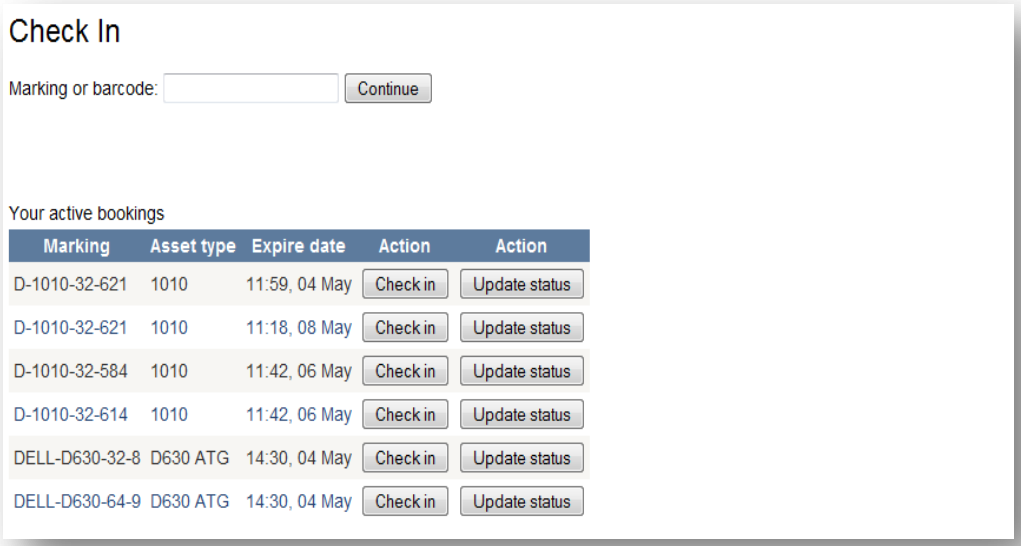
- If the asset is part of one of your bookings containing several assets (including parts of assets), all these assets will be displayed on the page. You then need to go through the list where you can either scan or fill in marking/bar code in the text fields or check the check-box saying “Missing” or just leave a row empty to check it about later. Only completed asset families will be checked out (if the check out process succeeds). Press the second **Check out** button and wait for the system response. A message will be shown above the table and the result will be explained with symbols for each asset in the table:
    - Red image means not checked out
    - Green image means checked out
    - Yellow image means the asset was set as missing



- If the asset is not part of any current or future bookings the user will be send to the **Booking information** page. When the required information is filled in and a booking is placed successfully the asset is checked out to the user. If the checked-out asset is part of an asset family, the user will be send back to the **Check out** page to complete the check out process for the whole family.

5.3 Checking in assets

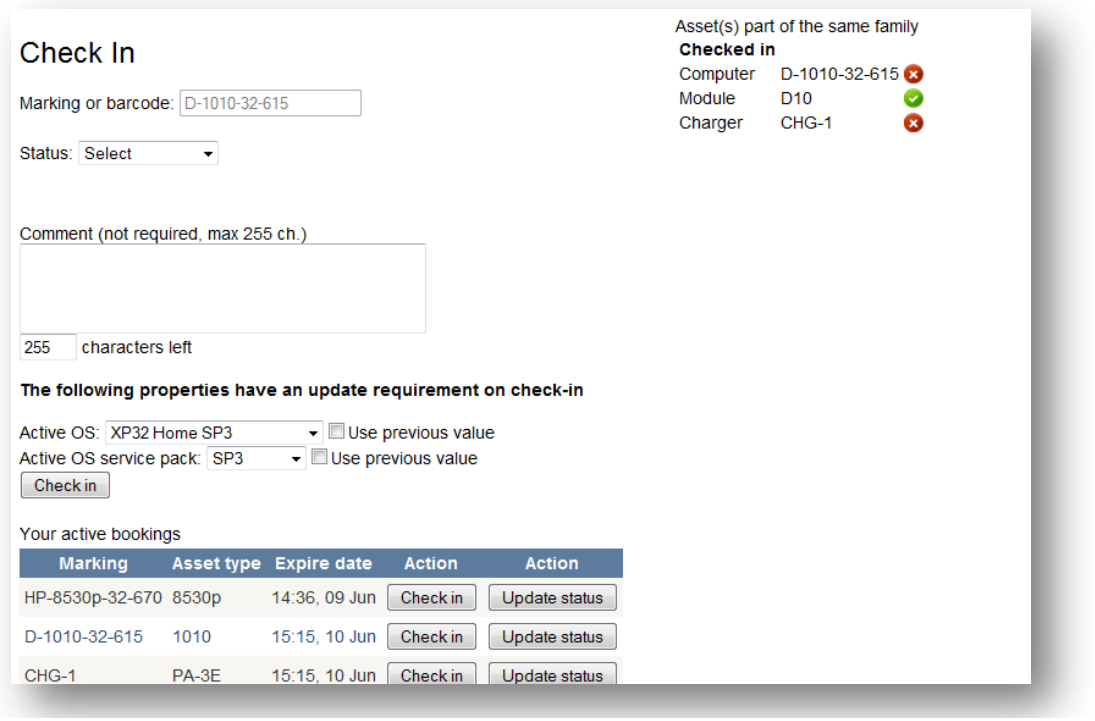
The **Check in** page can be accessed through a link in the menu or by using the link in **Your active bookings** table (if you have any) on **Start page**.



If you enter the page by using the link in the menu, there is several ways of checking in an asset. You can either using the bar code reader at the check in-/out station and scan the bar code label on the asset or by using the links in the "**My active bookings**" table or by using the keyboard to type in the marking or the bar code in the text field **Marking or bar code:** at the top of the check in page.

Press the **Continue** button to fill the page with necessary information for the check in process.

If you enter the page by using one of the Check in asset links on the start page, the Continue button is automatically clicked and the marking and some stored information about the selected asset will be filled into the editable fields.



To be able to continue the check in process you have to select a status from the **Status** list. If you select the status “Needs service” or “From service” a list of all available service types will be visible and you have to select one of them from the **Service type** list.

It is optional to fill in a comment in the **Comment** field. The field has a limit of 255 characters.

If the asset you are going to check in has any properties that are required for to fill in, they will be listed below the comment field. Go through each property and either select a value from the list or check the **Use previous value** check-box.

Press the **Check in** button to try to check in the asset. If the process succeeds the booking of the asset will have the status set to “Completed” in the database.

Depending on the outcome of the check in process, a message will be shown at the top of the page and the result will be shown as a set of symbols in the upper right corner saying **Checked in**. Each asset, that is part of the same family as the asset you are trying to check out will be listed as well. There are two different symbols available:

- Red image means not checked out
- Green image means checked out

If the asset is part of a family you need to check in all the other assets one by one. A message saying “Remember: If this asset is part of a family, you need to check them all in one by one.” will be visible.

If the status of the checked-in asset affects the following bookings, for instance if the status is set as missing, an email will be sent out to the administrative contact of the asset.

5.3.1.1 Update status during an active booking

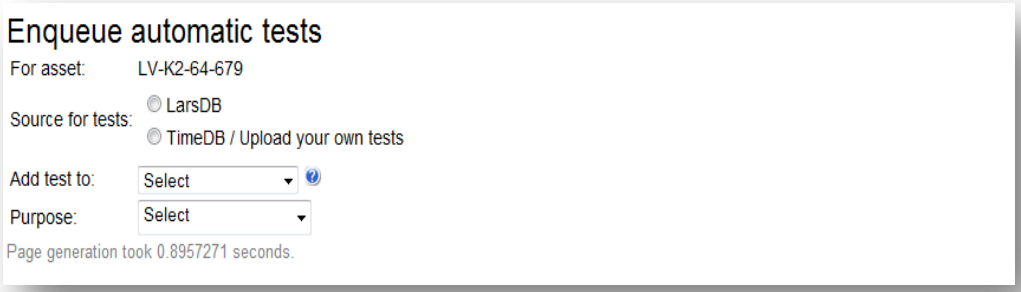
It is possible to change the status of an asset when it is checked out to you. Use the **Update status** link on the start page to access the Check-in page with the selected asset's marking filled into the **Marking or bar code** field at the top of the page. The user can only change the status of the asset, while the rest of the page is disabled. The Check-in button is replaced with the **Update status** button to save the new asset status.

6 Scheduling tests to be executed automatically

The web interface can be used to schedule tests that will be stored in the database and executed automatically by the TEA-Coordinator. The **AutoTest** page is used for this.

6.1 Schedule tests

The **AutoTest** page can be accessed for testbed assets only in the same way as the **Booking information** page can be accessed by regular assets. This means either by using the **Schedule test** link on the **Search** page, **Start page** or the **Asset details** page.



6.1.1 The schedule test top information fields, and what they are used for

Automatic tests can only be scheduled on testbed assets. If the asset does not fulfill the requirement a warning message will be shown: “Error. Asset is not an active testbed asset, go back and select another asset please”.

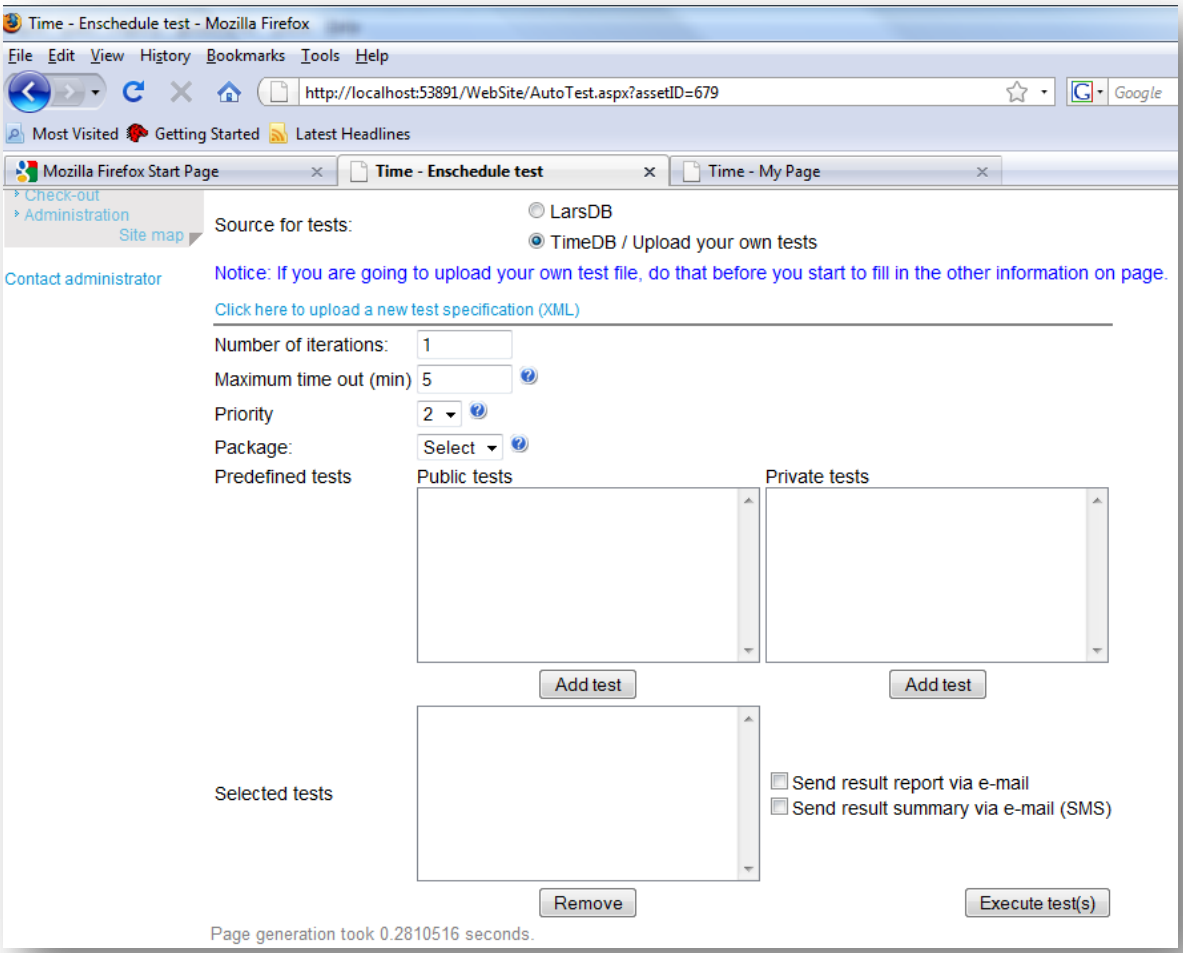
At the top of the page, next to **For asset** label the marking of the asset that was selected on the previous page will be displayed.

Make the choice of **Source of tests** by selecting the radio button for either using **LarsDB** (a.k.a. the Planning and Result database) or using **TimeDB / Upload your own tests.** When a radio button is selected the page will reload and show different fields based on the choice of database you made.

Common information for both sources is the dropdown lists for selecting which booking to add the test to and for what purpose. The **Add to test** list contains your current bookings and the alternative “Normal test queue”. The **Purpose** list contains a set of pre-defined purposes and the alternative “Other”. If you select “Other” a free text field will be visible to the right of the dropdown list.

6.1.2 Selected to use TimeDB / Upload your own tests

Some information is required before a test can be placed in the database. As the notice on the page says if you want to upload your own test file, this has to be done before the rest of the information is filled in. See section 6.1.2.1 for detailed instructions how to upload a test file.



The number of iterations of the test suite is shown in the text field next to the **Number of iterations** label. The default is set to one iteration.

The text field next to the **Maximum time out (min)** label specifies the number of minutes to wait for the SUT to become available after a planned disconnect. If this time is exceeded the test is aborted. The default time (in minutes) is specified in the settings file.

Use the **Priority** list to set the priority of the test suite. The priority indicates if the test has high (1), normal (2) or low (3) priority when it comes to test execution. The default is priority 2.

Use the **Package** list to select what package to use for the test. The package specifies what files and driver will be sent to the SUT for the test. If none of the packages in the list match your requirements you need to upload them to the TEA-Coordinator server and wait for the TEA-Coordinator to add them to the list

To add a test to your test specification, select a test from either the list box called **Public tests** or the one called **Private tests** and then press the **Add test** button below**.** A public test is visible for all users while a private test is visible only for the owner who uploaded it.

As you have selected a test and pressed the add button, the test will show up in the list box called **Selected tests.** If you regret that you have added a test to this list box, you can easily remove it by select it and then press the **Remove** button.

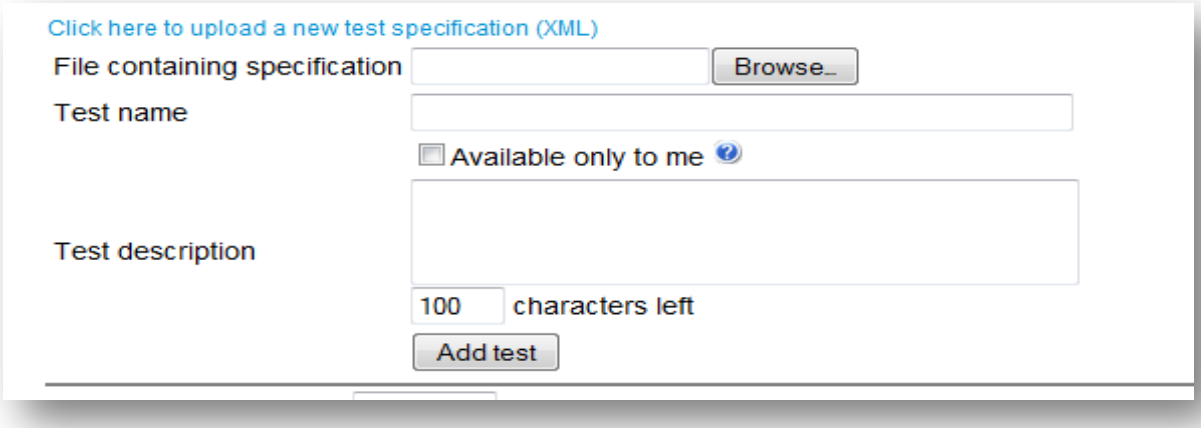
You can choose how you want to be notified if anything goes wrong and when the test execution is done. Use the checkboxes to specify if you want to be notified by e-mail and/or SMS.

When you are happy with the test specification, press the **Execute test(s)** button to store the test specification and test file(s) (if there is any) in the database.

The system will notify the user about the outcome and if a test is placed in queue successfully: “The test item will be executed on [marking]”.

6.1.2.1 Upload a test file

To upload a new test specification, press the link Click here to upload a new test specification (XML)**.** Then a new section will be visible on the page.



Press the **Browse** button to browse the file to be uploaded. Then type in a name of the test in the **Test name** text field. Check the checkbox called **Available only to me** if the test file should be private. Then type in a test description (max 100 characters) in the **Test description** text field**.** Finally press the **Add test** button to upload the test. If the upload is successful, the test will show up in either the list box for public or for private tests depending on the choice of making it private or not.

7 Possibilities to monitor the system

There are three different ways to monitor some of the system performance:

- On the **Test result database** page you can see the tables in the temporary result database. Use the **Test result database** link in the menu to access this page.

- On the **Watchdog result** page you can follow the actions that the Watchdog has taken. Use the **Watchdog** link in the menu to access this page.

- The **Testbed status info** page shows information of the assets in the testbed. Use the **Testbed status info** link in the menu to access this page.

- On the **Result viewer** page, you can monitor the result of an automatic test execution. The page is updated in real time. You can not access this page through the menu because it requires the test suite set ID as a parameter in the URL to be able to show any information. Use the page showing the test result database to find the test suite set ID for the test you are interested to see. Then use the following URL and use the test suite set ID you have found:

  ~/ResultViewer.aspx?TestSuiteSetID=[number]

  Note: The Result Viewer works for tests from the TimeDB ONLY.

7.1 Test result database page

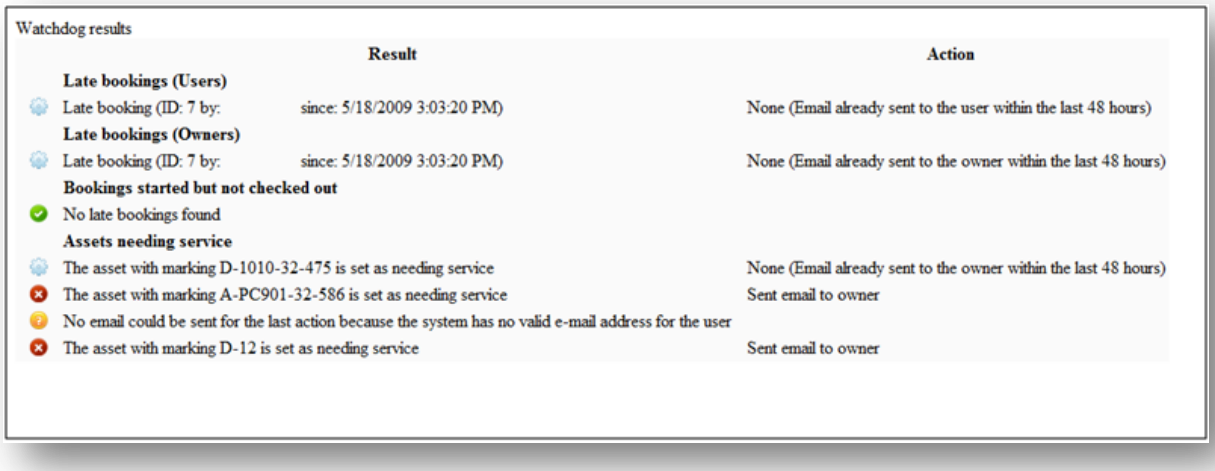
The **Test result database** page is a way to look at the content of the temporary result database. This database stores result from test execution of TimeDB tests. The page shows all the tables in the temporary result database.

7.2 Watchdog result page

The Watchdog shows its activities on the **Watchdog result** page. The page will open in a new window if you click on the **Watchdog** link in the menu.

The page is divided into four sections; **Late bookings (Users), Late bookings (Owners), Bookings started but not checked out, Assets needing service**, which are described below**.** Each row in each section describes an activity, the result and an action. The symbol to the left shows the status of the activity The following symbols are available:

- Red image means fail/error/not ok
- Green image means pass/ok
- Yellow image means warning
- Light blue wheel image means ongoing work


Watchdog results

Result		Action
Late bookings (Users)		
Late booking (ID: 7 by:	since: 5/18/2009 3:03:20 PM)	None (Email already sent to the user within the last 48 hours)
Late bookings (Owners)		
Late booking (ID: 7 by:	since: 5/18/2009 3:03:20 PM)	None (Email already sent to the owner within the last 48 hours)
Bookings started but not checked out		
No late bookings found		
Assets needing service		
The asset with marking D-1010-32-475 is set as needing service		None (Email already sent to the owner within the last 48 hours)
The asset with marking A-PC901-32-586 is set as needing service		Sent email to owner
No email could be sent for the last action because the system has no valid e-mail address for the user		
The asset with marking D-12 is set as needing service		Sent email to owner

7.2.1 Late bookings (Users)

The first section, late bookings (Users), shows actions taken to remind users that they need to return an asset because the booking has expired.

If no late bookings were found the a green symbol will be visible and a message saying “No late bookings found”.

7.2.2 Late bookings (Owners)

The second section for late bookings shows actions taken to notify owners that a user has not returned an asset when a booking has expired.

If no late bookings were found the a green symbol will be visible and a message saying “No late bookings found”.

7.2.3 Bookings started but not checked out

The third section, regarding bookings that have started but has assets that have not been checked out, shows these bookings. If one or more assets in a booking are not checked out within a pre-defined time in the settings file, will be canceled automatically by the system.

If no late bookings are found a green symbol will be displayed and a message saying “No late bookings found”.

7.2.4 Assets needing service

The last section lists the assets that need service. These assets have “Needs service” as their status.

If no assets need service, a green symbol is shown and a message saying “No assets needing service were found”.

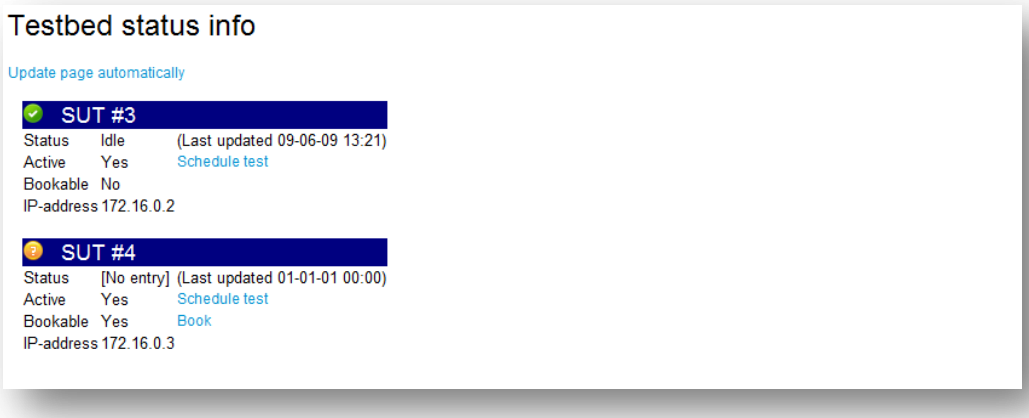
7.3 Testbed status info page

The **Testbed status info** page lets the user monitor the status of the assets in the testbed.

Each asset in the testbed, called a SUT, is represented by a table that shows information about:

- The asset's status and the time of the latest update
- Whether the asset is active or not. If the asset is active, a link to the AutoTest page will be visible. Click on the **Schedule test** link to go to the AutoTest page and schedule a test to run on this asset.
- Whether the asset is bookable or not. If the asset is bookable, a link to the booking information page will be visible. Click on the **Book** link to start to place a booking of this asset on the Booking information page.
- The IP address of the asset

Click on the **Update automatically** link to set the page in a mode where it updates automatically.



7.4 Result Viewer page

The result viewer page shows the test execution result for a specific test suite set. The ID of the set of test suites has to be specified in the URL. The page updates frequently, which means that you can follow your test progress in real time during the execution.

Note: The result viewer is only available for tests from the TimeDB ONLY.

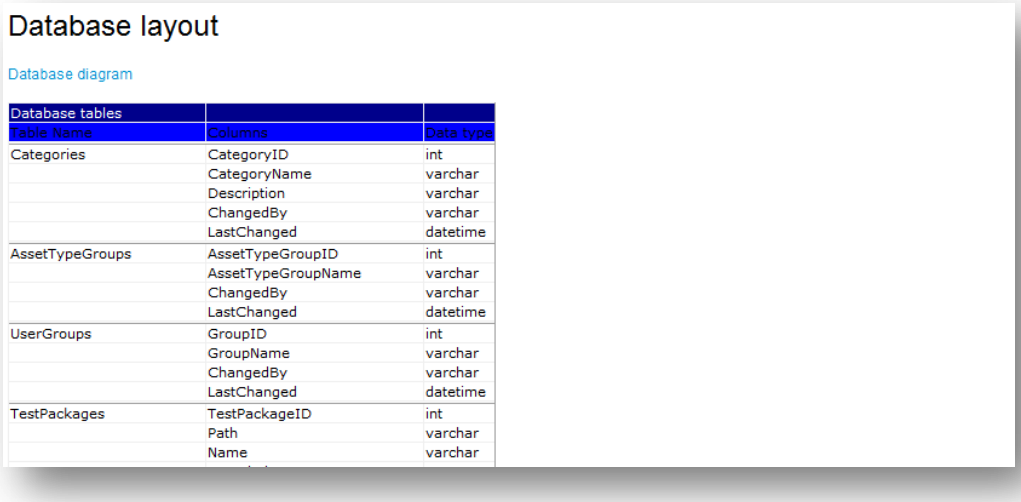
8 Database pages

To give the user necessary knowledge about the database, the layout and diagram can be accessed through the web interface. The pages **Database layout** and **Database diagram** hold information about the database design.

8.1 Database layout page

The **Database layout** page is accessed by using the link on the **SQL Search** page. The database layout is shown as a big table with three columns. The upper part of the big table contains information about all the database tables. The information is shown in three columns; “Table name”, “Column name” and “Data type”. The lower part of the big table contains information about the views that are stored in the database. There are three columns here as well; “View name”, “Column name” and “Data type”.

By using the link **Database diagram** at the top of the page, a new window will open and showing a big diagram of all tables in the database.


Database layout

Database diagram

Database tables		
Table Name	Columns	Data type
Categories	CategoryID	int
	CategoryName	varchar
	Description	varchar
	ChangedBy	varchar
	LastChanged	datetime
AssetTypeGroups	AssetTypeGroupID	int
	AssetTypeGroupName	varchar
	ChangedBy	varchar
	LastChanged	datetime
UserGroups	GroupID	int
	GroupName	varchar
	ChangedBy	varchar
	LastChanged	datetime
TestPackages	TestPackageID	int
	Path	varchar
	Name	varchar

8.2 Database diagram page

The **Database diagram** page is accessed by using the link on the **Database layout** page. The diagram will help you to find out information i.e. relations between the tables, primary and foreign keys.

9 Administration parts

A user can be set to have administrative rights. When such a user is visiting the page, a link to the Admin pages will be visible in the menu to the left. On the admin pages the administrator can add/edit/delete records in the database.

9.1 Admin page - in general

First of all the administrator uses a dropdown list next to **Select a database table** to select which database table s/he wants to use. The drop down list will be visible on all admin pages.

Depending on the choice of table, the web interface will show a section with editable fields for adding a new record in the selected table. All existing database records in the selected table will be shown at the bottom of the page below the label **Existing database records**.

The database records are sorted by the ID in descending order by default. Press the column headers to sort the table in the way you want.

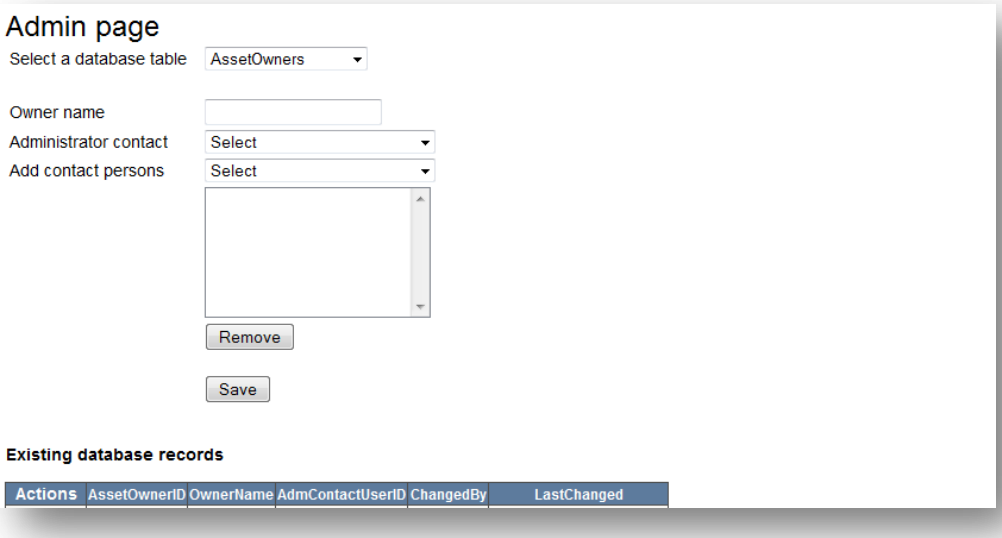
The leftmost column does always contain actions and is called “Action(s)”. The column contains an **Edit** link to edit an existing database record, and a **Delete** link to delete a record in the database. There are two exceptions; if the TestbedAssets table is selected, the delete link is missing and if the Assets table is selected, there is an additional link called **MakeCopy**.

If the **Delete** link is pressed, a confirmation pop-up will show to prompt the user for a “Yes” or “No” to answer the question: “Are you sure you want to delete the record?”.

If the **Edit** link is pressed, the information stored about the record will be displayed in the editable fields at the top of the page. The **Save** button will be replaced by two other buttons; **Update** and **Cancel**. If the user presses the first button, the system will try to update the information in the database. If the user presses the second button, the edit action will be canceled and the page will be reset. In either case, the system will notify the user about the outcome of the action.

9.2 Admin page – AssetOwners table selected

The AssetOwners table holds information about asset owners. The administrator can add, edit and delete an asset owner on this part of the Admin page.



9.2.1 Add a new asset owner

A name is required information for an asset owner. Fill in a name of the owner in the **Owner name** field.

An asset owner can have an administrative contact which is a user that will receive the e-mails from the system's Watchdog. Select a user in the dropdown list **Administrative contact**. The users are presented as [First name] [Last name] [(Signum)], eg. Sven Svensson (SIGNUM)

An asset owner can have a set of "owner" users. To add a user as one of the owners, use the **Add owners** list and select a user. The selected user will disappear from the list and show up in the list box. To remove a contact person, select it in the drop down list and press the **Remove** button. The selected user will then be removed from the list box and be visible in the dropdown list again.

To save the new asset owner, press the **Save** button. The system will verify and make sure no information is missing before it tries to save it in the database. The system will notify the user about the outcome. If the save action succeeds the editable fields will be cleared and the lists will be reset and the new asset owner will appear in the table. Otherwise the user will have a chance to correct the information and try to save the information again.

9.2.2 Edit an asset owner

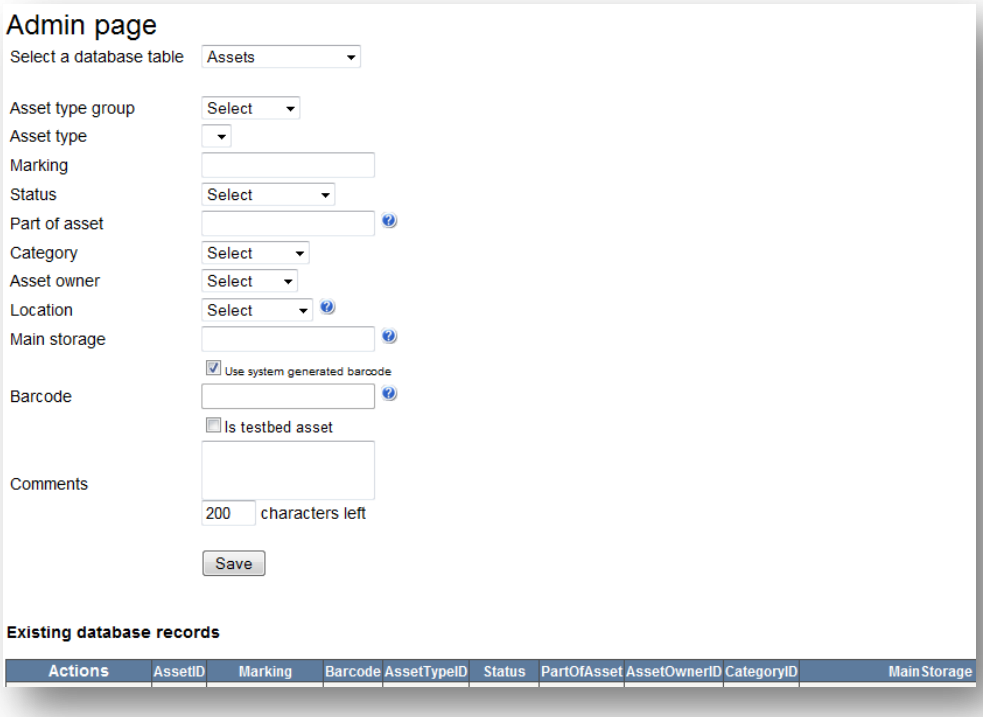
Press the **Edit** link and the stored information of the selected record will be filled into the editable fields. Edit the fields and press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.2.3 Delete an asset owner

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear from the **Existing database records** table.

9.3 Admin page – Assets table selected

The Assets table holds information about assets. The administrator can add, edit and delete an asset on this part of the Admin page.



9.3.1 Add a new asset

The new asset needs an asset group to belong to. Use the **Asset type group** list to select a suitable asset group. If you do not find any suitable asset group, go to the asset type group section of the Admin page and add a new one.

When an asset group is selected, the **Asset type** list will be filled with asset types that are related to the asset group. If there are no asset types related, an error message will be shown to the user, telling her/him that a new asset type need to be added and related to the asset type group. The user then needs to go to the asset types section of the Admin page.

If the asset group has a set of properties related to it e.g. a computer has an active operating system, CPU brand, several images etc, these properties will show up in a table containing editable fields to the right on the page.

All assets need a unique marking. Use the **Marking** field for that**.**

Define the current status of the asset by selecting a pre-defined status in the **Status** list.

If the asset is part of another existing asset, type in the marking of the parent asset in the **Part of asset** field. The field has an auto complete function that will help you to find the parent marking quickly. If the new asset is not part of a family, leave the field empty.

Use the **Category** list to select which category the new asset belongs to.

Define the owner of the asset by selecting an asset owner in the **Asset owner** list.

The new asset needs a home location. Use the **Location code** list to select a suitable location and type in the place where the asset is usually stored in the **Main location** field.

A unique barcode is required for each asset. The barcode can be printed on a label and put on the asset to be used in the check-out and -in processes. The user can choose if s/he wants the type in a barcode or leave it to the system to generate the barcode. Check the **System generated barcode** checkbox if you want to let the system take care of it.

An asset is either a regular asset or part of the testbed. Check the **Is testbed asset** checkbox to indicate that the new asset is in the testbed.

It is optional to add a comment in the **Comments** field. The comment is limited to max 200 characters. There is a character counter below the text field to help you to keep the comment length within the limit.

If the asset group you have chosen has a set of properties make sure you use the lists and the free text fields to fill in all information before you hit the save button.

Press the **Save** button to save the information about the new asset. The system will then verify and make sure no required information is missing before it tries to save it in the database. The system will notify the user about the outcome. If the save action succeeds the page will be reset and the new asset will appear in the **Existing database records** table, Otherwise the user will have a chance to correct the information and try to save the information again.

If the save operation succeeds a text field called **Marking** and a **Show asset details** button will be visible at the top of the page. The text field will contain the marking of the recently added asset. By clicking on the link the **Asset details** page will be shown.

9.3.2 Edit an asset

Press the **Edit** link and the stored information of the selected asset will be filled into the editable fields. You can then edit almost all information in the fields except of a few things.

- You are not allowed to change the asset type group
- You are not allowed to change the barcode

The comment field will always be empty when you enter the edit mode for a selected asset. If you type in anything in the comment field and then save it, it will be stored in a separate table for asset comments the keep track of the comment history for each asset.

Press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the page will be reset.

Shortcut: Instead of finding an asset to edit in the table, there is a shortcut to select an asset in edit mode. Use the following url:
~/Admin.aspx?assetID=[ID of the asset you want to edit]

9.3.3 Delete an asset

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear in the **Existing database records** table.

9.3.4 Make a copy of an existing asset

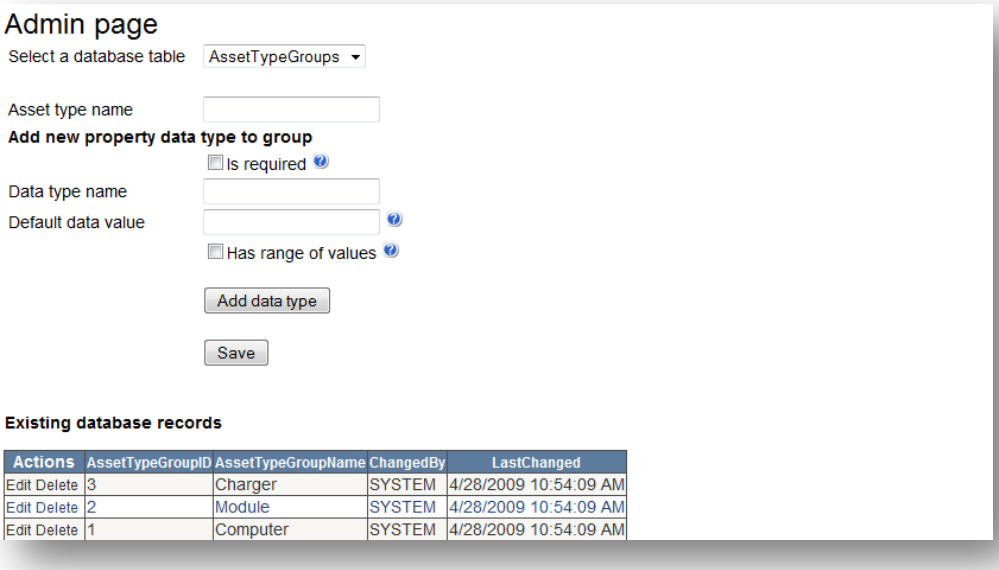
If you are going to add a set of assets that has very similar characters and properties you can create the first one from scratch and then make copies of it. By pressing the **MakeCopy** link the system will fill in all information (except of marking and barcode) in the editable fields.

Then you only have to fill in a unique marking in the **Marking** filed and either check the **Use system generated barcode** checkbox, or uncheck it and type in a barcode in the **Barcode** field to be able to save a new asset.

The **Save** button will be visible so the user can save the new asset. If the information is saved successfully, the page is reset and the new asset will show up in the table of **Existing database records.**

9.4 Admin page – Asset type groups table selected

The AssetTypeGroups table holds information about asset groups. The administrator can add, edit and delete an asset group on this part of the Admin page.



9.4.1 Add a new asset type group

First of all the asset group requires a name in the **Asset type group name** field.

It is optional to add new property data types to the group. There are two main types of properties; single-valued or multi-valued. See section 9.4.1.1 for adding a single-valued property data type and section 9.4.1.2 for adding a multi-valued property data type.

Press the **Save** button to save the new asset group. If the information is saved successfully, the page is reset and the new asset group will show up in the table of **Existing database records.**

9.4.1.1 Add a single-valued property data type

To add a single-valued property data type follow the steps below:

1. Check the **Is required** checkbox if the user must fill in or update this property when checking in an asset of this asset group.

2. Type in a name of the property type in the **Data type name** field.

3. Type in a value in the **Data value** field that will act as a default value for the assets of this group. If you do not want to give the property a default value, leave this field empty.

4. Leave the **Has range of values** unchecked to indicate that this is a single-valued property.

5. Press the **Add data type** button and the property data type will show up in a table on the right if the action succeeds.

9.4.1.2 Add a multi-valued property data type

To add a multi-valued property data type follow the steps below:

1. Check the **Is required** checkbox if the user must fill in or update this property when checking in an asset of this asset group.

2. Type in a name of the property type in the **Data type name** field.

3. Check the **Has range of values** unchecked to indicate that this is a multi-valued property. Three new buttons and a list box will be visible below the checkbox. As a rule of the system, a pre-defined value called “Unknown” will automatically be one of the range values. This value cannot be removed from the range of values.

4. Type in a value in the **Data value** field that will one of the range values.

5. Press the **Add** button to see the value appear in the list box and clear the data value field.

   Select a value in the list box and press the **Edit** button to be able to edit a range value. The selected value will show up in the editable data value field. The add, edit, remove buttons will be replaced by an update button and a cancel button. Press either the **Update** or the **Cancel** button to finish the edit action.

   Select a value in the list box and press the **Remove** button to delete a value from the range.

6. Repeat step 4 – 5 until the set of range values is completed.

7. Press the **Add data type** button and the property data type will show up in a table on the right if the action succeeds.

9.4.1.3 Edit an existing property data value of an asset type group

Press the **Edit** link on the row for the property data type you want to edit.

The available information about the property data type will be displayed in the editable fields to the left on the page. You can then edit the selected property data type depending on if it is a singled-value or a multi-valued property.

If it is a single-valued property data type you can:

- Change if the property must be updated when checking in an asset of this asset group, by checking/un-checking the **Is required** checkbox.
- Change the name of the property data type in the **Data type name** field
- Change the default value by editing the text in the **Data value** field
- You can ***NOT*** change it to a multi-valued property data type

If it is a multi-valued property data type:

- Change if the property must be updated when checking in an asset of this asset group, by checking/un-checking the **Is required** checkbox.
- Change the name of the property data type in the **Data type name** field
- Change the range of values by using the **Add** button to add new values to the range, using the **Edit** button to edit existing range values and using the **Remove** button to delete values from the set of range values.
- You can ***NOT*** change it to a single -valued property data type

Press the **Update** button to finish the edit action of the selected property data type. If you select the **Cancel** button, the edit action will revert, without changing anything.

The edited property data type will not be stored in the database until the **Update** button, related to the edit mode of the selected asset group, has been pressed.

If the **Cancel** button, related to the edit mode of the selected asset group, is pressed instead, the page will be reset and no changes will be stored in the database.

9.4.1.4 Delete an existing property data value of an asset type group

Press the **Delete** link on the row for the property you want to delete. A confirmation pop-up will show up, and prompt the user for a “Yes” or “No” to answer the question: “Are you sure you want to delete the record?”.

If you click “Yes” the property data type will disappear from the table. If you click “No” the delete action will be canceled.

The property data type will not be removed from the database until the **Update** button, related to the edit mode of the selected asset type group, has been pressed.

If the **Cancel** button, related to the edit mode of the selected asset type group, is pressed instead, the page will be reset and no changes will be stored in the database.

9.4.2 Edit an asset type group

Press the **Edit** link and the stored information of the selected asset type group will be filled into the editable fields. You can then edit most of the information in the fields except of switching between a single-value and a multi-value property data type.

Edit and delete the existing property data values by using the **Edit** and **Delete** links in the property table (if the asset type group has any added properties). See chapter 9.4.1.3 for editing properties and 9.4.1.4 for deleting properties.

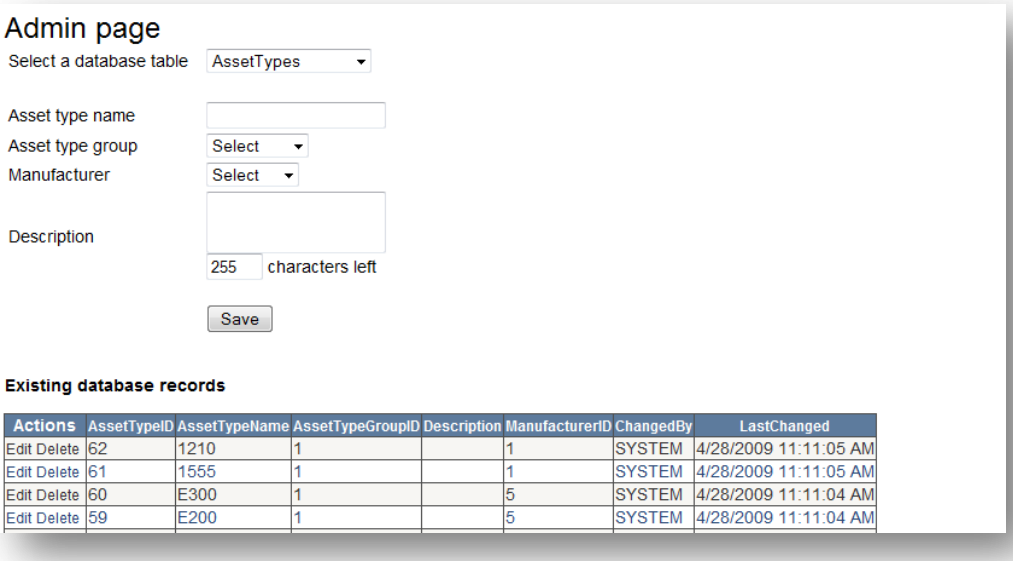
Press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.4.3 Delete an asset type group

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear from the **Existing database records** table.

9.5 Admin page – Asset types table selected

The AssetTypes table holds information about asset types. The administrator can add, edit and delete an asset type on this part of the Admin page.



Actions	AssetTypeID	AssetTypeName	AssetTypeGroupID	Description	ManufacturerID	ChangedBy	LastChanged
Edit Delete	62	1210	1		1	SYSTEM	4/28/2009 11:11:05 AM
Edit Delete	61	1555	1		1	SYSTEM	4/28/2009 11:11:05 AM
Edit Delete	60	E300	1		5	SYSTEM	4/28/2009 11:11:04 AM
Edit Delete	59	E200	1		5	SYSTEM	4/28/2009 11:11:04 AM

9.5.1 Add a new asset type

The asset type requires a name in the **Asset type name** field.

It also requires an asset type group to belong to. Use the **Asset type group** list to select an asset type group. If you do not find any suitable asset type group, go to the asset type group section of the Admin page and add a new one.

The asset type group needs a manufacturer. Select one from the **Manufacturers** list.

It is optional to add a description of the asset type in the **Description** field. The description is limited to max 255 characters. There is a character counter below the text field to help you to keep the description length within the limit.

Press the **Save** button to save the information about the new asset type. The system will then verify and make sure no required information is missing before it tries to save it in the database. The system will notify the user about the outcome. If the save action succeeds the page will be reset and the new asset type will appear in the **Existing database records** table. Otherwise the user will have a chance to correct the information and try to save the information again.

9.5.2 Edit an asset type

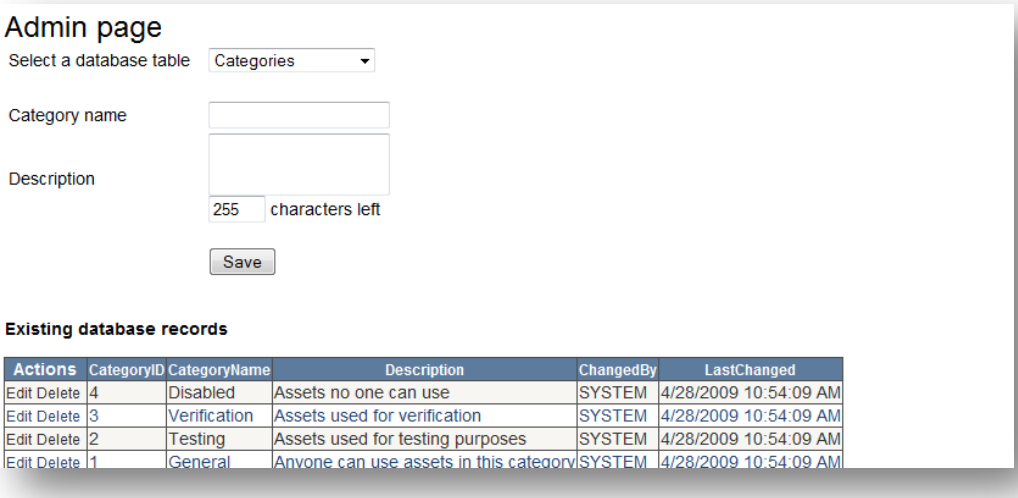
Press the **Edit** link and the stored information of the selected asset type will be filled into the editable fields. You can then edit the information in the fields and press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.5.3 Delete an asset type

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear in the **Existing database records** table.

9.6 Admin page – Categories table selected

The Categories table holds information about categories. The administrator can add, edit and delete a category on this part of the Admin page.



9.6.1 Add a new category

The new category needs a name. Give it a name by typing the name in the **Category name** field.

It is optional to add a description of the category in the **Description** field. The description is limited to max 255 characters. There is a character counter below the text field to help you to keep the description text within the limit.

Press the **Save** button to save the new category. The system will then try to save the information in the database. The system will notify the user about the outcome. If the save action succeeds the page will be reset and the new manufacturer will appear in the **Existing database records** table. Otherwise the user will have a chance to correct the information and try to save the information again.

9.6.2 Edit a category

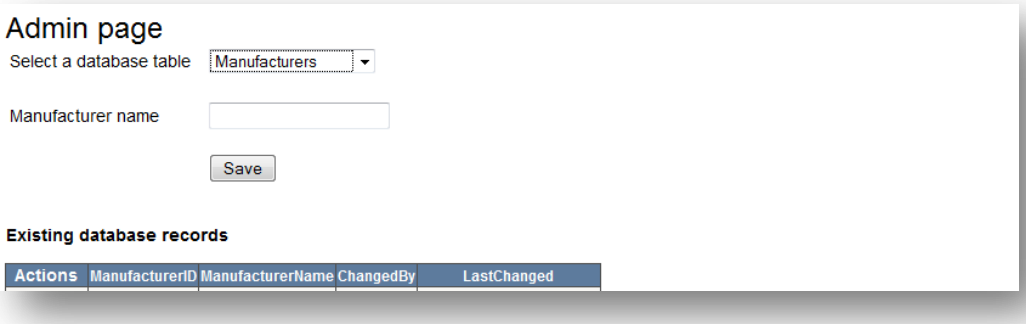
Press the **Edit** link and the stored information of the selected category will be filled into the editable fields. You can then edit the information in the fields and press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.6.3 Delete a category

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear in the **Existing database records** table.

9.7 Admin page – Manufacturers table selected

The Manufacturers table holds information about manufacturers. The administrator can add, edit and delete a manufacturer on this part of the Admin page.



9.7.1 Add a new manufacturer

To be able to add a new manufacturer, you first give it a name in the **Manufacturer name** field and then press the **Save** button.

The system will then try to save the information in the database. The system will notify the user about the outcome. If the save action succeeds the page will be reset and the new manufacturer will appear in the **Existing database records** table. Otherwise the user will have a chance to correct the information and try to save the information again.

9.7.2 Edit a manufacturer

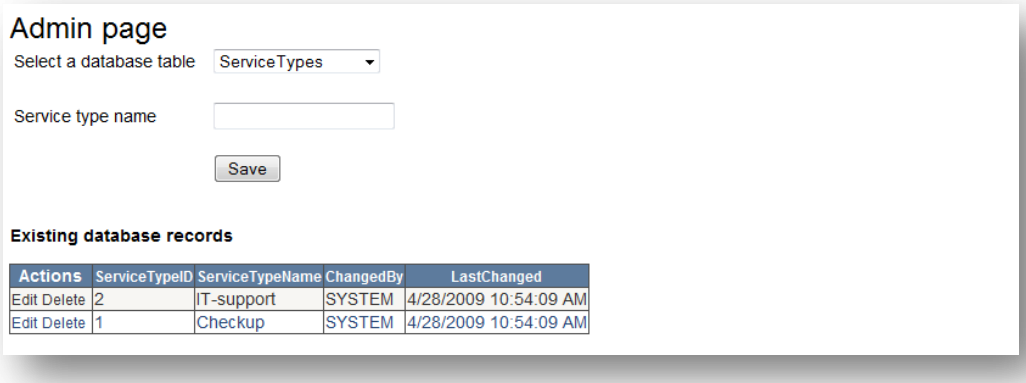
Press the **Edit** link and the stored information of the selected manufacturer will be filled into the editable fields. You can then edit the information in the fields and press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.7.3 Delete a manufacturer

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear from the **Existing database records** table.

9.8 Admin page – Service types table selected

The ServiceTypes table holds information about service types for the assets. The administrator can add, edit and delete a service type on this part of the Admin page.



9.8.1 Add a new service type

To add a new service type, first give it a name in the **Service type name** field and then press the **Save** button.

The system will then try to save the information in the database. The system will notify the user about the outcome. If the save action succeeds the page will be reset and the new service type will appear in the **Existing database records** table. Otherwise the user will have a chance to correct the information and try to save the information again.

9.8.2 Edit a service type

Press the **Edit** link and the stored information of the selected service type will be filled into the editable fields. You can then edit the information in the fields and press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.8.3 Delete a service type

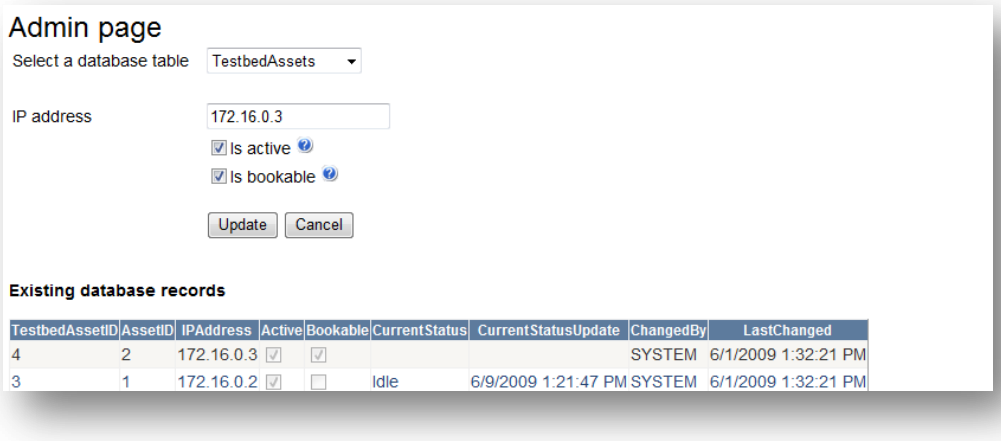
Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear in the **Existing database records** table.

9.9 Admin page – Testbed assets table selected

The TestbedAssets table holds information about assets in the testbed. The administrator can edit an asset in testbed on this part of the Admin page. If a user wants to add or remove an asset from the testbed, this can be done on the assets part of the Admin page.

9.9.1 Edit an asset in testbed

To edit the information of an asset in the testbed, press the **Edit** link. A new section will be visible on the page, filled in with the stored testbed related information of the selected asset.



You can change the IP-address of the asset by typing in a new one in the **IP-address** field.

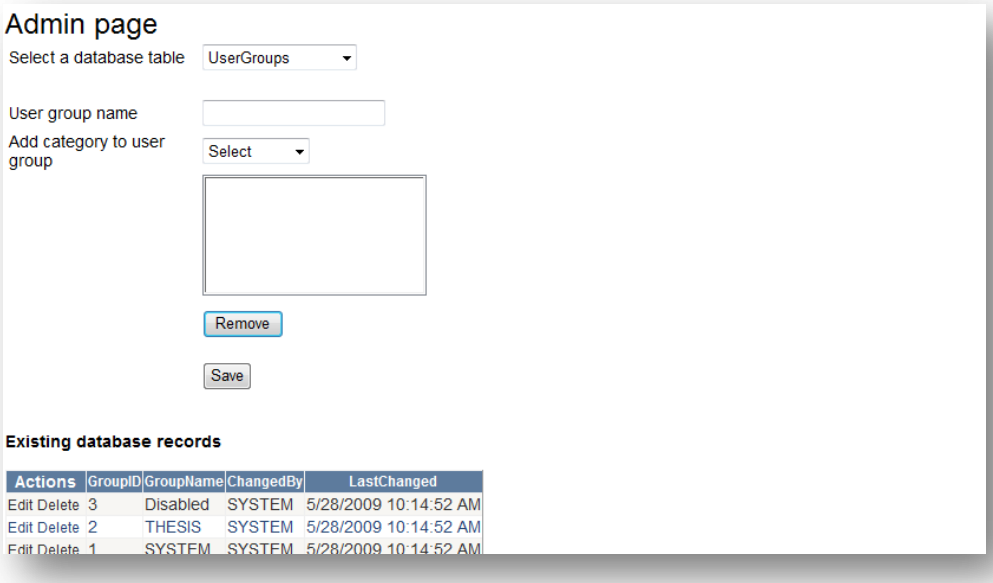
Check the **Active** checkbox if the testbed asset is connected to the network and ready for test execution.

Check the **Bookable** checkbox if the testbed asset can be booked by users for exclusive use over a period of time.

Press the **Update** button to save the changes or press the **Cancel** button to regret the edit action and reset the page. If the update action succeeds, the page is reset.

9.10 Admin page – User groups table selected

The UserGroups table holds information about groups of users. The administrator can add, edit and delete a user group on this part of the Admin page.



9.10.1 Add a new user group

Give the new user group a name by typing it into the **User group name** field.

To limit the assets in the system to be visible to different users, the assets and the user groups are related to categories. Use the **Add category to user group** list to select a category that the new user group will have access to. As a category is selected in the list, it will show up in the list box below and disappear from the list. If you want to remove a category from a user group, simply select it in the list box and press the **Remove** button. The selected category will then disappear from the list box and appear in the list.

Press the **Save** button to save the information about the new user group. The system will then verify and make sure no required information is missing before it tries to save it in the database. The system will notify the user about the outcome. If the save action succeeds the page will be reset and the new user group will appear in the **Existing database records** table. Otherwise the user will have a chance to correct the information and try to save the information again.

9.10.2 Edit a user group

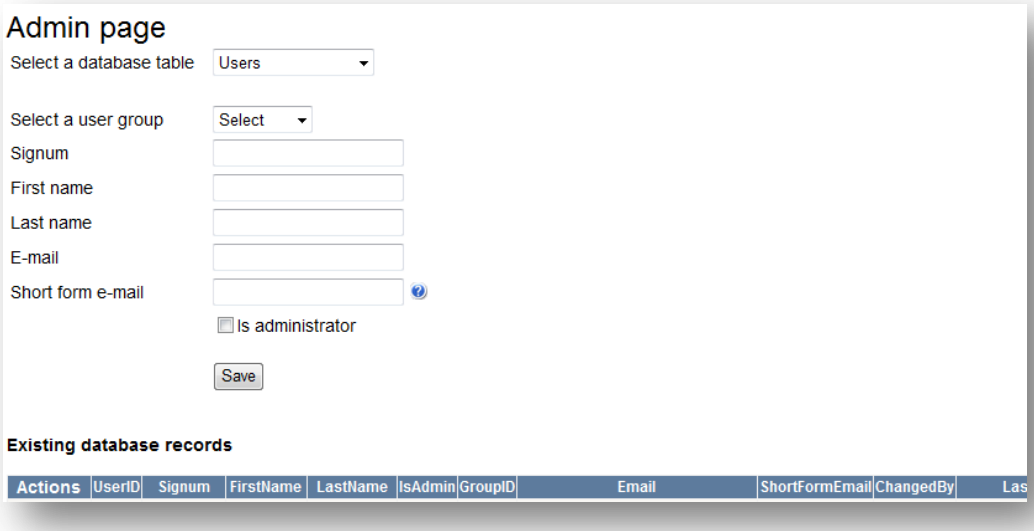
Press the **Edit** link and the stored information of the selected user group will be filled into the editable fields. You can then edit the information in the fields and press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.10.3 Delete a user group

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear from the **Existing database records** table

9.11 Admin page – Users table selected

The Users table holds information about users in the system. The administrator can add, edit and delete a user on this part of the Admin page.



9.11.1 Add a new user

A user needs to belong to a user group. Use the **Select a user group** list to select a suitable group for the new user. If you cannot find any suitable group, go to the user group part of the Admin page and add one.

All users need a unique signum, which shall be filled into the **Signum** field. Use the Ericsson signum for employees at Ericsson.

Fill in the first name of the user in the **First name** filed and the last name in the **Last name** field. These two fields are optional and can be left empty.

A user must have an e-mail address; use the **E-mail** field for this.

Use the **Short form e-mail** field to add an e-mail address used to send SMS to the user. This field is optional.

Check the **Is administrator** checkbox to indicate that the new user shall have administrator privileges.

Press the **Save** button to save the information about the new user. The system will then verify and make sure no required information is missing before it tries to save it in the database. The system will notify the user about the outcome. If the save action succeeds the page will be reset and the new user will appear in the **Existing database records** table. Otherwise the user will have a chance to correct the information and try to save the information again.

9.11.2 Edit a user

Press the **Edit** link and the stored information of the selected user will be filled into the editable fields. You can then edit the information in the fields and press the **Update** button to save the changes or press the **Cancel** button to regret the edit action. If the update action succeeds, the editable fields will be cleared and the lists will be reset.

9.11.3 Delete a user

Press the **Delete** link and wait for the system response. If the delete action succeeds the record will be removed from the database and disappear from the **Existing database records** table

10 Edit the configuration file

There is a configuration file (web.config) to make it simple to change static values in the TIME system. The file is stored in the web root folder.

The settings file is an xml document that contains lots of configuration information for the web portal. Most of the file is generated by the system itself. There are only three sections in the file that is necessary for the user to know in details and how to change it. These sections are; appSettings, connectionStrings and mailSettings and are described in the sections below.

10.1 The <appSettings>

The appSettings contains the following key value pairs:

```
<add key="CheckOutTime" value="120"/>
<add key="MaxBookingDurationHours" value="72"/>
<add key="DefaultBookingDurationHours" value="48"/>
<add key="KeepExportedExcelFilesOnDiskHours" value="48"/>
<add key="ExportedExcelBasePath" value="C:\TeaTime\ExcelFiles\"/>
<add key="BookingPurposes" value="GUI testing;Manual testing;Service"/>
<add key="TestingPurposes" value="Long-time test;Regular test"/>
<add key="StatusValues" value="Status1;Status2;Status3;Unknown"/>
<add key="LocationValues" value="LN;LN-Temp;LN-Testsys;LN-Cert;NonLN"/>

<add key="DefaultMinutesToWaitIfPlannedOutage" value="5"/>
<add key="TestFileUploadPath" value="C:\Teatime\Tests\"/>
<add key="EmailFromAddress" value="no-reply-teatime@ericsson.se"/>
<add key="WatchdogHoursBeforeEmailResend" value="48"/>
<add key="WatchdogEmailUserIfLateReturnHours" value="24"/>
<add key="WatchdogEmailOwnersIfLateReturnHours" value="48"/>
<add key="WatchdogCancelBookingIfNotCheckedOutHours" value="24"/>
<add key="BaseUrl" value="http://localhost:53287/WebSite/"/>
<add key="BarcodeLabelTemplatePath"
value="C:\\TeaTime\\TimeWISupport\\TEATIME_SMALL.LWL"/>
```

The “CheckOutTime” value defines how many minutes before a booking starts it will be possible for the user to check out assets part of said booking.

The “MaxBookingDurationHours” value defines how long (hours) a booking is allowed to be before an administrator will be notified.

The “DefaultBookingDurationHours” value defines simply the default duration (hours) of a booking, which is shown in the calendars and the editable text fields for time when a user enters the Booking information page.

The “KeepExportedExcelFilesOnDiskHours” value tells the system how many hours it has to store the exported excel files on the disk before they can be deleted.

The “ExportedExcelBasePath” value defines the file path to the default location of the exported Excel files.

The “BookingPurposes” value is a set of purposes from which the user can select a value when placing a booking. The different alternatives are separated by a semicolon (;).

The “TestingPurposes” value is a set of purposes from which the user can select a value when scheduling an automatic test. The different alternatives are separated by a semicolon (;).

The “StatusValues” value is a set of status values from which the user can select a value when setting the status of an asset. The different alternatives are separated by a semicolon (;).

The ”LocationValues” value is a set of locations from which the administrator can select a value when adding/editing an asset and when a user selects a location when searching for assets.

The “DefaultMinutesToWaitIfPlannedOutage” value defines the default number of minutes to wait when a SUT has planned outage before it is time to notify the user that the SUT has been disconnected too long.

The “TestFileUploadPath” value specifies the file path location for uploaded test files on the Autotest page.

The ”EmailFromAddress” value specifies the from e-mail address when the system sends emails to users.

The “WatchdogHoursBeforeEmailResend” value defines the time (hours) for the watchdog to wait before it resends an email to a user.

The “WatchdogEmailUserIfLateReturnHours” value defines the time (hours) that the watchdog shall wait when a user has not returned an asset before it sends an email to remind the user.

The “WatchdogEmailOwnersIfLateReturnHours” value defines the time (hours) that the watchdog shall wait when an asset is not returned in time before it emails the owners of the late asset.

The “WatchdogCancelBookingIfNotCheckedOutHours” value defines the time (hours) for which a booking shall be cancelled by the watchdog if the user has not checked it out yet.

The “BaseUrl” value specifies the base url for the pages in the TIME web interface.

The “BarcodeLabelTemplatePath” value specifies the file path to the location of the template of the barcode label.

10.2 The <connectionStrings>

The “connectionStrings” part contains the following connection strings:

```
<add name="TeatimeAdmin"
connectionString="Data Source=.\SQLEXPRESS;Database=Teatime;User
Id=teatime_admin;Password=**********"
providerName="System.Data.SqlClient"/>

<add name="TeatimeSelect"
connectionString="Data Source=.\SQLEXPRESS;Database=Teatime;User
Id=teatime_select;Password=**********"
providerName="System.Data.SqlClient"/>

<add name="TeatimeUser"
connectionString="Data Source=.\SQLEXPRESS;Database=Teatime;User
Id=teatime_user;Password=**********"
providerName="System.Data.SqlClient"/>

<add name="ResultSelect"
connectionString="Data Source=.\SQLEXPRESS;Database=Result;Integrated
Security=True"
providerName="System.Data.SqlClient"/>

<add name="MetricsSelect"
connectionString="Data Source=.\SQLEXPRESS;Database=metrics11;Integrated
Security=True"
providerName="System.Data.SqlClient"/>
```

The “TeatimeAdmin” connection string is used when an administrator connects to the database. The administrator has the privileges to insert / update / delete / select on all tables in the Time DB.

The “TeatimeSelect” connection string is used when a user connects to the database to execute a select query statement.

The “TeatimeUser” connection string is used when a user connects to the database to execute an insert / update / delete statement on a set of pre-defined tables.

The “ResultSelect” connection string is used for connection to the Result DB. The system uses this connection string when it fetches data from test results.

The “MetricSelect” connection string is used when a user enters the ResultPlanningDB to schedule an automatic test.

10.3 The <mailSettings>

The “mailSettings” contains information about the SMTP server:

```
<smtp deliveryMethod="Network">
      <network host="se-smtp.domain.tld" port="25"
      defaultCredentials="true"/>
</smtp>
```

11 Troubleshooting

11.1 I'm using Firefox and I have to log in every time I visit the page

The web interface uses Integrated Windows authentication to get the signum of the visiting user. Firefox does not send these credentials by default. Luckily, there is a setting to change this.

1. Browse to about:config
2. Find the preference "network.automatic-ntlm-auth.trusted-uris"
3. Add the server address or simply ".ericsson.se"
4. Close about:config
5. Firefox should now provide your windows credentials to TIME the same way Internet Explorer does

For more information, see http://markmonica.com/2007/11/20/firefox-and-integrated-windows-authentication/